VARUS

Generated by Doxygen 1.8.6

Tue Mar 14 2017 15:16:28

Contents

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

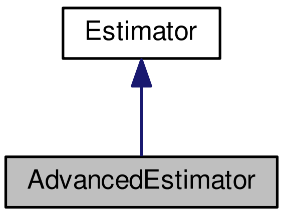
Chapter 4

Class Documentation

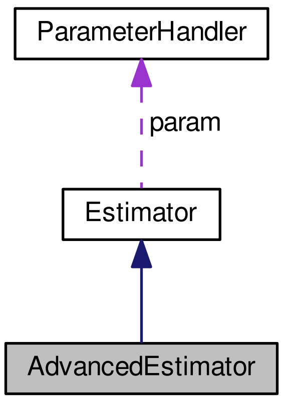
4.1 AdvancedEstimator Class Reference

```
#include <AdvancedEstimator.h>
```

Inheritance diagram for AdvancedEstimator:



Collaboration diagram for AdvancedEstimator:



Public Member Functions

- AdvancedEstimator (ParameterHandler ∗p)
- virtual ∼AdvancedEstimator ()
- void estimateP (std::vector< Run ∗ > &runs, unsigned int iterationNumber)

Public Attributes

- UDmap p_total

  The advanced estimator makes use of a pseudocount and a special pseudocount being distributed like a mixture of all runs combined.

Additional Inherited Members

4.1.1 Constructor & Destructor Documentation

4.1.1.1 AdvancedEstimator::AdvancedEstimator (ParameterHandler ∗ *p*)

4.1.1.2 AdvancedEstimator::∼AdvancedEstimator ( ) [virtual]

4.1.2 Member Function Documentation

4.1.2.1 void AdvancedEstimator::estimateP (std::vector< Run ∗ > & *runs*, unsigned int *iterationNumber*) [virtual]

Implements Estimator.

4.1.3 Member Data Documentation

4.1.3.1 UDmap AdvancedEstimator::p_total

The advanced estimator makes use of a pseudocount and a special pseudocount being distributed like a mixture of all runs combined.

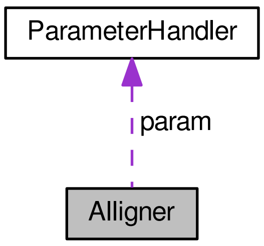
The documentation for this class was generated from the following files:

- /home/willy/workspace/VARUS/Implementation/AdvancedEstimator.h
- /home/willy/workspace/VARUS/Implementation/AdvancedEstimator.cpp

4.2 Alligner Class Reference

`#include <Alligner.h>`

Collaboration diagram for Alligner:



Public Member Functions

- Alligner (ParameterHandler ∗p)
- virtual ∼Alligner ()
- std::string shellCommand (Run ∗r)
- void mapReads (Run ∗r)
- void getAllignedReads (std::unordered_map< std::string, RNAread > &reads, Run ∗r)
- void updateObservations (Run ∗r, UUmap &totalObservations, std::unordered_map< std::string, RNAread > &reads, ChromosomeInitializer ∗c)
- void update (Run ∗r, UUmap &totalObservations, ChromosomeInitializer ∗c)

Public Attributes

- ParameterHandler ∗ param

  This class controlls the alignment-steps for the runs.

4.2.1 Constructor & Destructor Documentation

4.2.1.1 Alligner::Alligner (**ParameterHandler** ∗ *p*)

4.2.1.2 Alligner::∼Alligner ( ) `[virtual]`

4.2.2 Member Function Documentation

4.2.2.1 void Alligner::getAllignedReads (std::unordered_map< std::string, **RNAread** > & *reads,* **Run** ∗ *r*)

opens the SAM-file and retrieves how many times each read mapped to a transcript-block.

4.2.2.2 void Alligner::mapReads (**Run** ∗ *r*)

Calls STAR through the shell. The reads of the last batch downloaded from the passed run are then mapped to the transcript-units according to STAR.

4.2.2.3 std::string Alligner::shellCommand (**Run** ∗ *r*)

Constructs the shell command that calls STAR.

4.2.2.4 void Alligner::update (**Run** ∗ *r*, **UUmap** & *totalObservations*, **ChromosomeInitializer** ∗ *c*)

4.2.2.5 void Alligner::updateObservations (**Run** ∗ *r*, **UUmap** & *totalObservations*, std::unordered_map< std::string, **RNAread** > & *reads*, **ChromosomeInitializer** ∗ *c*)

4.2.3 Member Data Documentation

4.2.3.1 **ParameterHandler**∗ Alligner::param

This class controlls the alignment-steps for the runs.

The allignemnt is done with STAR.

