ERT Documentation

Release 2.3

Joakim Hove

Nov 12, 2018

CONTENTS:

CHAPTER

ONE

INTRODUCTION TO ERT AND ENSEMBLE BASED METHODS

The reservoir model for a green field is based on a range of subsurface input including seismic data, a geological concept, well logs and fluid samples. All of this data is uncertain, and it is quite obvious that the resulting reservoir model is quite uncertain. Although uncertain - reservoir models are still the only tool we have when we make reservoir management decisions for the future.

Since reservoir models are very important for future predictions there is much focus on reducing the uncertainty in the models. When the field has been in production for some time one can use true data assembled from the producing field to update the model. This process is commonly called *history matching* in the petroleum industry, in this manual we will use the term *model updating*. Before the model updating process can start you will need:

1. A reservoir model which has been *parameterized* with a parameter set $\{\lambda\}$.
2. Observation data from the producing field d.

Then the the actual model updating goes like this:

1. Simulate the behaviour of the field and assemble simulated data s.
2. Compare the simulated simulated data s with the observed data d.
3. Based on the misfit between s and d updated parameters $\{\lambda'\}$ are calculated.

Model updating falls into the general category of *inverse problems* - i.e. we know the results and want to determine the input parameters which reproduce these results. In statistical literature the the process is often called *conditioning*.

It is very important to remember that the sole reason for doing model updating is to be able to make better predictions for the future, the history has happened already anyway!

1.1 Embrace the uncertainty

The main purpose of the model updating process is to reduce the uncertainty in the description of the reservoir, however it is important to remember that the goal is *not* to get rid of all the uncertainty and find one true answer. There are two reasons for this:

1. The data used when conditioning the model is also uncertain. E.g. measurements of e.g. water cut and GOR is limited by the precision in the measurement apparatus and also the allocation procedures. For example for 4D seismic the uncertainty is large.
2. The model updating process will take place in the abstract space spanned by the parameters $\{\lambda\}$ - unless you are working on a synthetic example the *real reservoir* is certainly not in this space.

So the goal is to update the parameters $\{\lambda\}$ so that the simulations agree with the observations on average, with a variability which is of the same order of magnitude as the uncertainty in the observations. The assumption is then that if this model is used for predictions it will be unbiased and give a realistic estimate of the future uncertainty. This illustrated in figure `ensemble`.

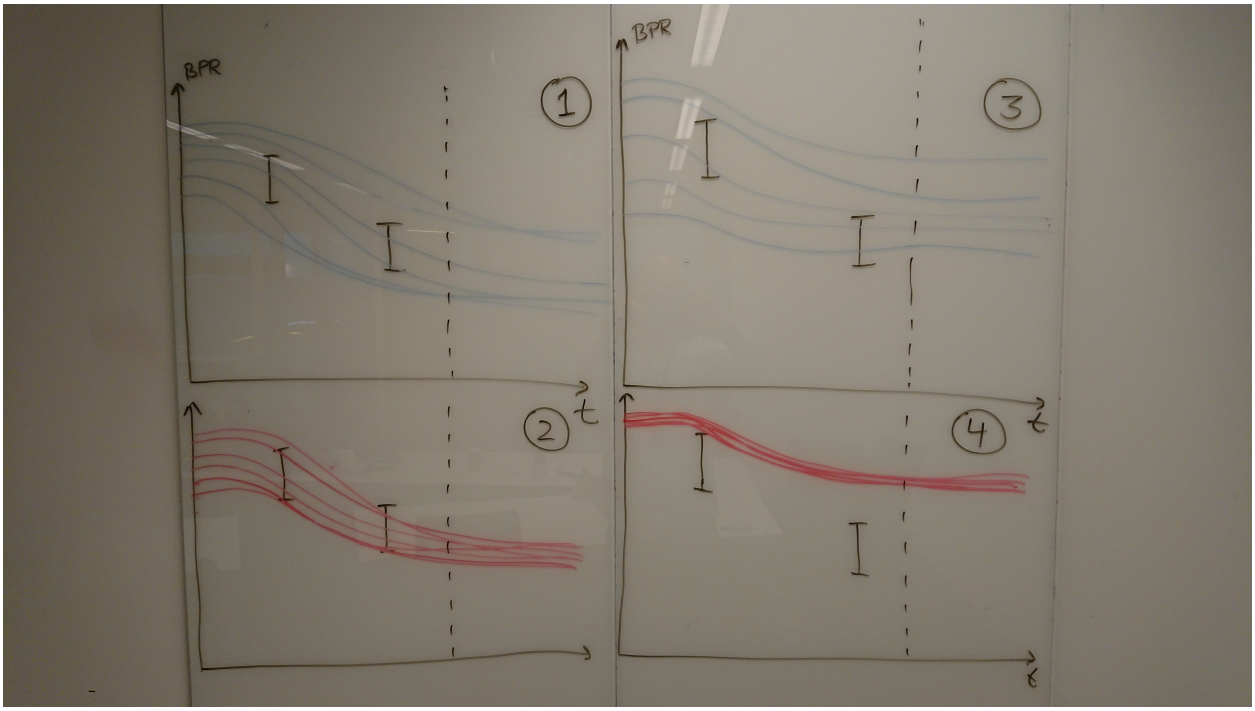

Fig. 1: Ensemble plots before and after model updating, for one succesfull updating and one updating which has gone wrong.

All the plots show simulations pressure in a cell as a function of time, with measurements. Plots (1) and (3) show simulations before the model updating (i.e. the *prior*) and plots (2) and (4) show the plots after the update process (the *posterior*). The dashed vertical line is meant to illustrate the change from history to prediction.

The left case with plots (1) and (2) is a succesfull history matching project. The simulations from the posterior distribution are centered around the observed values and the spread - i.e. uncertainty - is of the same order of magnitude as the observation uncertainty. From this case we can reasonably expect that predictions will be unbiased with an reasonable estimate of the uncertainty.

For the right hand case shown in plots (3) and (4) the model updating has *not* been successfull and more work is required. Looking at the posterior solution we see that the simulations are not centered around the observed values, when the observed values from the historical period are not correctly reproduced there is no reason to assume that the predictions will be correct either. Furthermore we see that the uncertainty in the posterior case (4) is much smaller than the uncertainty in the observations - this does not make sense; although our goal is to reduce the uncertainty it should not be reduced significantly beyond the uncertainty in the observations. The predictions from (4) will most probably be biased and greatly underestimate the future uncertainty[1].

1.2 Ensemble Kalman Filter - EnKF

The ERT application was originally devised created to do model updating of reservoir models with the EnKF algorithm. The experience from real world models was that EnKF was not very suitable for reservoir applications, and ERT has since changed to the Ensemble Smoother (ES) which can be said to be a simplified version of the EnKF. But the characteristics of the EnKF algorithm still influence many of the design decisions in ERT, it therefor makes sense to give a short introduction to the Kalman Filter and EnKF.

[1] : It should be emphasized that plots (3) and (4) show one simulated quantity from an assumed larger set of observations, in general there has been a different set of observations which has induced these large and unwanted updates.

1.2.1 The Kalman Filter

The Kalman FIlter originates in electronics the 60's. The Kalman filter is *widely* used, especially in applications where positioning is the goal - e.g. the GPS system. The typical ingredients where the Kalman filter can be interesting to try include:

1. We want to determine the final *state* of the system - this can typically the the position.
2. The starting position is uncertain.
3. There is an *equation of motion* - or *forward model* - which describes how the system evolves in time.
4. At fixed point in time we can *observe* the system, these observations are uncertain.

As a very simple application of the Kalman Filter, assume that we wish to estimate the position of a boat as $x(t)$; we know where the boat starts (initial condition), we have an equation for how the boat moves with time and at selected points in time t_k we get *measurements* of the position. The quantities of interest are:

x_k The estimated position at time t_k.

σ_k The uncertainty in the position at time t_k.

x_k^* The *estimated/forecasted* position at time t_k - this is the position estimated from x_{k-1} and $f(x,t)$, but before the observed data d_k is taken into account.

d_k **The observed values which are used in the updating process, the** d_k values are measured with a process external to the model updating.

σ_d **The uncertainty in the measurement** d_k **- a reliable** estimate of this uncertainty is essential for the algorithm to place "correct" weight on the measured values.

$f(x,t)$

The equation of motion - ***forward model*** **- which propagates** $x_{k-1} \rightarrow x_k^*$

The purpose of the Kalman Filter is to determine an updated x_k from x_{k-1} and d_k. The updated x_k is the value which *minimizes the variance* σ_k. The equations for updated position and uncertainty are:

$$x_k = x_k^* \frac{\sigma_d^2}{\sigma_k^2 + \sigma_d^2} + x_d \frac{\sigma_k^2}{\sigma_k^2 + \sigma_d^2}$$

$$\sigma_k^2 = \sigma_k^i 2* \left(1 - \frac{\sigma_k^{2*}}{\sigma_d^2 + \sigma_k^{2*}}\right)$$

Looking at the equation for the position update we see that the analyzed position x_k is a weighted sum over the forecasted positon x_k^* and measured position d_k - where the weighting depends on the relative weight of the uncertainties σ_k^* and σ_d. For the updated uncertainty the key take away message is that the updated uncertainty will always be smaller than the forecasted uncertainty: $\sigma_k < \sigma_k^*$.

1.2.2 Using an ensemble to estimate the uncertainty: EnKF

1.3 Ensemble Smoother - ES

CHAPTER

TWO

THE DATA TYPES AVAILABLE IN ERT

Very briefly described the purpose of ERT is to pass uncertain paramater values to a simulator[1], in a form which works as suitable input to the simulator and then subsequently load the results from the simulator. This means that data must be formatted in a form which the simulator can load, and also that ERT must be able to read files generated by the simulator.

The data managed by ERT are organized in different *data types* described in this chapter. Configuring the data used in the conditioning project is a very important part of setting up a ERT configuration file - in practical terms this is how you configure which uncertainty parameters should be studied. The data types in ert can be categorized in two ways:

1. How the data type behaves dynamically: is it a static parameter like porosity or permeability - i.e. does it serve as *input* to the simulator, or is it a quantity which is generated as a result from the simulation. When understanding the model updating algorithm and process it is important to understand this difference properly.
2. How the data type is implemented, what type of files does it read and write, how is it configured and so on.

NB: All datatypes have a common namespace; i.e. a globally unique keyword.

2.1 Parameters

The *parameter* datatypes will serve as input to the simulator. For a pure uncertainty study they will pass right through, model updating update parameters. Sample internally or load externally.

2.1.1 Scalar parameters with a template: GEN_KW

The GEN_KW datatype is used in situations where you have a handful of related[2] parameters. The intention behind the GEN_KW implementation is that ERT will *sample* parameters randomly according to a distribution specified by the user, then ERT will use the sampled values and a template file provided by the user to generate a file which can be used by the simulator.

In the main config file a GEN_KW instance is defined as follows:

```
GEN_KW  ID  templates/template.txt  include.txt  priors.txt
```

Here ID is an (arbitrary) unique string, `templates/template.txt` is the name of a template file, `include.txt` is the name of the file which is made for each realization based on the template file `templates/template.txt` and the prior distribution `priors.txt` is a file containing a list of parametrized keywords and a prior distribution for each. Note that you must manually edit the input files of the simulator to ensure that the generated file `include.txt` is included.

[1] *Simulator* should in this context be understood as the complete forward model, including various pre and post processing steps in addition to the actual reservoir simulation.

[2] ERT itself can not impose correlations among the parameters, if you need that you must implement it yourself in the forward model.

Example: configuring GEN_KW to estimate MULTPV values

Let us consider an example where the GEN_KW parameter type is used to estimate pore volume multipliers in an Eclipse model. We could then declare a GEN_KW instance in the main enkf configuration file:

```
GEN_KW PAR_MULTPV multpv_template.txt multpv.txt multpv_priors.txt
```

In the GRID or EDIT section of the ECLIPSE data file, we would insert the ollowing include statement:

```
INCLUDE
  'multpv.txt' /
```

The template file multpv_template.txt would contain some parametrized ECLIPSE statements:

```
BOX
       1 10 1 30 13 13 /

MULTPV
 300*<MULTPV_BOX1> /

ENDBOX

BOX
       1 10 1 30 14 14 /

MULTPV
       300*<MULTPV_BOX2> /

ENDBOX
```

Here, <MULTPV_BOX1> and <MULTPV_BOX2> will act as magic strings. Note that the '<' '>' must be present around the magic strings. In this case, the parameter configuration file multpv_priors.txt could look like this:

```
MULTPV_BOX2 UNIFORM 0.98 1.03
MULTPV_BOX1 UNIFORM 0.85 1.00
```

In general, the first keyword on each line in the parameter configuration file defines a key, which when found in the template file enclosed in '<' and '>', is replaced with a value. The rest of the line defines a prior distribution for the key. See Prior distributions available in enkf for a list of available prior distributions. The various prior distributions available for the `GEN_KW` keyword are here prior distributions available in ERT

The prior - it is really a *transformation*

The Ensemble Smoother method, which ERT uses for updating of parameters, works with normally distributed variables. So internally in ERT the interplay between `GEN_KW` variables and updates is as follows:

1. ERT samples a random variable `x ~ N(0,1)` - before outputing to the forward model this is *transformed* to `y ~ F(Y)` where the the distribution `F(Y)` is the correct prior distribution.
2. When the prior simulations are complete ERT calculates misfits between simulated and observed values and *updates* the parameters; hence the variables `x` now represent samples from a posterior distribution which is Normal with mean and standard deviation *different from (0,1)*.

The transformation prescribed by `F(y)` still "works" - but it no longer maps to a distribution in the same family as initially specified by the prior. A consequence of this is that the update process can *not* give you a posterior with updated parameters in the same distribution family as the Prior.

NORMAL

To set a normal (Gaussian) prior, use the keyword NORMAL. It takes two arguments, a mean value and a standard deviation. Thus, the following example will assign a normal prior with mean 0 and standard deviation 1 to the variable VAR1:

```
VAR1    NORMAL     0    1
```

LOGNORMAL

A stochastic variable is log normally distributed if the logarithm of the variable is normally distributed. In other words, if X is normally distributed, then Y = exp(X) is log normally distributed.

A log normal prior is suited to model positive quanties with a heavy tail (tendency to take large values). To set a log normal prior, use the keyword LOGNORMAL. It takes two arguments, the mean and standard deviation of the *logarithm* of the variable:

```
VAR2    LOGNORMAL   0   1
```

TRUNCATED_NORMAL

This *TRUNCATED_NORMAL* distribution works as follows:

1. Draw random variable $X \sim N(\mu, \sigma)$
2. Clamp X to the interval [min, max]

This is **not** a proper truncated normal distribution; hence the clamping to ‘‘[min,max]‘ should be an exceptional event. To configure this distribution for a situation with mean 1, standard deviation 0.25 and hard limits 0 and 10:

```
VAR3   TRUNCATED_NORMAL   1   0.25    0   10
```

UNIFORM

A stochastic variable is uniformly distributed if has a constant probability density on a closed interval. Thus, the uniform distribution is completely characterized by it’s minimum and maximum value. To assign a uniform distribution to a variable, use the keyword UNIFORM, which takes a minimum and a maximum value for a the variable. Here is an example, which assigns a uniform distribution between 0 and 1 to a variable `VAR4`:

```
VAR4 UNIFORM 0 1
```

It can be shown that among all distributions bounded below by a and above by b, the uniform distribution with parameters a and b has the maximal entropy (contains the least information). Thus, the uniform distribution should be your preferred prior distribution for robust modeling of bounded variables.

LOGUNIF

A stochastic variable is log uniformly distributed if it’s logarithm is uniformly distributed on the interval [a,b]. To assign a log uniform distribution to a a variable, use the keyword LOGUNIF, which takes a minimum and a maximum value for the output variable as arguments. The example

```
VAR5  LOGUNIF 0.00001 1
```

will give values in the range [0.00001,1] - with considerably more weight towards the lower limit. The log uniform distribution is useful when modeling a bounded positive variable who has most of it's probability weight towards one of the bounds.

