EUSTACE System Design

Joel R. Mitchelson

*Revisions*

|  |  |  |
| --- | --- | --- |
| **Date** | **Author** | **Comment** |
| 19/01/2016 | Joel Mitchelson | Created. |
| 26/01/2016 | Joel Mitchelson | Update to include interfacing, module list. |
| 14/03/2016 | Joel Mitchelson | Changed ‘available in SVN’ -> ‘maintained in SVN’ |
| 12/05/2016 | Joel Mitchelson | Cross out unknown regarding definition of a day (now agreed as PRDREQ015). Add section on system control architecture. |
| 22/09/2016 | Joel Mitchelson | Adjust source code header traceability requirement to allow ‘origin’ as alternative to author, in case external library code is required. Add new control architecture requirement for dealing with system downtime.  System library module list updated.  EUMOPPS specified as name for control architecture. |
| 22/01/2018 | Joel Mitchelson | Draft update to describe principles of advanced standard infilling. |
| 23/03/2018 | Joel Mitchelson |  |
| 11/04/2018 | Joel Mitchelson |  |
| 29/09/2018 | Joel Mitchelson | Update module list. Mention of LSF as well as Rose. |
| 19/09/2018 | Joel Mitchelson | Restructure to allow clear linkages with science documentation via appropriate UML diagrams. |
| 20/09/2018 | Joel Mitchelson | First draft of UML activity diagrams for EUMOPPS. |
| 25/10/2018 | Joel Mitchelson | Add draft activity diagrams for full system. |
| 29/01/2019 | Joel Mitchelson | Revsions to UML diagrams following Met Office feedback. |
| 30/01/2019 | Joel Mitchelson | Further revision to connect control architecture parameters to the actions that use them. |
| 26/02/2019 | Joel Mitchelson | Update diagrams in preparation for adding advanced standard in consistent way. |
| 29/03/2019 | Joel Mitchelson | Rearranged satstace and infill sections to have consistent headings.  Activity diagrams for satstace, and top level of infill. Descriptive text which relates diagrams to methods.  Removed project unknowns list (now resolved).  Moved computational strategy ideas which didn’t make it into final system to separate section at the end. |

# Introduction

This working document describes the design of the EUSTACE end-to-end system for product generation, to address the requirements set out in the EUSTACE system technical requirements document [1]. Since requirements include statements about desired system quality and maturity, suitable development processes to address these are also specified.

With reference to the description of action given in Annex 1 of the EUSTACE grant agreement [2], this document forms part of project deliverable D2.5: “Coded and tested system for product generation”, and was begun as part of project tasks T2.1 “Product Design” and T2.4 “System development”.

# System Overview

The system comprises a suite of software modules which can be run on the command line on the CEMS system (SYSREQ001, SYSREQ005).

There is at least one unique module corresponding to each of the processing blocks identified in figures 1 and 2 of the requirements (top-level schematic; detail of in-situ processing).

In addition, there is a suite of modules whose purpose is to provide quality checks of the data at key stages of processing. This is necessary so that output is sufficiently correct for validation (SYSREQ004) and to ensure that the system as a whole is of appropriate maturity (SYSREQ007). This suite includes the satstace and fullstace quality checks shown in figure 1 of requirements, plus tests of code and intermediate results introduced in section 3.

Some processing blocks use novel methods developed by project partners as set out in the description of action for WP1 and WP2 [2]. Collaboration with relevant partners will be undertaken to ensure that the implementation of these methods meets the quality criteria. The project partner with the relevant scientific expertise to certify that a software module complies with their published method is referred to here as the **science owner**.

# Correctness

SYSREQ007 requires appropriate maturity for potential production of products beyond the lifetime of the project, and SYSREQ004 requires appropriate output to pass validation checks. Both of these impose requirements for correctness.

Table 1 shows risk factors for errors in outputs, and mitigating actions which form part of process or product design.