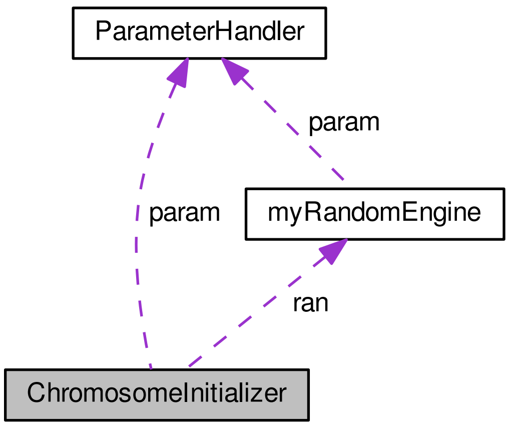
The documentation for this class was generated from the following files:

- /home/willy/workspace/VARUS/Implementation/Alligner.h
- /home/willy/workspace/VARUS/Implementation/Alligner.cpp

4.3 ChromosomeInitializer Class Reference

```
#include <ChromosomeInitializer.h>
```

Collaboration diagram for ChromosomeInitializer:



Public Member Functions

- ChromosomeInitializer (ParameterHandler ∗p)
- ∼ChromosomeInitializer ()
- void readInputRuns ()
- void getChromosomeLengths ()
- void create_transcript_units (UUmap &totalObservations)
- void initializeRuns (std::vector< Run ∗ > &runs)
- void initializeSigma (Run ∗r)

Public Attributes

- ParameterHandler ∗ param

    This class initializes the runs.
- myRandomEngine ∗ ran
- SUmap chromosomLengths
- SUmap transcriptUnitsByNames
- USmap transcriptUnits
- SUmap translate2int
- USmap translate2str
- std::vector< U32 > order
- std::vector< std::string > InputRuns

4.3.1 Constructor & Destructor Documentation

4.3.1.1 ChromosomeInitializer::ChromosomeInitializer (**ParameterHandler** ∗ *p*)

4.3.1.2 ChromosomeInitializer::∼ChromosomeInitializer ( )

4.3.2 Member Function Documentation

4.3.2.1 void ChromosomeInitializer::create_transcript_units (**UUmap** & *totalObservations*)

divides each chromosom in blocks of length blocksize names are given as x:1,...x:(x.size/blocksize) where x is a specific chromosom

4.3.2.2 void ChromosomeInitializer::getChromosomeLengths ( )

Reads in the length of the chromosomes in a file called "chrNameLength.txt".

4.3.2.3 void ChromosomeInitializer::initializeRuns (std::vector< **Run** ∗ > & *runs*)

Reads in information about the runs such as numOfSpots, accessionId from the Runlist.txt.

4.3.2.4 void ChromosomeInitializer::initializeSigma (**Run** ∗ *r*)

The sigma-vector contains the order in which the batches should be downloaded. The order is randomized so that potential biases in the order of the reads in the run are minimalized. The second argument is the seed used for the suffler. If set to a positive value, the seed is set, else the seed is derived randomly according to the current time.

4.3.2.5 void ChromosomeInitializer::readInputRuns ( )

Reads in the accession-ids of the Runs to download from.

reads from "Runlist.txt"

4.3.3 Member Data Documentation

4.3.3.1 SUmap ChromosomeInitializer::chromosomLengths

4.3.3.2 std::vector<std::string> ChromosomeInitializer::InputRuns

4.3.3.3 std::vector<U32> ChromosomeInitializer::order

4.3.3.4 ParameterHandler∗ ChromosomeInitializer::param

This class initializes the runs.

- Sigma order is set with this class. That is the order in which the batches should be downloaded.

- A translation-convention from UUmap to SUmap is created. UUmaps are used most of the time in this implementation, because they use less memory.

4.3.3.5 myRandomEngine∗ ChromosomeInitializer::ran

4.3.3.6 USmap ChromosomeInitializer::transcriptUnits

4.3.3.7 SUmap ChromosomeInitializer::transcriptUnitsByNames

4.3.3.8 SUmap ChromosomeInitializer::translate2int

4.3.3.9 USmap ChromosomeInitializer::translate2str

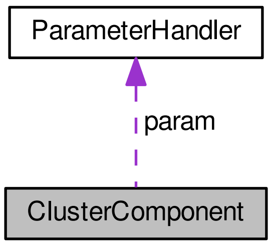
The documentation for this class was generated from the following files:

- /home/willy/workspace/VARUS/Implementation/ChromosomeInitializer.h
- /home/willy/workspace/VARUS/Implementation/ChromosomeInitializer.cpp

4.4 ClusterComponent Class Reference

```
#include <ClusterComponent.h>
```

Collaboration diagram for ClusterComponent:



Public Member Functions

- void setSeed (UDmap &start)
- void updateCenter ()
- void releaseRuns ()
- void calculateAlphaSum ()
- void calculateP ()
- double calculateVariance ()
- ClusterComponent (ParameterHandler ∗p)
- virtual ∼ClusterComponent ()

Public Attributes

- ParameterHandler ∗ param

  This Class represents a component of the cluster used in the clusterEstimator.
- UDmap alpha
- double alphaSum
- std::vector< Run ∗ > runs

4.4.1 Constructor & Destructor Documentation

4.4.1.1 ClusterComponent::ClusterComponent (**ParameterHandler** ∗ *p*)

4.4.1.2 ClusterComponent::∼ClusterComponent ( ) [virtual]

4.4.2 Member Function Documentation

4.4.2.1 void ClusterComponent::calculateAlphaSum ( )

Calculates the 'concentration-parameter'.

A higher variance in the frequencies of the runs leads to a smaller . This makes the estimation for runs with little reads more precisse. If the runs in the component only have very little variance, it is a good guess to assume that the reads in the runs with only little reads follow a similar distribution as the reads in the runs with more observations.

4.4.2.2 void ClusterComponent::calculateP ( )

The estimation p is calculated for all runs assigned to this component in the clustering-step.

The estimation is the same as in the advanced estimator, only for a single component.

4.4.2.3 double ClusterComponent::calculateVariance ( )

Calculates the variance in the observations of the runs assigned to this component.

The euclidean norm is used to get a scalar out of the vector.

4.4.2.4 void ClusterComponent::releaseRuns ( )

The runs clustered into this component are released again.

In the next step of the clustering the runs can then be assigned to a component again.

4.4.2.5 void ClusterComponent::setSeed (**UDmap** & *start*)

Sets the starting point for this cluster.

For each run the distance is calculated to this starting point in order to assign it to the run that is closest situated.

4.4.2.6 void ClusterComponent::updateCenter ( )

In each step of the k_means algorithm the center needs to be updated.

This is done by averaging over the runs assigned to this component.

4.4.3 Member Data Documentation

4.4.3.1 **UDmap** ClusterComponent::alpha

4.4.3.2 double ClusterComponent::alphaSum

4.4.3.3 **ParameterHandler**∗ ClusterComponent::param

This Class represents a component of the cluster used in the clusterEstimator.

Each cluster is assigned a part of the runs during clustering. Afterwards the runs are estimated based on the runs in the same cluster with them.

4.4.3.4 std::vector<**Run**∗> ClusterComponent::runs

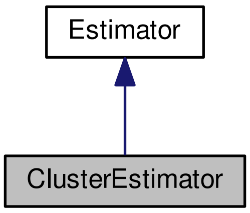
The documentation for this class was generated from the following files:

- /home/willy/workspace/VARUS/Implementation/ClusterComponent.h
- /home/willy/workspace/VARUS/Implementation/ClusterComponent.cpp

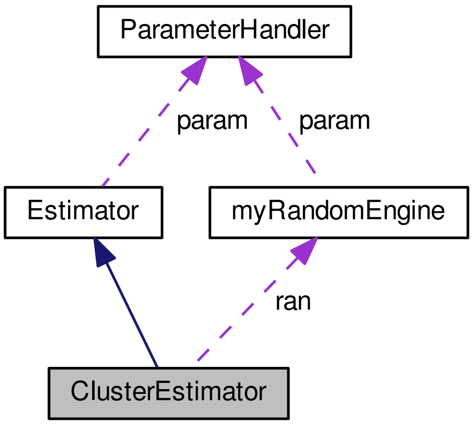
4.5 ClusterEstimator Class Reference

```
#include <ClusterEstimator.h>
```

Inheritance diagram for ClusterEstimator:



Collaboration diagram for ClusterEstimator:



Public Member Functions

- ClusterEstimator (ParameterHandler ∗p)
- virtual ∼ClusterEstimator ()
- void calculatePnoObs (std::vector< Run ∗ > &runs)
- void calculateRunQs (std::vector< Run ∗ > &runs)
- void initializeClusters (std::vector< Run ∗ > &runs)
- void kMeans (std::vector< Run ∗ > &runs)
- void estimateP (std::vector< Run ∗ > &runs, unsigned int iterationNumber)
- double distance (Run ∗r, ClusterComponent &c)
- void exportClusters (unsigned int iterationNumber)

Public Attributes

- std::vector< ClusterComponent > components

Calculates the estimation for the runs by first clustering the runs based on the similarity of the observations.
- myRandomEngine ∗ ran

Additional Inherited Members

4.5.1 Constructor & Destructor Documentation

4.5.1.1 ClusterEstimator::ClusterEstimator (**ParameterHandler** ∗ *p*)

p->components ClusterComponents are created.