CONST

The keyword CONST is used to assign a Dirac distribution to a variable, i.e. set it to a constant value. Here is an example of use:

```
VAR6 CONST 1.0
```

DUNIF

The keyword DUNIF is used to assign a discrete uniform distribution. It takes three arguments, the number bins, a minimum and maximum value. Here is an example which creates a discrete uniform distribution on [0,1] with 25 bins:

```
VAR7 DUNIF 25 0 1
```

ERRF

The ERRF keyword is used to define a prior resulting from applying the error function to a normally distributed variable with mean 0 and variance 1. The keyword takes four arguments:

```
VAR8 ERRF MIN MAX SKEWNESS WIDTH
```

The arguments MIN and MAX sets the minimum and maximum value of the transform. Zero SKEWNESS results in a symmetric distribution, whereas negative SKEWNESS will shift the distribution towards the left and positive SKEWNESS will shift it towards the right. Letting WIDTH be larger than one will cause the distribution to be unimodal, whereas WIDTH less than one will create a bi-modal distribution.

DERRF

The keyword DERRF is similar to ERRF, but will create a discrete output. DERRF takes 5 arguments:

```
VAR9 DERRF NBINS MIN MAX SKEWNESS WIDTH
```

NBINS set the number of discrete values, and the other arguments have the same effect as in ERRF.

Loading GEN_KW values from an external file

The default use of the GEN_KW keyword is to let the ERT application sample random values for the elements in the GEN_KW instance, but it is also possible to tell ERT to load a precreated set of data files, this can for instance be used as a component in a experimental design based workflow. When using external files to initialize the GEN_KW instances you supply an extra keyword `INIT_FILE:/path/to/priors/files%d` which tells where the prior files are:

```
GEN_KW  MY-FAULTS   MULTFLT.tmpl   MULTFLT.INC   MULTFLT.txt    INIT_FILES:priors/
↪multflt/faults%d
```

In the example above you must prepare files priors/multflt/faults0, priors/multflt/faults1, ... priors/multflt/faultsn which ert will load when you initialize the case. The format of the GEN_KW input files can be of two varieties:

1. The files can be plain ASCII text files with a list of numbers:

```
1.25
2.67
```

The numbers will be assigned to parameters in the order found in the MULTFLT.txt file.

2. Alternatively values and keywords can be interleaved as in:

```
FAULT1 1.25
FAULT2 2.56
```

in this case the ordering can differ in the init files and the parameter file.

The heritage of the ERT program is based on the EnKF algorithm, and the EnKF algorithm evolves around Gaussian variables - internally the GEN_KW variables are assumed to be samples from the N(0,1) distribution, and the distributions specified in the parameters file are based on transformations starting with a N(0,1) distributed variable. The slightly awkward consequence of this is that to let your sampled values pass through ERT unmodified you must configure the distribution NORMAL 0 1 in the parameter file; alternatively if you do not intend to update the GEN_KW variable you can use the distribution RAW.

2.1.2 3D field parameters: FIELD

The FIELD data type is used to parametrize quantities which have extent over the full grid; porosity and permeability are the most typical examples of quantities which are estimated and modelled with the FIELD data type. In the configuration file the FIELD keywords are configured like this:

```
FIELD  PORO PARAMETER  poro.grdecl  .....
```

PORO is in principle an arbitrary string ID, but if the fields in question represent e.g. the porosity use of a matching string of course makes sense. The string "PARAMETER" serves no purpose at the moment, but is legacy from the time when ERT could do full EnKF and also needed to handle dynamic fields like pressure and saturations.

The "poro.grdecl" argument represents the name of the file which ert will prepare for the forward model, observe the reservoir data file must have an *INCLUDE* statement corresponding to this file, i.e.

```
INCLUDE
    'poro.grdecl' /
```

For the example above.

Field initialization

Observe that ERT can *not* sample field variables internally, they must be supplied through another application - typically geo modelling software like RMS; so to use the FIELD datatype you must have a workflow external to ERT which can create/sample the fields. When you have established a workflow for generating these fields externally there are *two* ways to load them into ERT: *INIT_FILES* to load pregenerated initial fields or *FORWARD_INIT* to load as part of the forward model.

Initialization with INIT_FILES

In the situation where you do not have geo modelling as a part of the forward model you will typically use the geo modelling software to create an ensemble of geological realisations up front. Assuming you intend to update the porosity these realisations should typically be in the form of files `/path/poro_0.grdecl, /path/poro_1.grdecl, ... /path/poro_99.grdecl`. The `INIT_FILES:` directive is used to configure ERT to load those files when ERT is initializing the data. The number `0, 1, 2, ...` should be replaced with the integer format specified `%d` - which ERT will replace with the realization number runtime, i.e.

```
FIELD ... INIT_FILES:/path/poro_%d.grdecl
```

in this case. The files can be in eclipse grdecl format or rms roff format; the type is determined from the extension so you should use the common extensions `grdecl` or `roff`.

Initialization with FORWARD_INIT

When geomodelling is an integrated part of the forward model it is more attractive to let the forward model generate the parameter fields. To enable this we must pass the `FORWARD_INIT:True` when configuring the field, and also pass a name in the `INIT_FILES:poro.grdecl` for the file which should be generated by the forward model component.

Observe that there are two important differences to the `INIT_FILES:` attribute when it used as *the way* to initialize fields, and when it is used in combination with `FORWARD_INIT:True`. When `INIT_FILES:` is used alone the filename given should contain a `%d` which will be replaced with realization number, when used with `FORWARD_INIT:True` that is not necessary. Furthermore in the `FORWARD_INIT:True` case the *the path is interpreted relative to the runpath folder*, whereas in the other case the path is interpreted relative to the location of the main ERT configuration file.

When using `FORWARD_INIT:True` together with an update algorithm in ERT the field generated by the geo modelling software should only be used in the first iteration (prior), in the subsequent iterations the forward model should use the field as it comes out from ERT. The typical way to achieve this is:

1. The forward model component outputs to a temporary file `tmp_poro.grdecl`.
2. In the first iteration ERT will *not* output a file `poro.grdecl`, but in the second and subsequent iterations a `poro.grdecl` file will be created by ERT - this is at the core of the `FORWARD_INIT:True` functionality.
3. In the forward model there should be a job `CAREFULL_COPY` which will copy `tmp_poro.grdecl` *only if* `poro.grdecl` does not already exist. The rest of the forward model components should use `poro.grdecl`.

Field transformations

For Assisted history matching, the variables in ERT should be normally distributed internally - the purpose of the transformations is to enable working with normally distributed variables internally in ERT and expose another distribution to the forward model through the use of transformations. Thus, the optional arguments `INIT_TRANSFORM:FUNC` and `OUTPUT_TRANSFORM:FUNC` are used to transform the user input of parameter distribution. `INIT_TRANSFORM:FUNC` is a function which will be applied when the field are loaded into ERT. `OUTPUT_TRANSFORM:FUNC` is a function which will be applied to the field when it is exported from ERT, and `FUNC` is the name of a transformation function to be applied. The avaialble functions are listed below:

POW10 This function will raise x to the power of 10: $y = 10^x$.

TRUNC_POW10 This function will raise x to the power of 10 - and truncate lower values at 0.001.

LOG This function will take the NATURAL logarithm of x: $y = ln(x)$.

LN This function will take the NATURAL logarithm of x: $y = ln(x)$.

LOG10 This function will take the log10 logarithm of x: $y = log10(x)$.

EXP This function will calculate $y = exp(x)$.

LN0 This function will calculate $y = ln(x + 0.000001$

EXP0 This function will calculate $y = exp(x) - 0.000001$

The most common scenario is that a log-normal distributed permeability in the geo modelling software is transformed to become normally distributted in ERT, to achieve this you do:

1. `INIT_TRANSFORM:LOG` To ensure that the variables which were initially log-normal distributed are transformed to normal distribution when they are loaded into ert.
2. `OUTPUT_TRANSFORM:EXP` To ensure that the variables are reexponentiated to be log-normal distributed before going out to Eclipse.

2.1.3 2D Surface parameters: SURFACE

The SURFACE keyword can be used to work with surface from RMS in the irap format. The surface keyword is configured like this:

```
SURFACE TOP   OUTPUT_FILE:surf.irap   INIT_FILES:Surfaces/surf%d.irap   BASE_
↪SURFACE:Surfaces/surf0.irap
```

The first argument, TOP in the example above, is the identifier you want to use for this surface in ert. The `OUTPUT_FILE` key is the name of surface file which ERT will generate for you, `INIT_FILES` points to a list of files which are used to initialize, and `BASE_SURFACE` must point to one existing surface file. When loading the surfaces ERT will check that all the headers are compatible. An example of a surface IRAP file is:

```
-996   511     50.000000     50.000000
444229.9688   457179.9688  6809537.0000  6835037.0000
260      -30.0000   444229.9688  6809537.0000
0     0     0     0     0     0     0
2735.7461    2734.8909    2736.9705    2737.4048    2736.2539    2737.0122
2740.2644    2738.4014    2735.3770    2735.7327    2733.4944    2731.6448
2731.5454    2731.4810    2730.4644    2730.5591    2729.8997    2726.2217
2721.0996    2716.5913    2711.4338    2707.7791    2705.4504    2701.9187
....
```

The surface data will typically be fed into other programs like Cohiba or RMS. The surface data can be updated using the Smoother.

Initializing from the FORWARD MODEL

All the parameter types like FIELD,GEN_KW,GEN_PARAM and SURFACE can be initialized from the forward model. To achieve this you just add the setting FORWARD_INIT:True to the configuration. When using forward init the initialization will work like this:

1. The explicit initialization from the case menu, or when you start a simulation, will be ignored.
2. When the FORWARD_MODEL is complete ERT will try to initialize the node based on files created by the forward model. If the init fails the job as a whole will fail.
3. If a node has been initialized, it will not be initialized again if you run again. [Should be possible to force this ....]

When using FORWARD_INIT:True ERT will consider the INIT_FILES setting to find which file to initialize from. If the INIT_FILES setting contains a relative filename, it will be interpreted relativt to the runpath directory. In the

example below we assume that RMS has created a file petro.grdecl which contains both the PERMX and the PORO fields in grdecl format; we wish to initialize PERMX and PORO nodes from these files:

```
FIELD   PORO  PARAMETER    poro.grdecl     INIT_FILES:petro.grdecl  FORWARD_INIT:True
FIELD   PERMX PARAMETER    permx.grdecl    INIT_FILES:petro.grdecl  FORWARD_INIT:True
```

Observe that forward model has created the file petro.grdecl and the nodes PORO and PERMX create the ECLIPSE input files poro.grdecl and permx.grdecl, to ensure that ECLIPSE finds the input files poro.grdecl and permx.grdecl the forward model should contain a job which will copy/convert petro.grdecl -> (poro.grdecl,permx.grdecl), this job should not overwrite existing versions of permx.grdecl and poro.grdecl. This extra hoops is not strictly needed in all cases, but strongly recommended to ensure that you have control over which data is used, and that everything is consistent in the case where the forward model is run again.

2.1.4 General vector parameters: GEN_PARAM

The GEN_PARAM parameter type is used to estimate parameters which do not really fit into any of the other categories. As an example, consider the following situation:

Some external Software (e.g. Cohiba) makes a large vector of random numbers which will serve as input to the forward model. (It is no requirement that the parameter set is large, but if it only consists of a few parameters the GEN_KW type will be easier to use.) We want to update this parameter with enkf. In the main configuration file the input for a GEN_PARAM instance is as follows:

```
GEN_PARAM  ID  ECLIPSE_FILE  INPUT_FORMAT:xx  OUTPUT_FORMAT:xx  INIT_FILES:/path/to/
↪init/files%d (TEMPLATE:/template_file KEY:magic_string)
```

here ID is the usual unique string identifying this instance and ECLIPSE_FILE is the name of the file which is written into the run directories. The three arguments GEN_PARAM, ID and ECLIPSE_FILE must be the three first arguments. In addition you must have three additional arguments, INPUT_FORMAT, OUTPUT_FORMAT and INIT_FILES. INPUT_FORMAT is the format of the files enkf should load to initialize, and OUTPUT_FORMAT is the format of the files enkf writes for the forward model. The valid values are:

- ASCII - This is just text file with formatted numbers.
- ASCII_TEMPLATE - An plain text file with formatted numbers, and an arbitrary header/footer.
- BINARY_FLOAT - A vector of binary float numbers.
- BINARY_DOUBLE - A vector of binary double numbers.

Regarding the different formats - observe the following:

1. Except the format ASCII_TEMPLATE the files contain no header information.
2. The format ASCII_TEMPLATE can only be used as output format.
3. If you use the output format ASCII_TEMPLATE you must also supply a TEMPLATE:X and KEY:Y option. See documentation of this below.
4. For the binary formats files generated by Fortran can not be used - can easily be supported on request.

Regarding templates:

If you use OUTPUT_FORMAT:ASCII_TEMPLATE you must also supply the arguments TEMPLATE:/template/file and KEY:MaGiCKEY. The template file is an arbitrary existing text file, and KEY is a magic string found in this file. When enkf is running the magic string is replaced with parameter data when the ECLIPSE_FILE is written to the directory where the simulation is run from. Consider for example the follwing configuration:

```
TEMPLATE:/some/file   KEY:Magic123
```

The template file can look like this (only the Magic123 is special):

```
      Header line1
      Header line2
============
      Magic123
      ============
      Footer line1
      Footer line2
```

When ERT is running the string Magic123 is replaced with parameter values, and the resulting file will look like this:

```
Header line1
Header line2
============
1.6723
5.9731
4.8881
.....
============
Footer line1
Footer line2
```

2.2 Simulated data

The datatypes in the *Simulated data* chapter correspond to datatypes which are used to load results from a forward model simulation and into ERT. In a model updating workflow instances of these datatypes are compared with observed values and that is used as basis for the update process. Also post processing tasks like plotting and QC is typically based on these data types.

2.2.1 Summary: SUMMARY

The `SUMMARY` keyword is used to configure which summary vectors you want to load from the (Eclipse) reservoir simulation. In it's simplest form the `SUMMARY` keyword just lists the vectors you wish to load, you can have multiple `SUMMARY` keywords in your config file, and each keyword can mention multiple vectors:

```
SUMMARY  WWCT:OP_1  WWCT:OP_2  WWCT:OP_3
SUMMARY  FOPT FOPR  FWPR
SUMAMRY  GGPR:NORTH GOPR:SOUTH
```

If you in the observation use the `SUMMARY_OBSERVATION` or `HISTORY_OBSERVATION` keyword to compare simulatons and observations for a particular summary vector *that vector is automatically added* to the ERT configuration.

If you use the keyword `REFCASE` to configure an Eclipse reference case you can use wildcard notation to all summary vectors matching a pattern, i.e. this:

```
REFCASE eclipse/refcase/CASE
SUMMARY WWCT:* WGOR:*
SUMMARY F*
SUMMARY G*:NORTH
```

will load the `WWCT` and `WGOR` vectors for all wells, all field related vectors and all group vectors from the `NORTH` group.

2.2.2 General data: GEN_DATA

The GEN_DATA keyword is used to to load *arbitrary* which has been generated by the forward model. ERT does not have any awareness of the type of data encoded in a GEN_DATA keyword; it could be the result of gravimetric calculation or the pressure difference across a barrier in the reeservoir. This means that the GEN_DATA keyword is extremely flexible, but also slightly complicated to configure. Assume a GEN_DATA keyword is used to represent the result of an estimated of the position of the oil water contact which should be compared with a oil water contact from 4D seismic; this could be achieved with the configuration:

```
GEN_DATA 4DWOC  RESULT_FILE:SimulatedWOC_%d.txt  INPUT_FORMAT:ASCII   REPORT_STEPS:0
```

The 4DWOC is an arbitrary unique key, RESULT_FILE:SimulatedWOC%d.txt means that ERT will look for results in the file SimulatedWOC_0.txt. The INPUT_FORMAT:ASCII means that ERT will expect the result file to be formatted as an ASCII file, the other alternative formats are INPUT_FORMAT:BINARY_FLOAT and INPUT_FORMAT:BINARY_DOUBLE - in general only the ASCII alternative is used, and in the future that should at least be the default.

The REPORT_STEPS:0 is tightly bound to the %d integer format specifier in the result file - at load time the %d is replaced with the integer values given in the REPORT_STEPS: option, for the example given above that means that %d will be replaced with 0 and ERT will look for the file SimulatedWOC_0.txt. In principle it is possible to configure several report steps like: REPORT_STEPS:0,10,20 - then ERT will look for all three files SimulatedWOC_0.txt, SimultedWOC_10.txt and SimulatedWOC_20.txt. It is quite challenging to get this right, and the recommendation is to just stick with *one* result file at report step 0[3], in the future the possibility to load one keyword GEN_DATA for multiple report steps will probably be removed, but for now the GEN_DATA configuration is *quite strict* - it will fail if the RESULT_FILE attribute does not contain a %d.