Table 1: Correctness risk and mitigation

|  |  |
| --- | --- |
| **Risk factor** | **Mitigating actions** |
| Input data sets missing | Automated check against data management plan. |
| Input data misidentified or incomplete | Input data summary for manual inspection, automatically checked against input when downstream module is run. |
| Uncontrolled changes to input data | Compute and store checksum under version control; checksum verified when downstream module is run. |
| Uncontrolled changes to intermediate/output data | Version number stored in version control, written into intermediate/output data sets, verified after creation, and by downstream modules. |
| Corruption of intermediate data | Compute and store checksum under version control; checksum verified when downstream component is run. |
| Corruption of output data during storage or transmission | Compute and store checksum, and provide simple script for verification against this. |
| Documentation on file formats may not correspond to actual formats output, due to human error | Automated check against relevant file format document or CF standard. |
| Processing code may not correspond to scientific documentation due to human error | Regression test of module output against an exemplar data set that has been approved by science owner. Use of function-level unit tests encouraged.  Ideally: code made open source for peer review. |
| System code (including checks) may not perform correctly, due to human error | Acceptance tests will be created for all top-level code and at function level where possible. |
| Mitigating actions may not be taken | Checklist of mitigating actions verified by software engineer. |

# Science modules

## Interfacing

The interface between top-level science modules is by file input/output rather than shared memory. Files are written to disk by one module and read by the next. There are several reasons for this architecture:

* System data required at certain intermediate steps is the same as required for WP1 products, and so reuse of WP1 code is possible.
* Storage of data from intermediate steps allows additional quality-checking routines to be run.
* Avoids the development overhead in exposing programming interfaces to each module.
* Reduced computational resource requirement by allowing selective update of a subset of computations when only a subset of input data has changed or a sub-part of the system code is updated.
* Provides robustness to host system downtime.

## Science modules

A list of modules corresponding to figures 1 and 2 of the technical requirements document [1] is given below.

|  |  |  |  |
| --- | --- | --- | --- |
| **Title** | **Science owner** | **Input source** | **Output format** |
| Break detection over Europe | UBERN | TBC | EUSTACE station data product |
| Global break detection | UBERN | TBC | EUSTACE station data product |
| Homogenisation over Europe | UBERN | TBC | EUSTACE station data product |
| Pre-process in situ sea surface air temperature | Met Office | TBC | EUSTACE global field product |
| Relationship model in situ sea surface | Met Office | From: *Pre-process in situ sea surface air temperature* ; + model parameters TBC | EUSTACE global field product |
| Relationship model in situ ice | DMI | From: *Pre-process in situ ice surface air temperature* ; + model parameters TBC | EUSTACE global field product |
| Relationship model in situ lakes | UREAD | From: *Pre-process in situ lake surface air temperature* ; + model parameters TBC | EUSTACE global field product |
| Append uncertainty to satellite sea surface measurements | UREAD | TBC | CF NetCDF |
| Append uncertainty to satellite land surface measurements | ULEIC | TBC | CF NetCDF |
| Append uncertainty to satellite ice surface measurements | DMI | TBC | CF NetCDF |
| Append uncertainty to satellite lake surface measurements | UREAD | TBC | CF NetCDF |
| Relationship model satellite sea surface | Met Office | From: *Append uncertainty to satellite sea surface measurements*;  + model parameters TBC | EUSTACE global field product |
| Relationship model satellite land surface | Met Office | From: *Append uncertainty to satellite land surface measurements +* model parameters TBC | EUSTACE global field product |
| Relationship model satellite ice | DMI | From: *Append uncertainty to satellite ice surface measurements*  + model parameters TBC | EUSTACE global field product |
| Relationship model satellite lakes | UREAD | From: *Append uncertainty to satellite ice surface measurements*;  + model parameters TBC | EUSTACE global field product |
| satstace product generation | Met Office | From: *Relationship model satellite sea surface*; *Relationship model satellite land surface*; *Relationship model satellite ice*; *Relationship model satellite lakes*. | EUSTACE global field product |
| Infill | Met Office UBATH | From: *Global break detection*; *Homogenisation over Europe; Pre-process in situ sea surface air temperature*; *Relationship model in situ sea surface*; *Relationship model in situ land surface*; *Relationship model in situ ice*; *Relationship model in situ lakes*; *Relationship model satellite sea surface*; *Relationship model satellite land surface*; *Relationship model satellite ice*; *Relationship model satellite lakes*; + direct inputs from measurements TBC + model parameters TBC | EUSTACE global field product |
| satstace quality check | STFC | From: *satstace product generation* | TBC |
| fullstace quality check | STFC | From: *Infill* | TBC |