Observe that since the actual result file should be generated by the forward model, it is not possible for ERT to fully validate the GEN_DATA keyword at configure time. If for instance your forward model generates a file SimulatedWOC_0 (without the .txt extension you have configured), the configuration problem will not be detected before ERT eventuallly fails to load the file SimulatedWOC_0.txt.

2.2.3 Keyword results: CUSTOM_KW

The CUSTOM_KW datatype is in a way the dynamic analogue to the GEN_KW datatype. It is intended to load a list of key,value pairs from a file generated by the forward model. The CUSTOM_KW datatype is configured like this:

```
CUSTOM_KW KEYWORD RESULT_FILE:/name/of/file
```

Observe that even though it is a dynamic result type the CUSTOM_KW data type can not be used as the simulated value when comparing with an observation. An example of CUSTOM_KW file generated by the forward model could look like this:

```
KEY1 10
KEY2 246
KEY3 02
```

I.e. interleaved string keys and corresponding numeric values.

[3] The option is called *report step* - but the time aspect is not really important. You could just as well see it as an arbitrary label, the only important thing is that *if* you have a corresponding GEN_OBS observation of this GEN_DATA vector you must match the report step used when configuring the GEN_DATA and the GEN_OBS.

2.3 EnKF heritage

With regards to the datatypes in ERT this is a part of the application where the EnKF heritage shows through quite clearly, the datetypes offered by ERT would probably be different if ERT was made for Ensemble Smoother from the outset. Pecularites of EnKF heritage include:

1. The *FIELD* implementation can behave both as a dynamic quantity, i.e. pressure and saturation, and static property like porosity. In ERT it is currently *only used* as a parameter, but that this *possible* dual usage exists in the code adds a significant complexity.
2. The parameter types have an internal pseudo time dependence corresponding to the "update time" induced by the EnKF scheme. This pseudo time dependence is not directly exposed to the user, but it is still part of the implementation and e.g. when writing plugins which work with parameter data managed by ERT you must relate to it.
3. The time dependence of the *GEN_DATA* implementation. This is just too complex, there have been numerous problems with people who configure the *GEN_DATA* keywords incorrectly.

CHAPTER

THREE

RUNNING SIMULATIONS - THE FORWARD MODEL

A *very* important part of ERT functionality.

3.1 The forward model

3.2 The runpath directory

3.2.1 Default jobs

Reservoir simulation: eclipse

Reservoir modelling: RMS

File system utilities

3.2.2 Configuring your own jobs

3.2.3 The *job_dispatch* executable

3.3 Interfacing with the cluster

CHAPTER

FOUR

WORKFLOWS

CHAPTER

FIVE

CONFIGURING OBSERVATIONS FOR ERT

5.1 General overview

When using ERT to condition on dynamic data, it is necessary to specify which data to condition on. In particular, for a given piece of data to condition on, the ERT application needs to know:

- The actual measured value of the data.
- The uncertainty of the measured data.
- The time of measurement.
- How to simulate a response of the data given a parametrized ECLIPSE model.

To provide this observation to ERT, an observation file must be created. The observation file is a plain text file, and is in essence built around for different classes of observations and has an associated keyword for each class:

- Well or group rates from an existing ECLIPSE reference case: The HISTORY_OBSERVATION keyword.
- Well logs, RFTS and PLTs: The BLOCK_OBSERVATION keyword.
- Separator tests, region pressures, etc.: The SUMMARY_OBSERVATION keyword.
- Exotic observations (e.g. data from 4D seismic): The GENERAL_OBSERVATION keyword.

Observe that observations and datatypes are quite tightly linked together. Before reading this you should have a firm grasp of the dynamic data types.

5.2 The HISTORY_OBSERVATION keyword

The keyword HISTORY_OBSERVATION is used to condition on observations from the WCONHIST and WCONINJH keywords in schedule file provided to the enkf project (or alternatively an ECLIPSE summary file if you have changed the HISTORY_SOURCE keyword in the enkf project). The keyword is typically used to condition on production and injection rates for groups and wells, as well as bottom hole and tubing head pressures. An observation entered with the HISTORY_OBSERVATION keyword will be active at all report steps where data for the observation can be found.

In it's simplest form, a history observation is created as follows:

```
HISTORY_OBSERVATION WOPR:P1;
```

This will condition on WOPR in well P1 using a default observation error. The default observation error is a relative error of 10% to the measurement with a minimum error of 0.10. See below on how explicitly set the error.

In general, to condition on variable VAR in well or group WGNAME, one uses:

```
HISTORY_OBSERVATION VAR:WGNAME;
```

Note that there must be a colon ":" between VAR and WGNAME and that the statement shall end with a semi-colon ";". Thus, to condition on WOPR, WWCT and WGOR in well C-17, and for the GOPR for the whole field, one would add the following to the observation configuration:

```
HISTORY_OBSERVATION WOPR:C-17;
HISTORY_OBSERVATION WWCT:C-17;
HISTORY_OBSERVATION WGOR:C-17;

HISTORY_OBSERVATION GOPR:FIELD;
```

By default, the observation error is set to 10% of the observed value, with a minimum of 0.10. It can be changed as follows:

```
HISTORY_OBSERVATION GOPR:FIELD
{
   ERROR       = 1000;
   ERROR_MODE  = ABS;
};
```

This will set the observation error to 1000 for all observations of GOPR:FIELD. Note that both the items ERROR and ERROR_MODE as well as the whole definition shall end with a semi-colon.

The item ERROR_MODE can take three different values: ABS, REL or RELMIN. If set to REL, all observation errors will be set to the observed values multiplied by ERROR. Thus, the following will condition on water injection rate for the whole field with 20% observation uncertainity:

```
HISTORY_OBSERVATION GWIR:FIELD
{
   ERROR       = 0.20;
   ERROR_MODE  = REL;
};
```

If you do not want the observation error to drop below a given threshold, say 100, you can use RELMIN and the keyword ERROR_MIN:

```
HISTORY_OBSERVATION GWIR:FIELD
{
   ERROR       = 0.20;
   ERROR_MODE  = RELMIN;
   ERROR_MIN   = 100;
};
```

Note that the configuration parser does not threat carriage return different from space. Thus, the following statement is equivalent to the previous:

```
HISTORY_OBSERVATION GWIR:FIELD { ERROR = 0.20; ERROR_MODE = RELMIN; ERROR_MIN = 100; }
↪;
```

Also note that the special keyword include can be used to read an external file. This can be very useful if you want to change the standard configuration for a lot of observations in one go. For example, consider the following code:

```
HISTORY_OBSERVATION WOPR:P1 { include "hist_obs_wells.txt"; };
HISTORY_OBSERVATION WOPR:P2 { include "hist_obs_wells.txt"; };
HISTORY_OBSERVATION WOPR:P3 { include "hist_obs_wells.txt"; };
```

(continues on next page)

(continued from previous page)

```
HISTORY_OBSERVATION WOPR:P4 { include "hist_obs_wells.txt"; };
HISTORY_OBSERVATION WOPR:P5 { include "hist_obs_wells.txt"; };
```

Where the contents of the file hist_obs_wells.txt may be something like:

```
ERROR_MODE  = RELMIN;
ERROR       = 0.25;
ERROR_MIN   = 100;
```

In this case, changing the file hist_obs_wells.txt will affect all of the observations.

Note that the keyword include can be used anywhere in the configuration file. However, nested inclusion (use of include in a file that has already been included with include) is not allowed.

By default, an observation entered with the HISTORY_OBSERVATION keyword will get the observed values, i.e. the 'true' values, from the WCONHIST and WCONINJH keywords in the schedule file provided to the ERT project. However it also possible to get the observed values from a reference case. In that case you must set set HISTORY_SOURCE variable in the ERT configuration file, see Creating a configuration file for ERT.

To change the observation error for a HISTORY_OBSERVATION for one or more segments of the historic period, you can use the SEGMENT keyword. For example:

```
HISTORY_OBSERVATION GWIR:FIELD
{
   ERROR       = 0.20;
   ERROR_MODE  = RELMIN;
   ERROR_MIN   = 100;

   SEGMENT FIRST_YEAR
   {
      START = 0;
      STOP  = 10;
      ERROR = 0.50;
      ERROR_MODE = REL;
   };

   SEGMENT SECOND_YEAR
   {
      START      = 11;
      STOP       = 20;
      ERRROR     = 1000;
      ERROR_MODE = ABS;
   };
};
```

The items START and STOP sets the start and stop of the segment in terms of ECLIPSE restart steps. The keywords ERROR, ERROR_MODE and ERROR_MIN behaves like before. If the segments overlap, they are computed in alphabetical order. Error covariance for "merged" updates

When merging the historical observations from several report steps together in one update the different steps are not independent, and it is beneficial to use a error covariance matrix, by using the keywords AUTO_CORRF and AUTO_CORRF_PARAM ERT will automatically estimate a error-covariance matrix based on the auto correlation function specified by the AUTO_CORRF keyword, with the parameter given by the AUTO_CORRF_PARAM parameter (i.e. the auto correlation length). The currently available auto correlation functions are:

EXP ~ exp(-x) GAUSS ~ exp(-x*x/2)

where the parameter x is given as:

x = (t2 - t1) / AUTO_CORRF_PARAM

5.3 The SUMMARY_OBSERVATION keyword

The keyword SUMMARY_OBSERVATION can be used to condition on any observation whos simulated value is written to the ECLIPSE summary file, e.g. well rates, region properties, group and field rates etc. A quite typical usage of SUMMARY_OBSERVATION is to condition on the results of a separator test.

Note: Although it is possible to condition on well and group rates with SUMMARY_OBSERVATION, it is usually easier to use HISTORY_OBSERVATION for this.

In order to create a summary observation, four pieces of information are needed: The observed value, the observation error, the time of observation and a summary key. A typical summary observation is created as follows:

```
SUMMARY_OBSERVATION SEP_TEST_2005
{
   VALUE = 100;
   ERROR =   5;
   DATE  = 21/08/2005;
   KEY   = GOPR:BRENT;
};
```

This will create an observation of group oil production for the brent group on 21th of august 2005. The observed value was 100 with a standard deviation of 5. The name SEP_TEST_2005 will be used as a label for the observation within the ERT and must be unique.

Similarly to the name of a HISTORY_OBSERVATION, the item KEY in a SUMMARY_OBSERVATION is used to look up the simulated value from the summary file. And again, as when declaring a HISTORY_OBSERVATION, to condition on VAR in well, group or region WGRNAME, one uses:

```
KEY = VAR:WGRNAME;
```

For example, to condition on RPPW in region 8, one uses:

```
KEY = RPPW:8;
```

It is also possible to give the observation time as a restart number using the RESTART item or as time in days from simulation start using the DAYS item. Here are two examples:

```
-- Giving the observation time in terms of restart number.
SUMMARY_OBSERVATION SEP_TEST_2005
{
   VALUE    = 100;
   ERROR    =   5;
   RESTART  =  42;
   KEY      = GOPR:BRENT;
};


-- Giving the observation time in terms of days
-- from simulation start.
SUMMARY_OBSERVATION SEP_TEST_2008
{
   VALUE    = 213;
   ERROR    =  10;
   DAYS     = 911;
```

(continues on next page)

(continued from previous page)

```
   KEY      = GOPR:NESS;
};
```

5.4 The BLOCK_OBSERVATION keyword

This is observations of variables in grid blocks/cells. The observations can be of arbitrary ECLIPSE fields like PRESSURE (typically for an RFT), PORO or PERM. A block observation is entered with the BLOCK_OBSERVATION keyword. Here is an example of a typical block observation:

```
BLOCK_OBSERVATION RFT_2006
{
   FIELD = PRESSURE;
   DATE  = 22/10/2006;

   OBS P1 { I = 1;  J = 1;  K = 1;   VALUE = 100;  ERROR = 5; };
   OBS P2 { I = 2;  J = 2;  K = 1;   VALUE = 101;  ERROR = 5; };
   OBS P3 { I = 2;  J = 3;  K = 1;   VALUE = 102;  ERROR = 5; };
};
```

This will condition on observations of the pressure in grid blocks (1,1,1), (2,2,1) and (2,3,1) on the 22/10/2006.

By default the BLOCK_OBSERVATION requires that the specific field which has been observed (e.g. PRESSURE in the example above) must have been specified in main ERT configuration file using the FIELD keyword, and ECLIPSE must be configured to produce a restart file for this particular time. Alternatively it is possible to tell ERT to use the summary vector as source of the data:

```
BLOCK_OBSERVATION RFT_2006
{
   FIELD = PRESSURE;
   DATE  = 22/10/2006;
   SOURCE = SUMMARY;

   OBS P1 { I = 1;  J = 1;  K = 1;   VALUE = 100;  ERROR = 5; };
   OBS P2 { I = 2;  J = 2;  K = 1;   VALUE = 101;  ERROR = 5; };
   OBS P3 { I = 2;  J = 3;  K = 1;   VALUE = 102;  ERROR = 5; };
};
```

In this case the data will be loaded from the BPR vectors in the summary file.

Note the use of the sub class OBS to specify the actUal observed values, the observation errors and their grid location. Each OBS shall have a unique key within the BLOCK_OBSERVATION instance, and is required to have the items I, J, K, VALUE and ERROR. These are the grid i,j and k indicies for the observation point, the observed value and it's standard deviation.

As with a SUMMARY_OBSERVATION, the observation time can be given as either a date, days since simulation start or restart number. The respective keys for setting giving it as date, days or restart number are DATE, DAYS and RESTART. Note that each BLOCK_OBSERVATION instance must have an unique global name (RFT_2006 in the example above).

Block observations can often be quite long. Thus, it is often a good idea to use the special keyword include in order to store the OBS structures in a different file. This is done as follows:

```
BLOCK_OBSERVATION RFT_2006
{
```

(continues on next page)

(continued from previous page)

```
   FIELD    = PRESSURE;
   RESTART = 20;

   include 'RFT_2006_OBS_DATA.txt';
};
```

Where the file RFT_2006_OBS_DATA.txt contains the OBS instances:

```
OBS P1 { I = 1;  J = 1;  K = 1;   VALUE = 100;  ERROR = 5; };
OBS P2 { I = 2;  J = 2;  K = 1;   VALUE = 101;  ERROR = 5; };
OBS P3 { I = 2;  J = 3;  K = 1;   VALUE = 112;  ERROR = 5; };
OBS P4 { I = 3;  J = 3;  K = 1;   VALUE = 122;  ERROR = 5; };
OBS P5 { I = 4;  J = 3;  K = 1;   VALUE = 112;  ERROR = 5; };
OBS P6 { I = 5;  J = 3;  K = 1;   VALUE = 122;  ERROR = 5; };
```

5.5 The GENERAL_OBSERVATION keyword

The GENERAL_OBSERVATION keyword is used together with the GEN_DATA and GEN_PARAM type. This pair of observation and data types are typically used when you want to update something special which does not fit into any of the predefined enkf types. The ERT application just treats GENERAL_OBSERVATION (and also GEN_DATA) as a range of number with no particular structure, this is very flexible, but of course also a bit more complex to use:

```
GENERAL_OBSERVATION GEN_OBS1{
   DATA     = SOME_FIELD;
   RESTART  = 20;
   OBS_FILE = some_file.txt;
};
```

This example a minimum GENERAL_OBSERVATION. The keyword DATA points to the GEN_DATA instance this observation is 'observing', RESTART gives the report step when this observation is active. OBS_FILE should be the name of a file with observation values, and the corresponding uncertainties. The file with observations should just be a plain text file with numbers in it, observations and corresponding uncertainties interleaved. An example of an OBS_FILE:

```
1.46 0.26
25.0 5.0
5.00 1.00
```

This OBS_FILE has three observations: 1.46 +/- 0.26, 25.0 +/- 5.0 and 5.00 +/- 1.00. In the example above it is assumed that the DATA instance we are observing (i.e. comparing with) has the same number of elements as the observation, i.e. three in this case. By using the keywords INDEX_LIST or INDEX_FILE you can select the elements of the GEN_DATA instance you are interested in. Consider for example:

```
GENERAL_OBSERVATION GEN_OBS1{
   DATA       = SOME_FIELD;
   INDEX_LIST = 0,3,9;
   RESTART    = 20;
   OBS_FILE   = some_file.txt;
};
```

Here we use INDEX_LIST to indicate that we are interested in element 0,3 and 9 of the GEN_DATA instance:

```
GEN_DATA                        GEN_OBS1
========                        ===========
1.56 <----------------------> 1.46  0.26
23.0         /--------------> 25.0   5.00
56.0         |    /---------> 5.00  1.00
27.0 <-------/    |            ===========
 0.2              |
1.56              |
1.78              |
6.78              |
9.00              |
4.50 <------------/
========
```

In addition to INDEX_LIST it is possible to use INDEX_FILE which should just point at an plain text file with indexes (without any ',' or anything). Finally, if your observation only has one value, you can embed it in the config object with VALUE and ERROR.

5.5.1 Matching GEN_OBS and GEN_DATA

It is important to match up the GEN_OBS observations with the corresponding GEN_DATA simulation data correctly. The GEN_DATA result files must have an embedded '%d' to indicate the report step in them - in the case of smoother based workflows the actual numerical value here is not important. To ensure that GEN_OBS and corresponding GEN_DATA values match up correctly only the RESTART method is allowed for GEN_OBS when specifying the time. So consider a setup like this:

```
-- Config file:
GEN_DATA RFT_BH67 INPUT_FORMAT:ASCII RESULT_FILE:rft_BH67_%d    REPORT_STEPS:20
...                                                            /|\                /|\
...                                                             |                  |
-- Observation file:                                            |                  |
GENERAL_OBSERVATION GEN_OBS1{                                   +------------------/
   DATA       = RFT_BH67;                                       |
   RESTART    = 20;   <-----------------------------------------/
   OBS_FILE   = some_file.txt;
};
```

Here we see that the observation is active at report step 20, and we expect the forward model to create a file rft_BH67_20 in each realization directory. Error covariance

The optional keyword ERROR_COVAR can be used to point to an existing file, containing an error covariance matrix. The file should contain the elements of the matrix as formatted numbers; newline formatting is allowed but not necessary. Since the matrix should by construction be symmetric there is no difference between column-major and row-major order! The covariance matrix

[1 0.75 -0.25]

C = [0.75 1.25 -0.50] [-0.25 -0.50 0.85]

Can be represented by the file:

```
1
0.75
-0.25
0.75
1.25
```

(continues on next page)

(continued from previous page)

```
-0.50
-0.25
-0.50
0.85
```

without newlines, or alternatively:

```
1       0.75  -0.25
0.75    1.25  -0.50
-0.25  -0.50   0.85
```

with newlines.

CHAPTER

SIX

ECLIPSE - OR NOT

In the very first versions of ERT it was all about running Eclipse - there was no general forward model. The concept of general forward model came quite quickly and since then the special treatment of Eclipse has gradually been reduced. It is now possible to run a forward model without Eclipse at all, but ERT is still quite far from a situation where Eclipse is just any other forward model. For good and for bad there is quite a lot of functionality in ERT which is specifically targeted at Eclipse.

6.1 Eclipse keywords

DATA_FILE This is the name of ECLIPSE data file used to control the simulations. The datafile used with this keyword should be prepared for use with ERT, and will generally not be a 100% valid ECLIPSE data file:

```
-- Load the data file called ECLIPSE.DATA
DATA_FILE ECLIPSE.DATA
```

The preparations you should do to this file include:

1. You should insert `INCLUDE` statements to include the various uncertainty parameters in use at the right place in the datafile.
2. You must make sure that the include files used in the datafiles can be correctly resolved from the runpath location.
3. See the `DATA_KW` keyword which can be used to utilize more template functionality in the eclipse datafile.

When actually running the simulations the file given with the `DATA_FILE` keyword will be used as a template, and a realization specific datafile will be written to the runpath folder, the name of this file is given by the ECLBASE keyword.

ECLBASE This is the name of the eclipse simulations which will be generated by ERT, the `ECLBASE` argument should typically contain a `%d` format specifier, which will be replaced with the realisation number when ERT creates the runpath directories:

```
 ECLBASE eclipse/CASE_%d

Observe that it is possible to add a leading directory part to the ``ECLBASE``
setting as illustrated above; the directory ``eclipse/`` will be generated as
a subfolder in the runpath directories.
```

GRID This should be the name of an existing GRID/EGRID file. The main use of the grid is to map out active and inactive cells when using `FIELD` data. If you do not use `FIELD` data you do not need the `GRID` keyword. The grid argument will only be used by the main ERT application and not passed down to the forward model in any way.

SCHEDULE_FILE DEPRECATED!

6.2 Running with a different reservoir simulator

6.3 Running without a reservoir simulator

CHAPTER

SEVEN

THE SMOOTHER UPDATE IN ERT

CHAPTER

EIGHT

USE THE ERT API TO CREATE CUSTOM FUNCTIONALITY

CHAPTER

NINE

KEYWORDS FOR THE CONFIGURATION FILE

Go to main ERT page

9.1 General overview

The enkf application is started with a single argument, which is the name of the configuration file to be used. The enkf configuration file serves several purposes, which are:

- Defining which ECLIPSE model to use, i.e. giving a data, grid and schedule file.
- Defining which observation file to use.
- Defining how to run simulations.
- Defining where to store results.
- Creating a parametrization of the ECLIPSE model.

The configuration file is a plain text file, with one statement per line. The first word on each line is a keyword, which then is followed by a set of arguments that are unique to the particular keyword. Except for the DEFINE keyword, ordering of the keywords is not significant. Similarly to ECLIPSE data files, lines starting with "–" are treated as comments.

The keywords in the enkf configuration file can roughly be divded into two groups:

- Basic required keywords not related to parametrization. I.e., keywords giving the data, grid, schedule and observation file, defining how to run simulations and how to store results. These keywords are described in *Basic required keywords.*
- Basic optional keywords not related to parametrization. These keywords are described in *Basic optional keywords*.
- Keywords related to parametrization of the ECLIPSE model. These keywords are described in *Parametrization keywords*.
- Advanced keywords not related to parametrization. These keywords are described in *Advanced optional keywords*.

9.2 List of keywords

Keyword name	Required by user?	Default value	Purpo
ADD_FIXED_LENGTH_SCHEDULE_KW	NO		Suppo

Keyword name	Required by user?	Default value	Purpo
ANALYSIS_COPY	NO		Create
ANALYSIS_LOAD	NO		Load
ANALYSIS_SET_VAR	NO		Set an
ANALYSIS_SELECT	NO	STD_ENKF	Select
CASE_TABLE	NO		For ru
CONTAINER	NO		. . .
CUSTOM_KW	NO		Abilit
DATA_FILE	YES		Provi
DATA_KW	NO		Repla
DBASE_TYPE	NO	BLOCK_FS	Whic
DEFINE	NO		Defin
DELETE_RUNPATH	NO		Explic
ECLBASE	YES		Defin
END_DATE	NO		You c
ENKF_ALPHA	NO	1.50	Param
ENKF_BOOTSTRAP	NO	FALSE	Shoul
ENKF_CROSS_VALIDATION	NO	. . .	
ENKF_CV_FOLDS	NO	10	Numb
ENKF_KERNEL_PARAM	NO	1	
ENKF_LOCAL_CV	NO	FALSE	Shoul
ENKF_MERGE_OBSERVATIONS	NO	FALSE	Shoul
ENKF_MODE	NO	STANDARD	Whic
ENKF_PEN_PRESS	NO	FALSE	Shoul
ENKF_RERUN	NO	FALSE	Shoul
ENKF_SCALING	NO	TRUE	Do we
ENKF_TRUNCATION	NO	0.99	Cutof
ENSPATH	NO	storage	Folde
FIELD	NO		Ads g
FORWARD_MODEL	NO		Add t
GEN_DATA	NO		Speci
GEN_KW	NO		Add a
GEN_KW_TAG_FORMAT	NO	<%s>	Forma
GEN_KW_EXPORT_FILE	NO	parameter.txt	Name
GEN_PARAM	NO		Add a
GRID	NO		Provi
HISTORY_SOURCE	NO	REFCASE_HISTORY	Sourc
HOOK_WORKFLOW	NO		Instal
IGNORE_SCHEDULE	NO		
INSTALL_JOB	NO		Instal
ITER_CASE	NO	IES%d	Case
ITER_COUNT	NO	4	Numb
ITER_RETRY_COUNT	NO	4	Numb
JOBNAME	NO		Name
JOB_SCRIPT	NO		Pytho
LOAD_SEED	NO		Load
LOAD_WORKFLOW	NO		Load
LOAD_WORKFLOW_JOB	NO		Load
LICENSE_PATH	NO		A pat
LOCAL_CONFIG	NO		A file

Keyword name	Required by user?	Default value	Purpo
LOG_FILE	NO	log	Name
LOG_LEVEL	NO	1	How r
LSF_QUEUE	NO	normal	Name
LSF_RESOURCES	NO		
LSF_SERVER	NO		Set se
MAX_ITER_COUNT	NO		Maxir
MAX_RESAMPLE	NO	1	How r
MAX_RUNNING_RSH	NO		The m
MAX_RUNTIME	NO	0	Set th
MAX_SUBMIT	NO	2	How r
MIN_REALIZATIONS	NO	0	Set th
NUM_REALIZATIONS	YES		Set th
OBS_CONFIG	NO		File sp
PLOT_SETTINGS	NO		Possib
PRE_CLEAR_RUNPATH	NO	FALSE	Shoul
QUEUE_SYSTEM	NO		Syster
REFCASE	NO (see HISTORY_SOURCE and SUMMARY)		Refer
REFCASE_LIST	NO		Full p
RERUN_PATH	NO		. . .
RERUN_START	NO	0	. . .
RFT_CONFIG	NO		Confi
RFTPATH	NO	rft	Path t
RSH_COMMAND	NO		Comn
RSH_HOST	NO		Remo
RUNPATH	NO	simulations/realization%d	Direct
RUN_TEMPLATE	NO		Instal
STD_SCALE_CORRELATED_OBS	NO	FALSE	Try to
SCHEDULE_FILE	NO		Provi
SCHEDULE_PREDICTION_FILE	NO		Sched
SETENV	NO		You c
SINGLE_NODE_UPDATE	NO	FALSE	. . .
STOP_LONG_RUNNING	NO	FALSE	Stop l
STORE_SEED	NO		File w
SUMMARY	NO		Add s
SURFACE	NO		Surfa
TORQUE_QUEUE	NO		. . .
TIME_MAP	NO		Abilit
UMASK	NO		Contr
UPDATE_LOG_PATH	NO	update_log	Summ
UPDATE_PATH	NO		Modif
UPDATE_SETTINGS	NO		Possib
WORKFLOW_JOB_DIRECTORY	NO		Direct

:ref:‘ <>‘

9.3 Basic required keywords

These keywords must be set to make the enkf function properly.

DATA_FILE

This is the name of ECLIPSE data file used to control the simulations. The

data file should be prepared according to the guidelines given in Preparing an ECLIPSE reservoir model for use with enkf.

Example:

```
-- Load the data file called ECLIPSE.DATA
DATA_FILE ECLIPSE.DATA
```

ECLBASE

The ECLBASE keyword sets the basename used for the ECLIPSE simulations. It

can (and should, for your convenience) contain a %d specifier, which will be replaced with the realization numbers when running ECLIPSE. Note that due to limitations in ECLIPSE, the ECLBASE string must be in strictly upper or lower case.

Example:

```
-- Use MY_VERY_OWN_OIL_FIELD-0 etc. as basename.
-- When ECLIPSE is running, the %d will be,
-- replaced with realization number, giving:
--
-- MY_VERY_OWN_OIL_FIELD-0
-- MY_VERY_OWN_OIL_FIELD-1
-- MY_VERY_OWN_OIL_FIELD-2
-- ...
-- and so on.
ECLBASE MY_VERY_OWN_OIL_FIELD-%d
```

JOBNAME

As an alternative to the ECLBASE keyword you can use the JOBNAME keyword; in particular in cases where your forward model does not include ECLIPSE at all that makes more sense. If JOBANME is used instead of ECLBASE the same rules of no-mixed-case apply.

GRID

This is the name of an existing GRID/EGRID file for your ECLIPSE model. If you had to create a new grid file when preparing your ECLIPSE reservoir model for use with enkf, this should point to the new .EGRID file.

Example:

```
-- Load the .EGRID file called MY_GRID.EGRID
GRID MY_GRID.EGRID
```

NUM_REALIZATIONS

This is just the size of the ensemble, i.e. the number of realizations/members in the ensemble.

Example:

```
-- Use 200 realizations/members
NUM_REALIZATIONS 200
```

SCHEDULE_FILE

This keyword should be the name a text file containing the SCHEDULE section of the ECLIPSE data file. It should be prepared in accordance with the guidelines given in Preparing an ECLIPSE reservoir model for use with enkf. This SCHEDULE section will be used to control the ECLIPSE simulations. You can optionally give a second filename, which is the name of file which will be written into the directories for running ECLIPSE.

Example:

```
-- Parse MY_SCHEDULE.SCH, call the generated file ECLIPSE_SCHEDULE.SCH
SCHEDULE_FILE MY_SCHEDULE.SCH ECLIPSE_SCHEDULE.SCH
```

Observe that the SCHEDULE_FILE keyword is only required when you need ERT to stop and restart your simulations; i.e. when you are using the EnKF algorithm. If you are only using ERT to your simulations; or using smoother update it is recommended to leave the SCHEDULE_FILE keyword out. In that case you must make sure that the ECLIPSE datafile correctly includes the SCHEDULE section.

9.4 Basic optional keywords

These keywords are optional. However, they serve many useful purposes, and it is recommended that you read through this section to get a thorough idea of what's possible to do with the enkf application.

DATA_KW

The keyword DATA_KW can be used for inserting strings into placeholders in the ECLIPSE data file. For instance, it can be used to insert include paths.

Example:

```
-- Define the alias MY_PATH using DATA_KW. Any instances of <MY_PATH> (yes, with␣
↪brackets)
-- in the ECLIPSE data file will now be replaced with /mnt/my_own_disk/my_reservoir_
↪model
-- when running the ECLIPSE jobs.
DATA_KW  MY_PATH  /mnt/my_own_disk/my_reservoir_model
```

The DATA_KW keyword is of course optional. Note also that the enkf has some built in magic strings.

DELETE_RUNPATH

When the ert application is running it creates directories for the forward model simulations, one for each realization. When the simulations are done, ert will load the results into the internal database. By default the realization folders will be left intact after ert has loaded the results, but using the keyword DELETE_RUNPATH you can request to have (some of) the directories deleted after results have been loaded.

Example A:

```
-- Delete simulation directories 0 to 99
DELETE_RUNPATH 0-99
```

Example B:

```
-- Delete simulation directories 0 to 10 as well as 12, 15 and 20.
DELETE_RUNPATH 0 - 10, 12, 15, 20
```

The DELETE_RUNPATH keyword is optional.

END_DATE

When running a set of models from beginning to end ERT does not now in advance how long the simulation is supposed to be, it is therefor impossible beforehand to determine which restart file number should be used as target file, and the procedure used for EnKF runs can not be used to verify that an ECLIPSE simulation has run to the end.

By using the END_DATE keyword you can tell ERT that the simulation should go at least up to the date given by END_DATE, otherwise they will be regarded as failed. The END_DATE does not need to correspond exactly to the end date of the simulation, it must just be set so that all simulations which go to or beyond END_DATE are regarded as successfull.

Example:

:: END_DATE 10/10/2010

With this END_DATE setting all simulations which have gone to at least 10.th of October 2010 are OK.

ENSPATH

The ENSPATH should give the name of a folder that will be used for storage by the enkf application. Note that the contents of this folder is not intended for human inspection. By default, ENSPATH is set to "storage".

Example:

```
-- Use internal storage in /mnt/my_big_enkf_disk
ENSPATH /mnt/my_big_enkf_disk
```

The ENSPATH keyword is optional.

HISTORY_SOURCE

In the observation configuration file you can enter observations with the keyword HISTORY_OBSERVATION; this means that ERT will the observed 'true' values from the model history. Practically the historical values can be fetched either from the SCHEDULE file or from a reference case. What source to use for the historical values can be controlled with the HISTORY_SOURCE keyword. The different possible values for the HISTORY_SOURCE

keyword are:

REFCASE_HISTORY This is the default value for HISTORY_SOURCE, ERT will fetch the historical values from the *xxxH* keywords in the refcase summary, e.g. observations of WGOR:OP_1 is based the WGORH:OP_1 vector from the refcase summary.

REFCASE_SIMULATED In this case the historical values are based on the simulated values from the refcase, this is mostly relevant when a you want compare with another case which serves as 'the truth'.

SCHEDULE Load historical values from the WCONHIST and WCONINJE keywords in the Schedule file.