# Index of top-level modules

Cross-reference of all modules with their role in the system

|  |  |
| --- | --- |
| **Module** | **Role** |
| Eumopps | The EUMOPPS system implementing the control architecture as described in section 8 below. |
| Analysis | Infilled analysis. |
| Inputcheck | Automated check of data sets against data management plan (section 3). |
| Inputsources | Summarise input data for manual inspection (section 3). |
| Outputformats | Produce outputs compliant with EUSTACE product formats including automated verification against the defining document [3]. |
| Parsedoc | Read MS Word format documents for cross-verification. |
| Preprocess | Read in-situ data preprocessed to include uncertainty information and breakpoints. |
| Satgrid | Regrid satellite data, as used by science modules. |
| satgrid\_iris | As satgrid, but using Met Office Iris library. |
| Surfaceairmodel | Compute surface air temperature derived from satellite. |
| Timeutils | Utilities for dealing with time, including definition of the EUSTACE epoch. |

# File formats

File formats for EUSTACE global field products and EUSTACE station data products are described in the initial recommendations on data formats and management [3].

System configuration files use the JSON convention [4].

# Development Process

## Source control

All project code will be kept under source control, to satisfy SYSREQ003. Research code (including code from science owners described in section 7.1) will be placed in the SVN repository in subdirectories of:

<http://proj.badc.rl.ac.uk/svn/eustace/platform/trunk/research>

Creation of a subdirectory here will be documented on the project Wiki.

Once the code is ready to qualify for inclusion in the system, WP2 will place it in:

<http://proj.badc.rl.ac.uk/svn/eustace/platform/trunk/system>

## Traceability to science owners

The correctness requirement described in section 3 is interpreted to mean that computations must be traceable back to the relevant science owner. This facilitates traceability to peer-reviewed literature if the science owner publishes the method. The following process is used as a minimum:

1. Science owner produces example computation code and output which meets a minimal set of system requirements:
   1. Can be run from the CEMS command line
   2. Code maintained in SVN
   3. Source code headers state author / origin and SVN revision keyword
   4. Usage instructions maintained in SVN
   5. Output format documentation maintained in SVN
   6. Details made available on project Wiki
2. Science owner asserts that they are satisfied that example code and output corresponds to the method in their project reports and any associated publications, and records this on the project Wiki.
3. WP2 extends the example code to meet all other system requirements (such as modularity, speed of processing, whole earth coverage), creates cross-comparison code to verify against the example, and makes this available in SVN.
4. Science owner reviews verification test code to confirm that it is sufficient, and records their approval on the project Wiki.

## Developer guide

Information of a practical nature required for developers to maintain and extend the system are described in a developer guide. This is written using RST (ReST) files suitable for use with the sphinx auto-documentation tool [5]. The underlying content is stored alongside the code in the SVN repository. Both HTML and PDF formats are rendered using sphinx and made available in a directory on the host system.

The contents should include descriptions of:

* Required development environment
* Steps required to obtain the code
* Programming languages used
* Conventions for directory structures
* Coding style guide
* Guidelines for the development process / developer workflow
* Approach to testing and source comments and documentation
* API reference for the system (automatically generated using sphinx)

The developer workflow and approach to testing is consistent with SYSREQ003 (code is sufficiently correct for cross-validation checks) and SYSREQ007 (appropriate maturity for potential production of products beyond the lifetime of the project).

# Control Architecture

## Overview

The architecture for controlling the system is called EUMOPPS (**EU**STACE **M**et **O**ffice **P**rocessing and **P**rovenance **S**ystem). This addresses the correctness requirements (above). Its purpose is to provide a means of running the system with the following properties:

* Input data locations and filename patterns can be specified in a compact, human-readable form
* A human-readable report of available inputs is produced for manual inspection
* This allows the user to identify rogue (extra) inputs and missing data
* The necessary list of calculations to perform is determined from the inputs
* The range of inputs (e.g. in time) can be limited by configuration options
* It is possible to determine whether, in normal operation (i.e. in the absence of disk corruption, tampering, etc) the outputs are currently up-to-date with respect to the specified inputs
* During development, it is possible to skip calculations whose outputs are up-to-date with respect to the specified inputs (also assuming normal operation)
* For a given output, it is possible to determine (and if necessary reproduce) the steps taken to make it
* The progress of time-consuming operations can be monitored; and paused/continued if necessary
* If the host system suffers downtime during a sequence of operations, only those which have not already completed need to be re-run.