When setting HISTORY_SOURCE to either REFCASE_SIMULATED or REFCASE_HISTORY you must also set the REFCASE variable to point to the ECLIPSE data file in an existing reference case (should be created with the same schedule file as you are using now).

Example:

```
-- Use historic data from reference case
HISTORY_SOURCE  REFCASE_HISTORY
REFCASE         /somefolder/ECLIPSE.DATA
```

The HISTORY_SOURCE keyword is optional.

REFCASE

With the REFCASE key you can supply ert with a reference case which can be used for observations (see HISTORY_SOURCE), if you want to use wildcards with the SUMMARY keyword you also must supply a REFCASE keyword. The REFCASE keyword should just point to an existing ECLIPSE data file; ert will then look up and load the corresponding summary results.

Example:

```
-- The REFCASE keyword points to the datafile of an existing ECLIPSE simulation.
REFCASE /path/to/somewhere/SIM_01_BASE.DATA
```

INSTALL_JOB

The INSTALL_JOB keyword is used to learn the enkf application how to run external applications and scripts, i.e. defining a job. After a job has been defined with INSTALL_JOB, it can be used with the FORWARD_MODEL keyword. For example, if you have a script which generates relative permeability curves from a set of parameters, it can be added as a job, allowing you to do history matching and sensitivity analysis on the parameters defining the relative permeability curves.

The INSTALL_JOB keyword takes two arguments, a job name and the name of a configuration file for that particular job.

Example:

```
-- Define a Lomeland relative permeabilty job.
-- The file jobs/lomeland.txt contains a detailed
-- specification of the job.
INSTALL_JOB LOMELAND jobs/lomeland.txt
```

The configuration file used to specify an external job is easy to use and very flexible. It is documented in Customizing the simulation workflow in enkf.

The INSTALL_JOB keyword is optional.

OBS_CONFIG

The OBS_CONFIG key should point to a file defining observations and associated uncertainties. The file should be in plain text and formatted according to the guidelines given in Creating an observation file for use with enkf.

Example:

```
-- Use the observations in my_observations.txt
OBS_CONFIG my_observations.txt
```

The OBS_CONFIG keyword is optional, but for your own convenience, it is strongly recommended to provide an observation file.

RESULT_PATH

The enkf application will print some simple tabulated results at each report step. The RESULT_PATH keyword should point to a folder where the tabulated results are to be written. It can contain a %d specifier, which will be replaced with the report step by enkf. The default value for RESULT_PATH is "results/step_%d".

Example:

```
-- Changing RESULT_PATH
RESULT_PATH my_nice_results/step-%d
```

The RESULT_PATH keyword is optional.

RUNPATH

The RUNPATH keyword should give the name of the folders where the ECLIPSE simulations are executed. It should contain at least one %d specifier, which will be replaced by the realization number when the enkf creates the folders. Optionally, it can contain one more %d specifier, which will be replaced by the iteration number.

By default, RUNPATH is set to "simulations/realization-%d".

Example A:

:: – Giving a RUNPATH with just one %d specifer. RUNPATH /mnt/my_scratch_disk/realization-%d

Example B:

```
-- Giving a RUNPATH with two %d specifers.
RUNPATH /mnt/my_scratch_disk/realization-%d/iteration-%d
```

The RUNPATH keyword is optional.

When running workflows based on external scripts it is necessary to 'tell' the external script in some way or another were all the realisations are located in the filesystem. Since the number of realisations can be quite high this will easily overflow the commandline buffer; the solution which is used is therefor to let ert write a reagular file which looks like this:

```
0   /path/to/realisation0   CASE0   iter
1   /path/to/realisation1   CASE1   iter
...
N   /path/to/realisationN   CASEN   iter
```

The path to this file can then be passed to the scripts using the magic string <RUNPATH_FILE>. The RUNPATH_FILE

will by default be stored as .ert_runpath_list in the same directory as the configuration file, but you can set it to something else with the RUNPATH_FILE key.

9.5 Keywords controlling the simulations

MIN_REALIZATIONS

MIN_REALIZATIONS is the minimum number of realizations that must have succeeded for the simulation to be regarded as a success.

MIN_REALIZATIONS can also be used in combination with STOP_LONG_RUNNING, see the documentation for STOP_LONG_RUNNING for a description of this.

Example:

```
MIN_REALIZATIONS  20
```

The MIN_REALIZATIONS key can also be set as a percentage of NUM_REALIZATIONS

```
MIN_REALIZATIONS  10%
```

The MIN_REALIZATIONS key is optional, but if it has not been set *all* the realisations must succeed.

STOP_LONG_RUNNING

The STOP_LONG_RUNNING key is used in combination with the MIN_REALIZATIONS key to control the runtime of simulations. When STOP_LONG_RUNNING is set to TRUE, MIN_REALIZATIONS is the minimum number of realizations run before the simulation is stopped. After MIN_REALIZATIONS have succeded successfully, the realizatons left are allowed to run for 25% of the average runtime for successfull realizations, and then killed.

Example:

```
-- Stop long running realizations after 20 realizations have succeeded
MIN_REALIZATIONS  20
STOP_LONG_RUNNING TRUE
```

The STOP_LONG_RUNNING key is optional. The MIN_REALIZATIONS key must be set when STOP_LONG_RUNNING is set to TRUE.

MAX_RUNTIME

The MAX_RUNTIME keyword is used to control the runtime of simulations. When MAX_RUNTIME is set, a job is only allowed to run for MAX_RUNTIME, given in seconds. A value of 0 means unlimited runtime.

Example:

```
-- Let each realizations run for 50 seconds
MAX_RUNTIME 50
```

The MAX_RUNTIME key is optional.

9.6 Parameterization keywords

The keywords in this section are used to define a parametrization of the ECLIPSE model. I.e., defining which parameters to change in a sensitivity analysis and/or history matching project. For some parameters, it necessary to specify a prior distribution. See Prior distributions available in enkf for a complete list of available priors.

FIELD

The FIELD keyword is used to parametrize quantities which have extent over the full grid. Both dynamic properties like pressure, and static properties like porosity, are implemented in terms of FIELD objects. When adding fields in the config file the syntax is a bit different for dynamic fields (typically solution data from ECLIPSE) and parameter fields like permeability and porosity.

Dynamic fields

To add a dynamic field the entry in the configuration file looks like this:

:: FIELD <ID> DYNAMIC MIN:X MAX:Y

In this case ID is not an arbitrary string; it must coincide with the keyword name found in the ECLIPSE restart file, e.g. PRESSURE. Optionally, you can add a minimum and/or a maximum value with MIN:X and MAX:Y.

Example A:

```
-- Adding pressure field (unbounded)
FIELD PRESSURE DYNAMIC
```

Example B:

```
-- Adding a bounded water saturation field
FIELD SWAT DYNAMIC MIN:0.2 MAX:0.95
```

Parameter fields

A parameter field (e.g. porosity or permeability) is defined as follows:

```
FIELD  ID PARAMETER   <ECLIPSE_FILE>  INIT_FILES:/path/%d  MIN:X MAX:Y OUTPUT_
↪TRANSFORM:FUNC INIT_TRANSFORM:FUNC
```

Here ID is again an arbitrary string, ECLIPSE_FILE is the name of the file the enkf will export this field to when running simulations. Note that there should be an IMPORT statement in the ECLIPSE data file corresponding to the name given with ECLIPSE_FILE. INIT_FILES is a filename (with an embedded %d) to load the initial field from. Can be RMS ROFF format, ECLIPSE restart format or ECLIPSE GRDECL format.

The input arguments MIN, MAX, INIT_TRANSFORM and OUTPUT_TRANSFORM are all optional. MIN and MAX are as for dynamic fields.

For Assisted history matching, the variables in ERT should be normally distributed internally - the purpose of the transformations is to enable working with normally distributed variables internally in ERT. Thus, the optional arguments INIT_TRANSFORM:FUNC and OUTPUT_TRANSFORM:FUNC are used to transform the user input of parameter distribution. INIT_TRANSFORM:FUNC is a function which will be applied when they are loaded to ERT. OUTPUT_TRANSFORM:FUNC is a function which will be applied to the field when it is exported from ERT, and FUNC is the name of a transformation function to be applied. The avaialble functions are listed below:

“POW10” : This function will raise x to the power of 10: y = 10^x. “TRUNC_POW10” : This function will raise x to the power of 10 - and truncate lower values at 0.001. “LOG” : This function will take the NATURAL logarithm of x: y = ln(x). “LN” : This function will take the NATURAL logarithm of x: y = ln(x). “LOG10” : This function will take the log10 logarithm of x: y = log10(x). “EXP” : This function will calculate y = exp(x). “LN0” : This function will calculate y = ln(x + 0.000001 “EXP0” : This function will calculate y = exp(x) - 0.000001

For example, the most common scenario is that underlying log-normal distributed permeability in RMS are transformed to normally distributted in ERT, then you do:

INIT_TRANSFORM:LOG To ensure that the variables which were initially log-normal distributed are transformed to normal distribution when they are loaded into ert.

OUTPUT_TRANSFORM:EXP To ensure that the variables are reexponentiated to be log-normal distributed before going out to Eclipse.

If users specify the wrong function name (e.g INIT_TRANSFORM:I_DONT_KNOW), ERT will stop and print all the valid function names.

Regarding format of ECLIPSE_FILE: The default format for the parameter fields is binary format of the same type as used in the ECLIPSE restart files. This requires that the ECLIPSE datafile contains an IMPORT statement. The advantage with using a binary format is that the files are smaller, and reading/writing is faster than for plain text files. If you give the ECLIPSE_FILE with the extension .grdecl (arbitrary case), enkf will produce ordinary .grdecl files, which are loaded with an INCLUDE statement. This is probably what most users are used to beforehand - but we recomend the IMPORT form.

General fields

In addition to dynamic and parameter field there is also a general field, where you have fine grained control over input/output. Use of the general field type is only relevant for advanced features. The arguments for the general field type are as follows:

```
FIELD   ID  GENERAL    FILE_GENERATED_BY_ENKF  FILE_LOADED_BY_ENKF    <OPTIONS>
```

The OPTIONS argument is the same as for the parameter field.

GEN_DATA

The GEN_DATA keyword is used when estimating data types which enkf does not know anything about. GEN_DATA is very similar to GEN_PARAM, but GEN_DATA is used for data which are updated/created by the forward model like e.g. seismic data. In the main configuration file the input for a GEN_DATA instance is as follows:

```
GEN_DATA  ID RESULT_FILE:yyy INPUT_FORMAT:xx  REPORT_STEPS:10,20 ␣
↪ECL_FILE:xxx  OUTPUT_FORMAT:xx  INIT_FILES:/path/files%d␣
↪TEMPLATE:/template_file TEMPLATE_KEY:magic_string
```

The GEN_DATA keyword has many options; in many cases you can leave many of them off. We therefor list the required and the optional options separately:

Required GEN_DATA options

- RESULT_FILE - This if the name the file generated by the forward model and

read by ERT. This filename _must_ have a %d as part of the name, that %d will be replaced by report step when loading.

- INPUT_FORMAT - The format of the file written by the forward model (i.e.

RESULT_FILE) and read by ERT, valid values are ASCII, BINARY_DOUBLE and BINARY_FLOAT.

- REPORT_STEPS A list of the report step(s) where you expect the forward model

to create a result file. I.e. if the forward model should create a result file for report steps 50 and 100 this setting should be: REPORT_STEPS:50,100. If you have observations of this GEN_DATA data the RESTART setting of the corresponding GENERAL_OBSERVATION must match one of the values given by REPORT_STEPS.

Optional GEN_DATA options

- ECL_FILE - This is the name of file written by enkf to be read by the

forward model.

- OUTPUT_FORMAT - The format of the files written by enkf and read by the

forward model, valid values are ASCII, BINARY_DOUBLE, BINARY_FLOAT and ASCII_TEMPLATE. If you use ASCII_TEMPLATE you must also supply values for TEMPLATE and TEMPLATE_KEY.

- INIT_FILES - Format string with '%d' of files to load the initial data from.

Example:

```
GEN_DATA 4DWOC  INPUT_FORMAT:ASCII   RESULT_FILE:SimulatedWOC%d.
↪txt   REPORT_STEPS:10,100
```

Here we introduce a GEN_DATA instance with name 4DWOC. When the forward model has run it should create two files with name SimulatedWOC10.txt and SimulatedWOC100.txt. The result files are in ASCII format, ERT will look for these files and load the content. The files should be pure numbers - without any header.

**Observe that the GEN_DATA RESULT_FILE setting must have a %d format

specifier, that will be replaced with the report step..**

CUSTOM_KW

The keyword CUSTOM_KW enables custom data key:value pairs to be stored in ERT storage. Custom KW has many similarities to Gen KW and Gen Data but is fully defined by the user and contain only key_value pairs.

Example:

```
CUSTOM_KW GROUP_NAME <input_file>

--GROUP_NAME
This is similar to Gen KW where every keyword is prefixed with the GROUP_NAME like␣
↪this: GROUP_NAME:KEYWORD

--input_file
This is the input file expected to be generated by a forward model.

--Example
CUSTOM_KW COMPOSITION composition.txt
```

With this setup ERT will expect the file composition.txt to be present in the runpath. This file may look like this

```
oil 0.5
water 0.2
gas 0.2
unknown 0.1
state good
```

Every key-value pair must be a string followed by a space and a value. The value can either be a number or a string (all numbers are interpreted as floats).

After a successful run, ERT will store the COMPOSITION Custom KW in its filesystem and will be available for every realization. An export will present the values produced as:

- COMPOSITION:oil
- COMPOSITION:water
- COMPOSITION:gas
- COMPOSITION:unknown
- COMPOSITION:state

GEN_KW

The GEN_KW (abbreviation of general keyword) parameter is based on a template file and substitution. In the main config file a GEN_KW instance is defined as follows:

```
GEN_KW  ID  my_template.txt  my_eclipse_include.txt  my_priors.txt
```

Here ID is an (arbitrary) unique string, my_template.txt is the name of a template file, my_eclipse_include.txt is the name of the file which is made for each member based on my_template.txt and my_priors.txt is a file containing a list of parametrized keywords and a prior distribution for each. Note that you must manually edit the ECLIPSE data file so that my_eclipse_include.txt is included.

Let us consider an example where the GEN_KW parameter type is used to estimate pore volume multipliers. We would then declare a GEN_KW instance in the main enkf configuration file:

```
GEN_KW PAR_MULTPV multpv_template.txt multpv.txt multpv_priors.txt
```

In the GRID or EDIT section of the ECLIPSE data file, we would insert the following include statement:

```
INCLUDE
 'multpv.txt' /
```

The template file multpv_template.txt would contain some parametrized ECLIPSE statements:

```
BOX
 1 10 1 30 13 13 /
MULTPV
 300*<MULTPV_BOX1> /
ENDBOX

BOX
 1 10 1 30 14 14 /
MULTPV
 300*<MULTPV_BOX2> /
ENDBOX
```

Here, <MULTPV_BOX1> and <MULTPV_BOX2> will act as magic strings. Note that the '<' '>' must be present around the magic strings. In this case, the parameter configuration file multpv_priors.txt could look like this:

```
MULTPV_BOX2 UNIFORM 0.98 1.03
MULTPV_BOX1 UNIFORM 0.85 1.00
```

In general, the first keyword on each line in the parameter configuration file defines a key, which when found in the template file enclosed in '<' and '>', is replaced with a value. The rest of the line defines a prior distribution for the key. See Prior distributions available in enkf for a list of available prior distributions.

Example: Using GEN_KW to estimate fault transmissibility multipliers

Previously enkf supported a datatype MULTFLT for estimating fault transmissibility multipliers. This has now been depreceated, as the functionality can be easily achieved with the help of GEN_KW. In th enkf config file:

```
GEN_KW  MY-FAULTS   MULTFLT.tmpl   MULTFLT.INC   MULTFLT.txt
```

Here MY-FAULTS is the (arbitrary) key assigned to the fault multipliers, MULTFLT.tmpl is the template file, which can look like this:

```
MULTFLT
 'FAULT1'   <FAULT1>  /
 'FAULT2'   <FAULT2>  /
/
```

and finally the initial distribution of the parameters FAULT1 and FAULT2 are defined in the file MULTFLT.txt:

```
FAULT1   LOGUNIF   0.00001   0.1
FAULT2   UNIFORM   0.00      1.0
```

The various prior distributions available for the `GEN_KW` keyword are here prior distributions available in ERT

Loading GEN_KW values from an external file

The default use of the GEN_KW keyword is to let the ERT application sample random values for the elements in the GEN_KW instance, but it is also possible to tell ERT to load a precreated set of data files, this can for instance be used as a component in a experimental design based workflow. When using external files to initialize the GEN_KW instances you supply an extra keyword `INIT_FILE:/path/to/priors/files%d` which tells where the prior files are:

```
GEN_KW  MY-FAULTS   MULTFLT.tmpl   MULTFLT.INC   MULTFLT.txt    INIT_FILES:priors/
↪multflt/faults%d
```

In the example above you must prepare files priors/multflt/faults0, priors/multflt/faults1, ... priors/multflt/faultsn which ert will load when you initialize the case. The format of the GEN_KW input files can be of two varieties:

1. The files can be plain ASCII text files with a list of numbers:

```
1.25
2.67
```

The numbers will be assigned to parameters in the order found in the MULTFLT.txt file.

2. Alternatively values and keywords can be interleaved as in:

```
FAULT1 1.25
FAULT2 2.56
```

in this case the ordering can differ in the init files and the parameter file.

The heritage of the ERT program is based on the EnKF algorithm, and the EnKF algorithm evolves around Gaussian variables - internally the GEN_KW variables are assumed to be samples from the N(0,1) distribution, and the distributions specified in the parameters file are based on transformations starting with a N(0,1) distributed variable. The slightly awkward consequence of this is that to let your sampled values pass through ERT unmodified you must configure the distribution NORMAL 0 1 in the parameter file; alternatively if you do not intend to update the GEN_KW variable you can use the distribution RAW.

GEN_PARAM

The GEN_PARAM parameter type is used to estimate parameters which do not really fit into any of the other categories. As an example, consider the following situation:

Some external Software (e.g. Cohiba) makes a large vector of random numbers which will serve as input to the forward model. (It is no requirement that the parameter set is large, but if it only consists of a few parameters the GEN_KW type will be easier to use.) We want to

update this parameter with enkf. In the main configuration file the input for a GEN_PARAM instance is as follows:

```
GEN_PARAM  ID  ECLIPSE_FILE  INPUT_FORMAT:xx  OUTPUT_FORMAT:xx
↪INIT_FILES:/path/to/init/files%d (TEMPLATE:/template_file
↪KEY:magic_string)
```

here ID is the usual unique string identifying this instance and ECLIPSE_FILE is the name of the file which is written into the run directories. The three arguments GEN_PARAM, ID and ECLIPSE_FILE must be the three first arguments. In addition you must have three additional arguments, INPUT_FORMAT, OUTPUT_FORMAT and INIT_FILES. INPUT_FORMAT is the format of the files enkf should load to initialize, and OUTPUT_FORMAT is the format of the files enkf writes for the forward model. The valid values are:

- ASCII - This is just text file with formatted numbers.
- ASCII_TEMPLATE - An plain text file with formatted numbers, and an arbitrary

header/footer.

- BINARY_FLOAT - A vector of binary float numbers.
- BINARY_DOUBLE - A vector of binary double numbers.

Regarding the different formats - observe the following:

1. Except the format ASCII_TEMPLATE the files contain no header information.
2. The format ASCII_TEMPLATE can only be used as output format.
3. If you use the output format ASCII_TEMPLATE you must also supply a

TEMPLATE:X and KEY:Y option. See documentation of this below.

1. For the binary formats files generated by Fortran can not be used - can

easily be supported on request.

Regarding templates: If you use OUTPUT_FORMAT:ASCII_TEMPLATE you must also

supply the arguments TEMPLATE:/template/file and KEY:MaGiCKEY. The template file is an arbitrary existing text file, and KEY is a magic string found in this file. When enkf is running the magic string is replaced with parameter data when the ECLIPSE_FILE is written to the directory where the simulation is run from. Consider for example the follwing configuration:

```
TEMPLATE:/some/file   KEY:Magic123
```

The template file can look like this (only the Magic123 is special):

```
Header line1
Header line2
============
Magic123
============
Footer line1
Footer line2
```

When enkf is running the string Magic123 is replaced with parameter values, and the resulting file will look like this:

```
Header line1
Header line2
============
1.6723
5.9731
4.8881
.....
============
Footer line1
Footer line2
```

SURFACE

The SURFACE keyword can be used to work with surface from RMS in the irap format. The surface keyword is configured like this:

```
SURFACE TOP   OUTPUT_FILE:surf.irap   INIT_FILES:Surfaces/surf%d.irap
↪BASE_SURFACE:Surfaces/surf0.irap
```

The first argument, TOP in the example above, is the identifier you want to use for this surface in ert. The OUTPUT_FILE key is the name of surface file which ERT will generate for you, INIT_FILES points to a list of files which are used to initialize, and BASE_SURFACE must point to one existing surface file. When loading the surfaces ERT will check that all the headers are compatible. An example of a surface IRAP file is:

```
-996   511     50.000000     50.000000
444229.9688   457179.9688  6809537.0000  6835037.0000
260      -30.0000   444229.9688  6809537.0000
0     0     0     0     0     0     0
2735.7461    2734.8909    2736.9705    2737.4048    2736.2539    2737.0122
2740.2644    2738.4014    2735.3770    2735.7327    2733.4944    2731.6448
2731.5454    2731.4810    2730.4644    2730.5591    2729.8997    2726.2217
2721.0996    2716.5913    2711.4338    2707.7791    2705.4504    2701.9187
....
```

The surface data will typically be fed into other programs like Cohiba or RMS. The data can be updated using e.g. the Smoother.

Initializing from the FORWARD MODEL

All the parameter types like FIELD,GEN_KW,GEN_PARAM and SURFACE can be initialized from the forward model. To achieve this you just add the setting FORWARD_INIT:True to the configuration. When using forward init the initialization will work like this:

1. The explicit initialization from the case menu, or when you start a

simulation, will be ignored.

1. When the FORWARD_MODEL is complete ERT will try to initialize the node

based on files created by the forward model. If the init fails the job as a whole will fail.

1. If a node has been initialized, it will not be initialized again if you run

again. [Should be possible to force this ....]

When using FORWARD_INIT:True ERT will consider the INIT_FILES setting to find which file to initialize from. If the INIT_FILES setting contains a relative filename, it will be interpreted relativt to the runpath directory. In the example below we assume that RMS has created a file petro.grdecl which

contains both the PERMX and the PORO fields in grdecl format; we wish to initialize PERMX and PORO nodes from these files:

```
FIELD   PORO  PARAMETER    poro.grdecl     INIT_FILES:petro.grdecl ␣
↪FORWARD_INIT:True
FIELD   PERMX PARAMETER    permx.grdecl    INIT_FILES:petro.grdecl ␣
↪FORWARD_INIT:True
```

Observe that forward model has created the file petro.grdecl and the nodes PORO and PERMX create the ECLIPSE input files poro.grdecl and permx.grdecl, to ensure that ECLIPSE finds the input files poro.grdecl and permx.grdecl the forward model should contain a job which will copy/convert petro.grdecl -> (poro.grdecl,permx.grdecl), this job should not overwrite existing versions of permx.grdecl and poro.grdecl. This extra hoops is not strictly needed in all cases, but strongly recommended to ensure that you have control over which data is used, and that everything is consistent in the case where the forward model is run again.

SUMMARY

The SUMMARY keyword is used to add variables from the ECLIPSE summary file to the parametrization. The keyword expects a string, which should have the format VAR:WGRNAME. Here, VAR should be a quantity, such as WOPR, WGOR, RPR or GWCT. Moreover, WGRNAME should refer to a well, group or region. If it is a field property, such as FOPT, WGRNAME need not be set to FIELD.

Example:

```
-- Using the SUMMARY keyword to add diagnostic variables
SUMMARY WOPR:MY_WELL
SUMMARY RPR:8
SUMMARY F*          -- Use of wildcards requires that you have entered a REFCASE.
```

The SUMMARY keyword has limited support for '*' wildcards, if your key contains one or more '*' characters all matching variables from the refcase are selected. Observe that if your summary key contains wildcards you must supply a refcase with the REFCASE key - otherwise it will fail hard.

Note: Properties added using the SUMMARY keyword are only diagnostic. I.e., they have no effect on the sensitivity analysis or history match.

9.7 Keywords controlling the ES algorithm

See the sub keyword `OVERLAP_LIMIT` under the :code:'UPDATE_SETTINGS'keyword.

ENKF_BOOTSTRAP

Boolean specifying if we want to resample the Kalman gain matrix in the update step. The purpose is to avoid that the ensemble covariance collapses. When this keyword is true each ensemble member will be updated based on a Kalman gain matrix estimated from a resampling with replacement of the full ensemble.

In theory and in practice this has worked well when one uses a small number of ensemble members.

ENKF_CV_FOLDS

Integer specifying how many folds we should use in the Cross-Validation (CV) scheme. Possible choices are the integers between 2 and the ensemble size (2-fold CV and leave-one-out CV respectively). However, a robust choice for the number of CV-folds is 5 or 10 (depending on the ensemble size).

Example:

```
-- Setting the number of CV folds equal to 5
ENKF_CV_FOLDS 5
```

Requires that the ENKF_LOCAL_CV keyword is set to TRUE

ENKF_FORCE_NCOMP

Bool specifying if we want to force the subspace dimension we want to use in the EnKF updating scheme (SVD-based) to a specific integer. This is an alternative to selecting the dimension using ENKF_TRUNCATION or ENKF_LOCAL_CV.

Example:

```
-- Setting the the subspace dimension to 2
ENKF_FORCE_NCOMP     TRUE
ENKF_NCOMP              2
```

ENKF_LOCAL_CV

Boolean specifying if we want to select the subspace dimension in the SVD-based EnKF algorithm using Cross-Validation (CV) [1]. This is a more robust alternative to selecting the subspace dimension based on the estimated singular values (See ENKF_TRUNCATION), because the predictive power of the estimated Kalman gain matrix is taken into account.

Example:

```
-- Select the subspace dimension using Cross-Validation
ENKF_LOCAL_CV TRUE
```

ENKF_PEN_PRESS

Boolean specifying if we want to select the subspace dimension in the SVD-based EnKF algorithm using Cross-Validation (CV), and a penalised version of the predictive error sum of squares (PRESS) statistic [2]. This is recommended when overfitting is a severe problem (and when the number of ensemble members is small)

Example:

```
-- Select the subspace dimension using Cross-Validation
ENKF_LOCAL_CV TRUE

-- Using penalised PRESS statistic
ENKF_PEN_PRESS TRUE
```

ENKF_MODE

The ENKF_MODE keyword is used to select which EnKF algorithm to use. Use the value STANDARD for the original EnKF algorithm, or SQRT for the so-called square root scheme. The default value for ENKF_MODE is STANDARD.

Example A:

```
-- Using the square root update
ENKF_MODE SQRT
```

Example B:

```
-- Using the standard update
ENKF_MODE STANDARD
```

The ENKF_MODE keyword is optional.

ENKF_MERGE_OBSERVATIONS

If you use the ENKF_SCHED_FILE option to jump over several dates at a time you can choose whether you want to use all the observations in between, or just the final. If set to TRUE, all observations will be used. If set to FALSE, only the final observation is used. The default value for ENKF_MERGE_OBSERVATIONS is FALSE.

Example:

```
-- Merge observations
ENKF_MERGE_OBSERVATIONS TRUE
```

ENKF_NCOMP

Integer specifying the subspace dimension. Requires that ENKF_FORCE_NCOMP is TRUE.

ENKF_RERUN

This is a boolean switch - TRUE or FALSE. Should the simulation start from time zero after each update.

ENKF_SCALING

This is a boolean switch - TRUE (Default) or FALSE. If TRUE, we scale the data ensemble matrix to unit variance. This is generally recommended because the SVD-based EnKF algorithm is not scale invariant.

ENKF_TRUNCATION

Truncation factor for the SVD-based EnKF algorithm (see Evensen, 2007). In this algorithm, the forecasted data will be projected into a low dimensional subspace before assimilation. This can substantially improve on the results obtained with the EnKF, especially if the data ensemble matrix is highly collinear (Saetrom and Omre, 2010). The subspace dimension, p, is selected such that

```
\frac{\sum_{i=1}^{p} s_i^2}{\sum_{i=1}^r s_i^2} \geq \mathrm{ENKF\_TRUNCATION},
```

where si is the ith singular value of the centered data ensemble matrix and r is the rank of this matrix. This criterion is similar to the explained variance criterion used in Principal Component Analysis (see e.g. Mardia et al. 1979).

The default value of ENKF_TRUNCATION is 0.99. If ensemble collapse is a big problem, a smaller value should be used (e.g 0.90 or smaller). However, this does not guarantee that the problem of ensemble collapse will disappear. Note that setting the truncation factor to 1.00, will recover the Standard-EnKF algorithm if and only if the covariance matrix for the observation errors is proportional to the identity matrix.

STD_SCALE_CORRELATED_OBS

With this kewyord you can instruct ERT to use the simulated data to estimate the correlations in the observations, and then inflate the observation standard deviation as a way to estimate the real information content in the observations. The method is based on PCA, the scaling factor is calculated as:

```
\sqrt{\frac{N_{\sigma}}{N_{\mathrm{obs}}}}
```

where N_{sigma} is the number of singular components, at (fixed) truncation 0.95 and $N_{mathrm{obs}}$ is the number of observations. The STD_SCALE_CORRELATED_OBS keyword will flatten all your observations, including temporal and spatial correlations. For more fine grained control you can use the STD_CALE_CORRELATED_OBS workflow job, or even write your own plugins.

UPDATE_LOG_PATH

A summary of the data used for updates are stored in this directory.

References

- Evensen, G. (2007). “Data Assimilation, the Ensemble Kalman Filter”, Springer.
- Mardia, K. V., Kent, J. T. and Bibby, J. M. (1979). “Multivariate Analysis”, Academic Press.
- Saetrom, J. and Omre, H. (2010). “Ensemble Kalman filtering with shrinkage regression techniques”, Computational Geosciences (online first).

9.8 Analysis module

The final EnKF linear algebra is performed in an analysis module. The keywords to load, select and modify the analysis modules are documented here.

ANALYSIS_LOAD

The ANALYSIS_LOAD key is the main key to load an analysis module:

```
ANALYSIS_LOAD ANAME  analysis.so
```

The first argument ANAME is just an arbitrary unique name which you want to use to refer to the module later. The second argument is the name of the shared library file implementing the module, this can either be an absolute path as /path/to/my/module/ana.so or a relative file name as analysis.so. The module is loaded with dlopen() and the normal shared library search semantics applies.

ANALYSIS_SELECT

This command is used to select which analysis module to actually use in the updates:

```
ANALYSIS_SELECT ANAME
```

Here ANAME is the name you have assigned to the module when loading it with ANALYSIS_LOAD.

ANALYSIS_SET_VAR

The analysis modules can have internal state, like e.g. truncation cutoff values, these values can be manipulated from the config file using the ANALYSIS_SET_VAR keyword:

```
ANALYSIS_SET_VAR  ANAME  ENKF_TRUNCATION  0.97
```

To use this you must know which variables the module supports setting this way. If you try to set an unknown variable you will get an error message on stderr.

ANALYSIS_COPY

With the ANALYSIS_COPY keyword you can create a new instance of a module. This can be convenient if you want to run the same algorithm with the different settings:

```
ANALYSIS_LOAD   A1  analysis.so
ANALYISIS_COPY  A1  A2
```

We load a module analysis.so and assign the name A1; then we copy A1 -> A2. The module A1 and A2 are now 100% identical. We then set the truncation to two different values:

```
ANALYSIS_SET_VAR A1 ENKF_TRUNCATION 0.95
ANALYSIS_SET_VAR A2 ENKF_TRUNCATION 0.98
```

Developing analysis modules

In the analysis module the update equations are formulated based on familiar matrix expressions, and no knowledge of the innards of the ERT program are required. Some more details of how modules work can be found here modules.txt. In principle a module is 'just' a shared library following some conventions, and if you are sufficiently savy with gcc you can build them manually, but along with the ert installation you should have utility script ert_module which can be used to build a module; just write ert_module without any arguments to get a brief usage description.

9.9 Advanced optional keywords

The keywords in this section, controls advanced features of the enkf application. Insight in the internals of the enkf application and/or ECLIPSE may be required to fully understand their effect. Moreover, many of these keywords are defined in the site configuration, and thus optional to set for the user, but required when installing the enkf application at a new site.

ADD_FIXED_LENGTH_SCHEDULE_KW

Real low level fix for some SCHEDULE parsing problems.