The process to meet these criteria is as follows:

* User creates a JSON file to specify required inputs by filename pattern containing a number of fields
* Scan the directory to generate a listing of all files matching the pattern for each data source; including version and checksum information from each. This catalogue file is stored on disk in a binary format (actually NetCDF) for rapid access.
* Read the catalogue file and produce a readable report listing max/min/count for each field in the filename patterns
* Produce a listing of operations required to generate outputs from inputs, including version, checksum, and code version expected. A configuration file may be used to control this [perhaps multiple configuration files available, in which case would need to embed the content of the config file used in the operation listing, so that we can see what was requested]
* Use the rose or LSF utility to perform the operations and output NetCDF files, including a source attribute which lists inputs.
* Using the listing of operations, produce a catalogue of resulting outputs, including their own versions and checksums and versions/checksums of the inputs they depend upon
* Use the resulting outputs to produce a new listing of derived calculations; etc.

The basic idea behind this system can be sketched as follows:

Build

Run

Seal

[ extend ]

[ complete ]

Where the meaning of each activity is as follows:

Build

Generate or extend a **catalogue file**. This contains a list of input filenames, checksums, commands to run, and expected output filenames.

Run

Run the commands specified in the catalogue file by scheduling to run on a parallel compute resource (using rose or LSF) and store catalogue ID and code revision number in all output files.

Seal

Check that expected outputs are present, with expected catalogue ID, and write their checksums into the catalogue file too.

An existing catalogue can be extended by repeating these operations.

A more detailed design is shown in section 8.2. The diagram assumes that the user chooses the same catalogue filename throughout and hence a single catalogue file in the file system is accessed and modified.

## Control architecture to run EUMOPPS using a given description file

Run operation

Build catalogue

description filename

catalogue filename

Issue seal command

Description file  
<< data store >>

Catalogue file  
<< data store >>



Launch operation tasks

Issue build command

[ more operations ]

Issue run command

operation name

catalogue filename

catalogue filename

Seal catalogue

[ more operations ]

Check complete

[ ok ]

System operator

Host computer

Parallel compute cluster

File system

Version control

Results files  
<< data store >>

Revision number

Revision number

Input data files  
<< data store >>

[ error]

The build catalogue activity is further detailed in section 8.3.

## Control architecture to build a EUMOPPS catalogue from description file

Receive build command

description filename

catalogue filename

Read description file

Catalogue  
[ No details of input or output files ]

Search for data inputs matching description

Catalogue  
[ Details of input but not output files ]

Process list of operations

Catalogue  
[ Latest details of all input and output files but not all output checksums ]

Store results

Description file  
<< data store >>



update : yes/no

Append to existing catalogue

Catalogue file  
<< data store >>



Catalogue  
[ Details of input and existing details of output files ]

[update]

[no update]

Build catalogue

Input data files   
<< data store >>



# Activity diagram for EUSTACE system

Activities of the complete system are shown in the following diagram. This corresponds to the data flow described in the technical requirements, and the subsystems defined there as ‘satstace’ and ‘fullstace’.

Pre-process satellite

Relationship models (satellite)

Infill

Pre-process in-situ

In-situ sources  
<< data store >>

In-situ sources with uncertainty and breakpoint information  
<< data store >>   


Satellite sources  
<< data store >> 

**satstace**

Globally complete analyses  
<< data store >>   


**fullstace**

Relationship model parameters  
<< data store >> 

Model covariates  
<< data store >> 

Satellite with uncertainty information  
<< data store >> 

Surface air temperature derived from satellite  
<< data store >>  


For brevity the EUMOPPS catalogue files are omitted from this diagram and only observational input/output data is shown.

An instance of EUMOPPS is used for ‘Relationship models (satellite)’ and another for ‘Infill’.

# Input preprocessing

At the time of writing the pre-processing modules ‘Pre-process satellite’ and ‘Pre-process in-situ’ do not use the EUMOPPS architecture (but could in future be adapted to do so). The code remains in the form provided by the science owners, together with instructions for running it.

# Satellite-derived surface air temperature

## Theoretical concepts

There is a distinct surface-air relationship model for each surface type (ice, land, ocean). These models allow calculation of satellite derived surface air temperature on any specified day, given the corresponding day of pre-processed satellite surface temperature observations, together with relationship model parameters which are fixed. For the individual calculations required of each surface type, please refer to scientific methods documentation in TODO.