DEFINE

With the DEFINE keyword you can define key-value pairs which will be substituted in the rest of the configuration file. The DEFINE keyword expects two arguments: A key and a value to replace for that key. Later instances of the key enclosed in '<' and '>' will be substituted with the value. The value can consist of several strings, in that case they will be joined by one single space.

Example:

```
-- Define ECLIPSE_PATH and ECLIPSE_BASE
DEFINE  ECLIPSE_PATH  /path/to/eclipse/run
DEFINE  ECLIPSE_BASE  STATF02
DEFINE  KEY           VALUE1       VALUE2 VALUE3            VALUE4

-- Set the GRID in terms of the ECLIPSE_PATH
-- and ECLIPSE_BASE keys.
GRID    <ECLIPSE_PATH>/<ECLIPSE_BASE>.EGRID
```

Observe that when you refer to the keys later in the config file they must be enclosed in '<' and '>'. Furthermore, a key-value pair must be defined in the config file before it can be used. The final key define above KEY, will be replaced with VALUE1 VALUE2 VALUE3 VALUE4 - i.e. the extra spaces will be discarded.

TIME_MAP

Normally the mapping between report steps and true dates is inferred by ERT indirectly by loading the ECLIPSE summary files. In cases where you do not have any ECLIPSE summary files you can use the TIME_MAP keyword to specify a file with dates which are used to establish this mapping:

Example:

```
-- Load a list of dates from external file: "time_map.txt"
TIME_MAP time_map.txt
```

The format of the TIME_MAP file should just be a list of dates formatted as dd/mm/yyyy. The example file below has four dates:

```
01/01/2000
01/07/2000
01/01/2001
01/07/2001
```

SCHEDULE_PREDICTION_FILE

This is the name of a schedule prediction file. It can contain %d to get different files for different members. Observe that the ECLIPSE datafile should include only one schedule file, even if you are doing predictions.

9.10 Keywords related to running the forward model

FORWARD_MODEL

The FORWARD_MODEL keyword is used to define how the simulations are executed. E.g., which version of ECLIPSE to use, which rel.perm script to run, which rock physics model to use etc. Jobs (i.e. programs and scripts) that are to be used in the FORWARD_MODEL keyword must be defined using the INSTALL_JOB keyword. A set of default jobs are available, and by default FORWARD_MODEL takes the value ECLIPSE100.

The FORWARD_MODEL keyword expects a series of keywords, each defined with INSTALL_JOB. The enkf will execute the jobs in sequentially in the order they are entered. Note that the ENKF_SCHED_FILE keyword can be used to change the FORWARD_MODEL for sub-sequences of the run.

Example A:

```
-- Suppose that "MY_RELPERM_SCRIPT" has been defined with
-- the INSTALL_JOB keyword. This FORWARD_MODEL will execute
-- "MY_RELPERM_SCRIPT" before ECLIPSE100.
FORWARD_MODEL MY_RELPERM_SCRIPT ECLIPSE100
```

Example B:

```
-- Suppose that "MY_RELPERM_SCRIPT" and "MY_ROCK_PHYSICS_MODEL"
-- has been defined with the INSTALL_JOB keyword.
-- This FORWARD_MODEL will execute "MY_RELPERM_SCRIPT", then
-- "ECLIPSE100" and in the end "MY_ROCK_PHYSICS_MODEL".
FORWARD_MODEL MY_RELPERM_SCRIPT ECLIPSE100 MY_ROCK_PHYSICS_MODEL
```

For advanced jobs you can pass string arguments to the job using a KEY=VALUE based approach, this is further described in: passing arguments. In available jobs in enkf you can see a list of the jobs which are available.

JOB_SCRIPT

Running the forward model from enkf is a multi-level process which can be summarized as follows:

1. A Python module called jobs.py is written and stored in the directory where

the forward simulation is run. The jobs.py module contains a list of job-elements, where each element is a Python representation of the code entered when installing the job.

1. The enkf application submits a Python script to the enkf queue system, this

script then loads the jobs.py module to find out which programs to run, and how to run them.

1. The job_script starts and monitors the individual jobs in the jobs.py

module.

The JOB_SCRIPT variable should point at the Python script which is managing the forward model. This should normally be set in the site wide configuration file.

QUEUE_SYSTEM

The keyword QUEUE_SYSTEM can be used to control where the simulation jobs are executed. It can take the values LSF, TORQUE, RSH and LOCAL.

The LSF option will submit jobs to the LSF cluster at your location, and is recommended whenever LSF is available.

The TORQUE option will submit jobs to the TORQUE a torque based system, using the commands qsub, qstat etc., if available.

If you do not have access to LSF or TORQUE you can submit to your local workstation using the LOCAL option and to homemade cluster of workstations using the RSH option. All of the queue systems can be further configured, see separate sections.

Example:

```
-- Tell ert to use the LSF cluster.
QUEUE_SYSTEM LSF
```

The QUEUE_SYSTEM keyword is optional, and usually defaults to LSF (this is site dependent).

9.11 Configuring LSF access

The LSF system is the most useful of the queue alternatives, and also the alternative with most options. The most important options are related to how ert should submit jobs to the LSF system. Essentially there are two methods ert can use when submitting jobs to the LSF system:

1. For workstations which have direct access to LSF ert can submit directly with no further configuration. This is preferred solution, but unfortunately not very common.
2. Alternatively ert can issue shell commands to bsub/bjobs/bkill to submit jobs. These shell commands can be issued on the current workstation, or alternatively on a remote workstation using ssh.

The main switch between alternatives 1 and 2 above is the LSF_SERVER option.

LSF_SERVER

By using the LSF_SERVER option you essentially tell ert two things about how jobs should be submitted to LSF:

1. You tell ert that jobs should be submitted using shell commands.
2. You tell ert which server should be used when submitting

So when your configuration file has the setting:

```
LSF_SERVER   be-grid01
```

ert will use ssh to submit your jobs using shell commands on the server be-grid01. For this to work you must have passwordless ssh to the server be-grid01. If you give the special server name LOCAL ert will submit using shell commands on the current workstation.

bsub/bjobs/bkill options

By default ert will use the shell commands bsub,bjobs and bkill to interact with the queue system, i.e. whatever binaries are first in your PATH will be used. For fine grained control of the shell based submission you can tell ert which programs to use:

```
QUEUE_OPTION   LSF  BJOBS_CMD  /path/to/my/bjobs
QUEUE_OPTION   LSF  BSUB_CMD   /path/to/my/bsub
```

Example 1

```
LSF_SERVER    be-grid01
QUEUE_OPTION  LSF     BJOBS_CMD   /path/to/my/bjobs
QUEUE_OPTION  LSF     BSUB_CMD    /path/to/my/bsub
```

In this example we tell ert to submit jobs from the workstation be-grid01 using custom binaries for bsub and bjobs.

Example 2

```
LSF_SERVER   LOCAL
```

In this example we will submit on the current workstation, without using ssh first, and we will use the default bsub and bjobs executables. The remaining LSF options apply irrespective of which method has been used to submit the jobs.

LSF_QUEUE

The name of the LSF queue you are running ECLIPSE simulations in.

9.12 Configuring TORQUE access

The TORQUE system is the only available system on some clusters. The most important options are related to how ert should submit jobs to the TORQUE system.

- Currently, the TORQUE option only works when the machine you are logged into have direct access to the queue system. ert then submit directly with no further configuration.

The most basic invocation is in other words:

```
QUEUE_SYSTEM TORQUE
```

qsub/qstat/qdel options

By default ert will use the shell commands qsub,qstat and qdel to interact with the queue system, i.e. whatever binaries are first in your PATH will be used. For fine grained control of the shell based submission you can tell ert which programs to use:

```
QUEUE_SYSTEM TORQUE
QUEUE_OPTION TORQUE QSUB_CMD /path/to/my/qsub
QUEUE_OPTION TORQUE QSTAT_CMD /path/to/my/qstat
QUEUE_OPTION TORQUE QDEL_CMD /path/to/my/qdel
```

In this example we tell ert to submit jobs using custom binaries for bsub and bjobs.

Name of queue

The name of the TORQUE queue you are running ECLIPSE simulations in.

```
QUEUE_OPTION TORQUE QUEUE name_of_queue
```

Name of cluster (label)

The name of the TORQUE cluster you are running ECLIPSE simulations in. This might be a label (serveral clusters), or a single one, as in this example baloo.

```
QUEUE_OPTION TORQUE CLUSTER_LABEL baloo
```

Max running jobs

The queue option MAX_RUNNING controls the maximum number of simultaneous jobs submitted to the queue when using (in this case) the TORQUE option in QUEUE_SYSTEM.

```
QUEUE_SYSTEM TORQUE
-- Submit no more than 30 simultaneous jobs
-- to the TORQUE cluster.
QUEUE_OPTION TORQUE MAX_RUNNING 30
```

Queue options controlling number of nodes and CPUs

When using TORQUE, you must specify how many nodes a single job is should to use, and how many CPUs per node. The default setup in ert will use one node and one CPU. These options are called NUM_NODES and NUM_CPUS_PER_NODE.

If the numbers specified is higher than supported by the cluster (i.e. use 32 CPUs, but no node has more than 16), the job will not start.

If you wish to increase these number, the program running (typically ECLIPSE) will usually also have to be told to correspondingly use more processing units (keyword PARALLEL)

```
QUEUE_SYSTEM TORQUE
-- Use more nodes and CPUs
-- in the TORQUE cluster per job submitted
-- This should (in theory) allow for 24 processing
-- units to be used by eg. ECLIPSE
QUEUE_OPTION TORQUE NUM_NODES 3
QUEUE_OPTION TORQUE NUM_CPUS_PER_NODE 8
```

Keep output from qsub

Sometimes the error messages from qsub can be useful, if something is seriously wrong with the environment or setup. To keep this output (stored in your home folder), use this:

```
QUEUE_OPTION TORQUE KEEP_QSUB_OUTPUT 1
```

** Slow submit to torque **

To be more gentle with the torqueue system you can instruct the driver to sleep for every submit request. The argument to the SUBMIT_SLEEP is the number of seconds to sleep for every submit, can be a fraction like 0.5.

```
QUEUE_OPTION TORQUE SUBMIT_SLEEP 0.25
```

** Torque debug log **

You can ask the torqueu driver to store a debug log of the jobs submitted, and the resulting job id. This is done with the queue option DEBUG_OUTPUT:

```
QUEUE_OPTION TORQUE DEBUG_OUTPUT torque_log.txt
```

9.13 Configuring the RSH queue

RSH_HOST

You can run the forward model in enkf on workstations using remote-shell commands. To use the RSH queue system you must first set a list of computers which enkf can use for running jobs:

```
RSH_HOST   computer1:2  computer2:2   large_computer:8
```

Here you tell enkf that you can run on three different computers: computer1, computer2 and large_computer. The two first computers can accept two jobs from enkf, and the last can take eight jobs. Observe the following when using RSH:

You must have passwordless login to the computers listed in RSH_HOST otherwise it will fail hard. enkf will not consider total load on the various computers; if have said it can take two jobs, it will get two jobs, irrespective of the existing load.

RSH_COMMAND

This is the name of the executable used to invoke remote shell operations. Will typically be either rsh or ssh. The command given to RSH_COMMAND must either be in PATH or an absolute path.

```
MAX_RUNNING_RSH
```

The keyword MAX_RUNNING_RSH controls the maximum number of simultaneous jobs running when using the RSH option in QUEUE_SYSTEM. It MAX_RUNNING_RSH exceeds the total capacity defined in RSH_HOST, it will automatically be truncated to that capacity.

Example:

```
-- No more than 10 simultaneous jobs
-- running via RSH.
MAX_RUNNING_RSH 10
```

9.14 Keywords related to plotting

PLOT_DRIVER

This is the name of the sub system used for creating plots. The default system is called 'PLPLOT' - all the other options regarding plotting are sub options which are only relevant when you are using PLPLOT. In addition to PLPLOT you can chose the value 'TEXT'; this will actually not produce any plots, just textfiles which can be used for plotting with your favorite plotting program. This is particularly relevant if you have some special requirements to the plots.

PLOT_ERRORBAR

Should errorbars on the observations be plotted?

PLOT_ERRORBAR_MAX

When plotting summary vectors for which observations have been 'installed' with the OBS_CONFIG keyword, ert will plot the observed values. If you have less than PLOT_ERRORBAR_MAX observations ert will use errorbars to show the observed values, otherwise it will use two dashed lines indicating +/- one standard deviation. This option is only meaningful when PLOT_PLOT_ERRORBAR is activated.

To ensure that you always get errorbars you can set PLOT_ERRORBAR_MAX to a very large value, on the other hand setting PLOT_ERRORBAR_MAX to 0 will ensure that ert always plots observation uncertainty using dashed

lines of +/- one standard deviation.

The setting here will also affect the output when you are using the TEXT driver to plot.

PLOT_HEIGHT

When the PLPLOT driver creates a plot file, it will have the height (in pixels) given by the PLOT_HEIGHT keyword. The default value for PLOT_HEIGHT is 768 pixels.

PLOT_REFCASE

Boolean variable which is TRUE if you want to add the refcases to the plots.

Example:

```
PLOT_REFCASE TRUE
```

REFCASE_LIST

Provide one or more Eclipse .DATA files for a refcase to be added in the plots. This refcase will be plotted in different colours. The summary files related to the refcase should be in the same folder as the refcase.

Example:

```
REFCASE_LIST /path/to/refcase1/file1.DATA /path/to/refcase2/file2.DATA
```

PLOT_SETTINGS

The `PLOT_SETTINGS` keyword is a "master keyword" which can be used to configure some aspects of the plotting. These settings will affect the default behaviour when you create a new plot, you can still changes these settings interactively.

When using the `PLOT_SETTINGS` keyword you supply a secondary keyword and a values as the tow arguments:

```
PLOT_SETTINGS SHOW_REFCASE False
```

Will make sure that your plots are created without the refcase plotted as default. The available secondary keys are:

SHOW_REFCASE : Default True SHOW_HISTORY : Default True

RFT_CONFIG

RFT_CONFIGS argument is a file with the name of the rfts followed by date (day month year) Ex.

```
RFT_CONFIG  ../models/wells/rft/WELLNAME_AND_RFT_TIME.txt
```

Where the contents of the file is something like

```
be-linapp16(inmyr) -/models/wells/rft 34> more WELLNAME_AND_RFT_TIME.txt
A-1HP  06 05 1993
A-9HW  31 07 1993
C-1HP  11 12 2007
C-5HP  21 12 1999
C-6HR  09 11 1999
D-4HP  10 07 2003
K-3HW  09 02 2003
K-6HW  08 11 2002
K-7HW  21 04 2005
D-6HP  22 04 2006
```

RFTPATH

RFTPATHs argument is the path to where the rft-files are located

```
RFTPATH  ../models/wells/rft/
```

With the keyword HOOK_WORKFLOW you can configure workflow 'hooks'; meaning workflows which will be run automatically at certain points during ERTs execution. Currently there are four points in ERTs flow of execution where you can hook in a workflow, before the simulations start, PRE_SIMULATION; after all the simulations have completed POST_SIMULATION; before the update step, PRE_UPDATE and after the update step, POST_UPDATE. The POST_SIMULATION hook is typically used to trigger QC workflows:

```
HOOK_WORKFLOW initWFLOW         PRE_SIMULATION
HOOK_WORKFLOW preUpdateWFLOW    PRE_UPDATE
HOOK_WORKFLOW postUpdateWFLOW   POST_UPDATE
HOOK_WORKFLOW QC_WFLOW1         POST_SIMULATION
HOOK_WORKFLOW QC_WFLOW2         POST_SIMULATION
```

In this example the workflow initWFLOW will run after all the simulation directories have been created, just before the forward model is submitted to the queue. The workflow preUpdateWFLOW will be run before the update step and postUpdateWFLOW will be run after the update step. When all the simulations are complete the two workflows QC_WFLOW1 and QC_WFLOW2 will be run.

Observe that the workflows being 'hooked in' with the HOOK_WORKFLOW must be loaded with the LOAD_WORKFLOW keyword.

Currently, PRE_UPDATE and POST_UPDATE are only available from python.

9.15 Manipulating the Unix environment

The two keywords SETENV and UPDATE_PATH can be used to manipulate the Unix environment of the ERT process, tha manipulations only apply to the running ERT instance, and are not applied to the shell.

SETENV

You can use the SETENV keyword to alter the unix environment enkf is running in. This is probably most relevant for setting up the environment for the external jobs invoked by enkf.