## Computational strategy

Given that the calculations for one day are independent of another, the method is trivially parallel by running calculations for each required day as separate processes on the JASMIN resource.

## Activity Diagram

For a given surface type, the relationship model calculation activity is as follows:

Relationship model  
{surface}  
{day *N}*

Relationship model  
{surface}  
{day 0}

Satellite with uncertainty information {surface} {day 0}  
<< data store >>

Satellite with uncertainty information {surface} {day *N}*  
<< data store >>

Relationship model parameters {surface}  
<< data store >>

Surface air temperature derived from satellite  
{surface} {day 0}  
<< data store >>

Surface air temperature derived from satellite  
{surface} {day *N*}  
<< data store >>

The activites “Relationship model {surface} {day 0}” are managed using the EUMOPPS system. In the EUMOPPS implementation of satstace these are the activities which take place in the run operation blocks of EUMOPPS that are depicted in the diagram of section 8.2.

# Advanced standard infilling

The correspondence between infilling calculations and the scientific method is described in section 12.1. The consideration given to computational resource when designing software to perform these calculations is described in sections 12.2 and the resulting interpretation as computational architecture to perform the infill activity is shown in section 12.3.

## Theoretical concepts

In the model described by Morice [6] the behaviour of a global surface air temperature field is written as:

and are space and time coordinates respectively, and is a vector of model parameters for which we wish to estimate a probability density given the available observations. The model function is carefully selected such that the statistics of real-world temperature fields can be represented.

Observations of this field taken at time and space are taken to follow an observation model of the form:

is observational error, and is the sum of observational biases affecting this observation. The probability density for is assumed known (this is provided in metadata with the input observations), but the posterior probability density for is unknown and must be estimated given the data.

The method follows a Bayesian formulation, and so begins with a prior probability density for values of and . These are then updated in the presence of observations to produce a posterior density. The prior density may also be refined according to examples of data, a process known as **hyperparameter estimation**. The final output stage involves sampling from the posterior density on a regular grid.

### Elements

In practice is modelled as a finite linear sum of **elements** which contribute to the temperature:

Each element describes a Gaussian process. Our prior assumption is that elements are independent of each other, but after updating with observations we may discover covariance between them. For examples of elements see below. To simplify the formulation, we note that the bias term can be expressed just like any other element by writing:

Where is the vector of all unknown biases, and is a matrix of ones and zeroes. There will be a 1 in the *i*th row and *j*th column of if and only if observation *i* is known to be affected by bias *j*. So now is modelled as:

And the observation model becomes:

Where one of those elements is .

### Components

We can choose to constrain a subset of elements to represent a Gaussian process which is independent of those represented by any other elements. This is a modelling assumption, and one which we may exploit for computational efficiency. A **component** is the name given to such a subset of elements. It can be computationally efficient to proceed in iterative fashion, updating the subset of model parameters  which correspond to a given component, and then holding these constant whilst other components are updated. In the advanced standard method there are three components: climatology, large-scale, and local.

### Design matrix and solution

The required output from the method is a precision matrix for the conditional distribution of model coefficients conditioned given the available observations, and the conditional mean . These are defined by the following equations:

Equation 1

is known as a **design matrix**. Each row of represents the model function evaluation at the corresponding observation location:

There are three main tasks to be considered in computational strategy:

1. Retrieval and simultaneous computation of the quantities on the right hand sides of Equation 1.
2. Solution for in linear system.
3. Update with new observations given an initial solution.

## Computational strategies

### Use of sparse matrix storage and operations throughout

Matrix sparsity and symmetry assumptions are key tools for performance efficiency by avoiding unnecessary arithmetic and storage/retrieval. The sparsity and symmetry structures should therefore be observed when storing and operating on these quantities. A method which permits efficient indexing such as **jagged diagonal storage** (JDS) is appropriate for storage of basis functions and priors.

### Use of block structure for parallel computation

We can exploit the block-diagonal structure of for parallel computation. Consider that there are blocks of which appear like:

Each block represents observational uncertainty for a subset of observations, and those observations are written as column vector with corresponding basis function evaluation rows stacked as matrix . And so:

And similarly:

Each element of the sums on the right hand side can be evaluated in parallel and requires only information about the *rth* block of observations (and corresponding basis function evaluations). In the case of EUSTACE the index *r* is used to represent a step in time. Note however that the prior does allow correlation between space and time. For this reason a component of this general kind is called a **space-time component**.