Example:

```
-- Setting up LSF
SETENV  LSF_BINDIR      /prog/LSF/7.0/linux2.6-glibc2.3-x86_64/bin
SETENV  LSF_LIBDIR      /prog/LSF/7.0/linux2.6-glibc2.3-x86_64/lib
SETENV  LSF_UIDDIR      /prog/LSF/7.0/linux2.6-glibc2.3-x86_64/lib/uid
SETENV  LSF_SERVERDIR   /prog/LSF/7.0/linux2.6-glibc2.3-x86_64/etc
SETENV  LSF_ENVDIR      /prog/LSF/conf
```

Observe that the SETENV command is not as powerful as the corresponding shell utility. In particular you can not use $VAR to refer to the existing value of an environment variable. To add elements to the PATH variable it is easier to use the UPDATE_PATH keyword.

UPDATE_PATH

The UPDATE_PATH keyword will prepend a new element to an existing PATH variable. I.e. the config

```
UPDATE_PATH   PATH  /some/funky/path/bin
```

will be equivalent to the shell command:

```
setenv PATH /some/funky/path/bin:$PATH
```

The whole thing is just a workaround because we can not use $PATH.

The UPDATE_SETTINGS keyword is a *super-keyword* which can be used to control parameters which apply to the Ensemble Smoother update algorithm. The :code:‘UPDATE_SETTINGS‘currently supports the two subkeywords:

OVERLAP_LIMIT Scaling factor used when detecting outliers. Increasing this factor means that more observations will potentially be included in the assimilation. The default value is 3.00..

Including outliers in the Smoother algorithm can dramatically increase the coupling between the ensemble members. It is therefore important to filter out these outlier data prior to data assimilation. An observation, textstyle d^o_i, will be classified as an outlier if

```
|d^o_i - \bar{d}_i| > \mathrm{ENKF\_ALPHA} \left(s_{d_i} + \sigma_{d^o_i}
↪\right),
```

where textstyleboldsymbol{d}^o is the vector of observed data, textstyleboldsymbol{bar{d}} is the average of the forcasted data ensemble, textstyleboldsymbol{s_{d}} is the vector of estimated standard deviations for the forcasted data ensemble, and textstyleboldsymbol{s_{d}^o} is the vector standard deviations for the observation error (specified a priori).

STD_CUTOFF If the ensemble variation for one particular measurment is below this limit the observation will be deactivated. he default value for this cutoff is 1e-6.

Observe that for the updates many settings should be applied on the analysis module in question.

UMASK

The *umask* is a concept used by Linux to controll the permissions on newly created files. By default the files created by ert will have the default permissions of your account, but by using the keyword *UMASK* you can alter the permissions of files created by ert.

To determine the initial permissions on newly created files start with the initial permissions *-rw-rw-rw-* (octal 0666) for files and *-rwxrwxrwx* (octal 0777) for directories, and then *~subtract* the current umask setting. So if you wish the newly created files to have permissions *-rw-r—* you need to subtract write permissions for group and read and write permissions for others - corresponding to *umask 0026*.

```
UMASK 0022
```

We remove write permissions from group and others, implying that everyone can read the files and directories created by ert, but only the owner can write to them. Also everyone can execute the directories (i.e. list the content).

```
UMASK 0
```

No permissions are removed, i.e. everyone can do everything with the files and directories created by ert.

The umask setting in ert is passed on to the forward model, and should apply to the files/directories created by the forward model also. However - the executables in the forward model can in principle set it's own umask setting or alter permissions in another way - so there is no guarantee that the umask setting will apply to all files created by the forward model.

The octal permissions are based on three octal numbers for owner, group and others, where each value is based on adding the constants:

1: Execute permission 2: Write permission 4: Read permission

So an octal permission of 0754 means:

- Owner(7) can execute(1), write(2) and read(4).
- Group(5) can execute(1) and read(4).
- Others(2) can read(4)

CHAPTER

TEN

RELEASE NOTES FOR ERT

10.1 Version 2.3

10.1.1 2.3 ert application

PR: 67 - 162

10.1.2 2.3 libres

PR: 105 - 411

10.1.3 2.3 libecl

PR: 170 - 506

10.2 Version 2.2

10.2.1 2.2: ert application

Version 2.2.1 September 2017 PR: 1 - 66 Cherry-picked: 70

Misc:

- Using res_config changes from libres [16]
- files moved from libecl to libres: [51]
- replaced ert.enkf with res.enkf [56]
- Created ErtVersion: [61, 66].
- Using res_config: [62]
- Removed dead workflow files: [64]

Build and testing:

- Cleanup after repo split [1, 2, 3 , 4, 5 , 6]
- Added test_install functionality [7]
- Added travis build script for libecl+libres+ert: [15, 17, 18, 19, 21, 26, 27, 39, 52-55, 63]

- MacOS build error: [28].
- Created simple gui_test bin/gui_test [32]
- cmake - create symlink: [41, 42, 43]
- Initial Python3 testing [58, 60].

Queue and running:

- Added base run model - gui model updates: [20].
- Added single simulation pretest running [33, 36, 50, 67].
- Add run_id to simulation batches.

10.2.2 2.2: libres

Version 2.2.9 September 2017 PR: 1 - 104 Cherry-picks: [106, 108, 110, 118, 121, 122, 123, 127]

Misc:

- implement legacy from ert.xxx [1, 20, 21, 22]
- Setting up libres_util and moving ert_log there [13, 44, 48].
- Added subst_list + block_fs functionality to res_util - moved from libecl [27, 68, 74].
- Do not generate parameters.txt if no GEN_KW is specified.[89]
- Started using RES_VERSION [91].
- CONFIG_PATH subtitution settings - bug fixed[43, 96].
- Will load summary if GEN_DATA is present [123, 127]

Build and test fixes:

- Simple functionality to do post-install testing[3]
- Use libecl as cmake target[6,‘15 <https://github.com/Statoil/res/pull/15/>‘_]
- removed stale binaries [7, 9]
- travis will build all repositories [23].
- Travis + OSX [69, 72]
- Remove statoil specific settings from build sytem [38].
- Travis split for parallell builds [79].

Config refactor:

> In this release cycle there have been large amount of changes to the code configuring the ERT state; the purpose of these changes has been to prepare for further development with Everest. The main net change is that a new configuration object - res_config has been created ,which holds all the configuration subobjects:
>
> [10, 14, 35, 39, 45, 52, 54, 58-62, 66, 75]

Queue layer: ‘

- Improved logging [17, 37].

- Funcionality to create a queue_config object copy [36].

As part of this development cycle the job_dispatch script has been included in the libres distribution. There are many PR's related to this script:

[28, 40, 41, 51, 53, 63, 64, 83, 84, 85, 93, 94, 95, 97-99, 101, 103, 108, 110]

- Create a common run_id for one batch of simulations, and generally treat one batch of simulations as one unit, in a better way than previously: [42, 67]
- Added PPU (Paay Per Use) code to LSF driver [71].
- Workflow job PRE_SIMULATION_COPY [73, 88].
- Allow to unset QUEUE_OPTION [87].
- Jobs failing due to dead nodes are restarted [100].

Documentation:

- Formatting bugs: [49, 50]
- Removed doxygen + build rst [29]

10.2.3 2.2: libecl

Version 2.2.0 September 2017 PR: 1 - 169 Open PR: 108, 145

Grid:

- Extracted implementation ecl_nnc_geometry [1, 66, 75, 78, 80, 109].
- Fix bug in cell_contains for mirrored grid [51, 53].
- Extract subgrid from grid [56].
- Expose mapaxes [63, 64].
- grid.get_lgr - numbered lookup [83]
- Added NUMRES values to EGRID header [125].

Build & testing:

- Removed warnings - added pylint [4, 5, 6, 10, 11, 12]
- Accept any Python 2.7.x version [17, 18]
- Remove ERT testing & building [3, 19]
- Changes to Python/cmake machinery [25, 30, 31, 32, 49, 52, 62].
- Added cmake config file [33, 44, 45, 47].
- Only *one* library [54, 55, 58,

69, 73, 77, 91, 133] - Removed stale binaries [59]. - Require cmake >= 2.8.12 [67]. - Fix build on OSX [87, 88, 95, 103]. - Fix broken behavior with internal test data [97]. - Travis - compile with -Werror [122, 123, 127, 130] - Started to support Python3 syntax [150, 161] - Add support for paralell builds on Travis [149]

libecl now fully supports OSX. On Travis it is compiled with -Werror=all which should protect against future warnings.

C++:

- Removed use of deignated initializers [7].
- Memory leak in EclFilename.cpp [14].

- Guarantee C linkage for ecl_data_type [65].
- New smspec overload [89].
- Use -std=c++0x if -std=c++11 is unavailable [118]
- Make all of (previous(libutil compile with C++ [162]

Well:

- Get well rates from restart files [8,‘20 <https://github.com/Statoil/res/pull/20/>‘_].
- Test if file exists before load [111].
- Fix some warnings [169]

Core:

- Support for variable length strings in binary eclipse files [13, 146].
- Renamed root package ert -> ecl [21]
- Load INTERSECT summary files with NAMES instead WGNAMES [34 - 39].
- Possible memory leak: [61]
- Refactored binary time search in __get_index_from_sim_time() [113]
- Possible to mark fortio writer as “failed” - will unlink on close [119].
- Allow keywords of more than 8 characters [120, 124].
- ecl_sum writer: Should write RESTART keyword [129, 131]
- Made EclVersion class [160]
- Functionality to dump an index file for binary files: [155, 159, 163, 166, 167]

Misc:

- Added legacy pacakge ert/ [48, 99]
- Improved logging - adding enums for og levels [90, 140, 141]
- Refactored to use snake_case instead of CamelCase [144, 145]

Version 2.1.0 February 2017 PR: 1150 - 1415 Open PR: 1352, 1358, 1362

Queue system/workflows:

- Functionality to blacklist nodes from LSF [1240, 1256, 1258, 1274, 1412, 1415].
- Use bhist command to check lsf job status if job has expired from bjobs [1301].
- Debug output from torque goes to stdout [1151].
- Torque driver will not abort if qstat returns invalid status [1411].
- Simulation status USER_EXIT - count as failed [1166].
- Added Enum identifier ‘JOB_QUEUE_DO_KILL_NODE_FAILURE [1268].
- Have deprecated the ability to set queue options directly on the drivers [1288].
- Added system for version specific loading for workflow job model description files [1177].
- Job loader should not try to load from directory [1187].
- Refactoring of max runtime - initiated from WPRO [1237].

- Determine which nodes are running a job [1251].

Build updates:

- Check if python modules are present in the required version [1150].
- Do not build ERT_GUI if PyQt4 is not found [1150, 1230].
- Do not fail build numpy is not found [1153].
- Allow for user provided CMAKE_C_FLAGS on linux [1300].
- Require exactly version 2.7 of Python [1307].
- Travis testing improvements [1363].
- Removed devel/ directory from source [1196].
- Setting correct working directory, and build target dependency for sphinx-apidoc / html generation [1385].

Eclipse library:

- C++ move constructor and operator= for smspec_node [1155, 1200].
- fortio_ftruncate( ) [1161].
- INIT writer will write keywords DEPTH, DX, DY, DZ [1164, 1172, 1311, 1388].
- Grid writer will take unit system enum argument [1164].
- New function ecl_kw_first_different( ) [1165].
- Completion variables can be treated as totals [1167].
- Fixed bug in ecl_kw_compare_numeric( ) [1168].
- Refactored / inlined volume calculations in ecl_grid [1173, 1184].
- Made function ecl_kw_size_and_type_equal() public [1192].
- Fixed bug in ecl_grid_cell_contains() [1402, 1404, 1195, 1419].
- OOM bug in ecl_kw_grdecl loader for large files [1207].
- Cache cell volumes in ecl_grid [1228].
- Geertsma / gravity functionality [1227, 1284, 1289, 1292, 1364, 1408].
- Summary + restart - will allow some keyword differences [1296].
- Implemented ecl_rst_file_open_write_seek( ) [1236].
- Optionally apply mapaxes [1242, 1281].
- Expose and use ecl_file_view datastructere - stop using ‘blocks’ in ecl_file objects [1250].
- ecl_sum will internalize Aquifer variables [1275].
- Make sure region properties RxxT are marked as total + depreecated some properties [1285].
- ecl_kw_resize() + C++ operator[] [1316]
- Added small C++ utility to create eclipse filenames[1396].
- Make sure restart and INIT files are written with correct unit ID [1399, 1407].
- Skip keyword data type: ‘C010’ without failing [1406, 1410].
- Adding parsing of the last (optional) config token for the SLAVES kwd [1409].
- Add nnc index to the information exported by ecl_nnc_export() [1204].

- Marked solvent related total keywords ?NIT and ?NPT.* as totals [1241].
- Top active cell in grid [1322].
- Added absolute epsilon to ecl_kw comparsion [1345,1351].

Smoother, updates and 'running':

- Fixed bug with local updates of GEN_DATA [1291].
- Changed default MDA weights and removed file input [1156, 1190, 1191].
- Bug in handling of failed realisations [1163].
- Fix bug missing assignment of analysis module in ES-MDA [1179].
- OpenMP implementation of fwd_step [1185, 1324,1342].
- Removes the ability to update dynamic variables [1189].
- Allow max CV FOLD to be the number of ensembles [1205, 1208].
- Fix for min_realizations logic [1206].
- Can assign a specific analyis module for one local update [1224].
- Handle updates when some summary relaisations are "too short" [1400, 1405].
- Extending hook manager to support PRE_UPDATE and POST_UPDATE hooks [1340,1360].
- RML logging is on by default [1318].
- Changed default ENKF_ALPHA value to 3.0 [??]
- Implemented subsspacce inversion algorithm [1334, 1344].

libgeometry:

- Added function to create new geo_surface (i.e. IRAP) object [1308].
- Get (x, y) pos from surface [1395].

Python code:

- cwrap has been moved out to stand-alone module, out of ert package [1159, 1320, 1325, 1326, 1328, 1332, 1338, 1341, 1343, 1347, 1350, 1353]
- Simplified loading of shared libraries [1234].
- Python3 preparations [1231, 1347].
- Added __repr__ methods: [1266, 1327, 1329, 1331, 1348, 1373, 1375, 1377, 1384, 1387].
- Implement __getitem__( ) for gen_data [1331].
- Removed cstring_obj Python class [1387].
- EclKW.numpy_array returns shared buffer ndarray [1180].
- Minor bug in ecl_kw.py [1171].
- Added EclKW.numpyView( ) and EclKW.numpyCopy( ) [1188].
- Bug in EclKW slice access [1203].
- Expose active_list better in Python [1392].
- @TYPE@_vector suppports negative indices in __getitem__ and __setitem__; added repr method [1378].
- added root() methdo ert/__init__.py [1293].

GUI / Configuration / Documentation

- Bug when viewing plots while simulating [1157.]
- Bug when plotting short vectors [1303].
- Completely refactored the ERT Gui event system [1158, 1162].
- Marked keywords INIT_SECTION and SCHEDULE_FILE as deprecated [1181].
- Removed outdated keywords from documentation [1390].
- Documented UMASK keyword [1186].
- ConfigParser: Can turn off validation + warnings [1233, 1249, 1287].
- Make ies advanced option only [1401].
- Removing MAX_RUNNING_LOCAL and MAX_RUNNING_LSF from user doc [1398].
- Apply plot style to other plots [1397].
- Fig bug in initialization when out of range [1394].
- Added new object for generic config settings [1391].
- Changes to plot settings [11359,376,1380,1382,1386].
- Fix bug in load case manually [1368].
- Documentation of plugins [1194].
- Changed all time handling to UTC. This will affect loading old cases [1229, 1257].
- Removed keyword QC_PATH + updated GRID [1263].
- Making sure the ertshell is creating the run path [1280].
- Create Doxygen [1277,1278,1294,1309,1317].
- Ability to run analysis from GUI [1314].
- Improved documentation of priors [1319].
- Bug in config parsing with relative paths [1333].
- Field documentation updates [1337].

libwecl_well:

- Internalize rates for wells and connections in the well library [1403].
- New function well_ts_get_name() [1393].

libutil:

- Functions for parsing and outputting dates in ISO format[1248].
- stringlist_join - like Python str.join [1243].
- bug in matrix_dgemm [1286].
- Resurrected block_fs utilities from the past [1297].
- Slicing for runpath_list [1356].

CHAPTER

ELEVEN

INDICES AND TABLES

- genindex
- modindex
- search