It will also be helpful to consider an extended special case of a component where the prior has a block structure which matches that of the observational uncertainty. This occurs in EUSTACE for the local component that models daily variation. The model variables are *a priori* independent from one day to the next. This results in a block diagonal where the index *r* corresponds to day number. This means that the posterior will also have block structure where block *r* is:

And hence the mean is given by a vector components that solve:

Thus subvectors of the system can be both evaluated and solved in parallel. Because this simplification occurs in EUSTACE when the prior does not vary in time, this kind of component is known as a **spatial component**.

### Disk-based storage for communicating intermediate results between processes

Parellisation of the kind described requires that results output by one process can be used as input to another. On the JASMIN host used to implement EUSTACE, disk-based storage has been optimised for speed and concurrency, and therefore this is used as the storage medium for precomputed quantities. MPI was considered but not used on this occasion due to complexity in implementing and fault-finding without gain in speed versus the file-based approach.

### Computational strategies for the linear system

Solution of sparse positive-definite symmetric linear systems falls into two main categories: direct (non-iterative) solutions and iterative ones. In this project only direct solutions were explored. Direct solutions which have been tried on CEMS include: open source cholmod using Intel MKL libraries as driver routines; Intel Sparse Direct Solver for Clusters; and MUMPS, Harwell Scientific Libraries. The Intel solver was selected as running in reasonable time and with a robust, well-documented API.

## Activity Diagrams

The overall flow of the **infill** activity can be summarised as follows:

Compute Component

System Configuration  
<< data store >>

Output Grid

Raw Binary Files  
<< data store >>

System State  
<< data store >>

Generate Input Files

In-situ sources with uncertainty and breakpoint information  
<< data store >>

Surface air temperature derived from satellite  
<< data store >>

Globally complete analyses  
<< data store >>

next component

next iteration

Model Covariates  
<< data store >>

The **Compute Component** activity corresponds to the **component** calculation as described in section 12.2.2. The most efficient choice for inner structure depends on whether the component is a **space-time component** or a **spatial component** according to the definitions from section 12.2.2.

TODO: activity diagram for space-time component.

TODO: activity diagram for spatial component.

## Further Strategies

### Precomputation of quantities that are used repeatedly

There are common cases in which spatial coordinates are identical across a collection of observations, and therefore component function evaluations and are the same for all observations in that collection. Examples include: an in-situ station that does not move; or multiple satellite-derived fields each on the same spatial grid. Efficiency could potentially be improved by precomputing these basis functions so that the calculation is done only once, and then multiple nodes can access them without recalculation time. However the evaluation of basis functions was found to be a relatively short part of the total calculation time of each executable process, and so this strategy was not explored further on the project.

### Iterative solutions inside component calculation

No iterative solutions for the inner sparse symmetric system were explored. These do offer potential as they are guaranteed to converge, and it would be straightforward to propose a reasonable initialisation, based for example on taking a weighted mean over some subset of observations.

# Bibliography

|  |  |
| --- | --- |
| [1] | J. R. Mitchelson, “EUSTACE System Technical Requirements,” Met Office Hadley Centre, Exeter, UK, 2016. |
| [2] | European Commission, *Grant agreement number: 640171 — EUSTACE — H2020-EO-2014, Annex 1 - Description of Action,* 2014. |
| [3] | J. R. Mitchelson, J. Kennedy, C. Morice, F. Lindgren, A. Stephens and A. Waterfall, “EUSTACE initial recommendations on data formats and management,” [Online]. Available: http://proj.badc.rl.ac.uk/eustace/browser/platform/trunk/system/data/docs/WP2\_MS12\_recommendations\_data\_formats.docx. [Accessed 26 01 2016]. |
| [4] | E. International, “The JSON Data Interchange Format,” [Online]. Available: http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf. [Accessed 26 01 2016]. |
| [5] | Georg Brandl and the Sphinx team, “SPHINX Python Documentation Generator,” [Online]. Available: http://www.sphinx-doc.org/en/stable/ext/autodoc.html. [Accessed 22 09 2016]. |
| [6] | C. Morice, “Overview of Methods for the EUSTACE WP2 Advanced Standard Analysis,” 2016. |