4.5.1.2 ClusterEstimator::∼ClusterEstimator ( ) [virtual]

4.5.2 Member Function Documentation

4.5.2.1 void ClusterEstimator::calculatePnoObs (std::vector< **Run** ∗ > & *runs*)

Calculates the estimation for runs with no observations.

The estimation is done by using the observations from all runs with observations.

4.5.2.2 void ClusterEstimator::calculateRunQs (std::vector< **Run** ∗ > & *runs*)

4.5.2.3 double ClusterEstimator::distance (**Run** ∗ *r*, **ClusterComponent** & *c*)

4.5.2.4 void ClusterEstimator::estimateP (std::vector< **Run** ∗ > & *runs*, unsigned int *iterationNumber*) [virtual]

Calculates the estimation for the runs by first clustering the runs based on the similarity of the observations.

Implements Estimator.

4.5.2.5 void ClusterEstimator::exportClusters (unsigned int *iterationNumber*)

Exports the cluster into a file.

After a number indicating the cluster-number runs are written. All runs written below a number are assigned to the same cluster.

4.5.2.6 void ClusterEstimator::initializeClusters (std::vector< **Run** ∗ > & *runs*)

4.5.2.7 void ClusterEstimator::kMeans (std::vector< **Run** ∗ > & *runs*)

kMeans is a clustering algorithm.

Each point is a frequency of observations. Among all frequencies the best cluster given n clusterpoints is searched.

4.5.3 Member Data Documentation

4.5.3.1 std::vector<**ClusterComponent**> ClusterEstimator::components

Calculates the estimation for the runs by first clustering the runs based on the similarity of the observations.

This estimator is inspired by the Dirichlet-Mixture and aims to yield reasonable results while still having affordable computation-times.

4.5.3.2 myRandomEngine∗ ClusterEstimator::ran

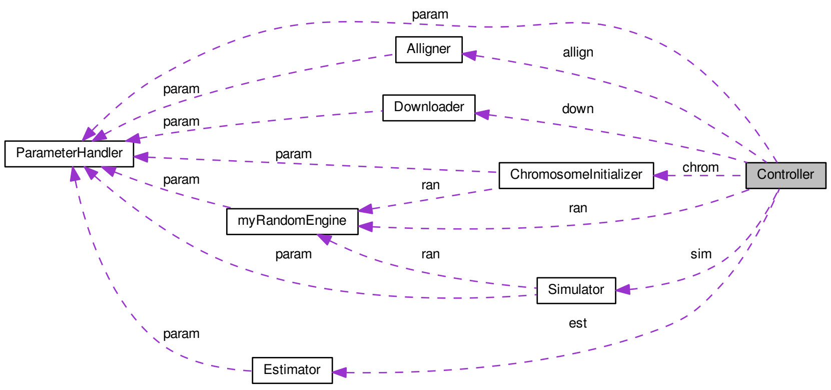
The documentation for this class was generated from the following files:

- /home/willy/workspace/VARUS/Implementation/ClusterEstimator.h
- /home/willy/workspace/VARUS/Implementation/ClusterEstimator.cpp

4.6 Controller Class Reference

```
#include <Controller.h>
```

Collaboration diagram for Controller:



Public Member Functions

- Controller (ParameterHandler ∗p)
- virtual ∼Controller ()
- void initialize ()
- void algorithm ()
- Run ∗ chooseNextRun ()
- bool continuing ()
- double score (const double v)
- void profit (Run ∗d)
- void calculateProfit (std::vector< Run ∗ > &runs)
- double score (UUmap &obs)
- void exportTotalObservationCSV (std::string name)
- void exportTotalObservationCSVlessInfo (std::string name)
- void printRun2File (Run ∗r)
- void createDieFromRun (Run ∗r)
- void createDiceFromRuns ()
- void updateDownloadableRuns ()

Public Attributes

- ParameterHandler ∗ param

  Controlls the other objects.
- ChromosomeInitializer ∗ chrom

- Alligner ∗ allign
- Downloader ∗ down
- Simulator ∗ sim
- myRandomEngine ∗ ran
- Estimator ∗ est
- unsigned int batchCount
- double maxProfit
- double totalProfit
- std::vector< Run ∗ > runs
- std::vector< Run ∗ > downloadableRuns
- UUmap totalObservations
- double totalScore

4.6.1 Constructor & Destructor Documentation

4.6.1.1 Controller::Controller (**ParameterHandler** ∗ *p*)

Different Objects are created based on the settings in the passed ParameterHandler p.

All pointers to objects not needed are initialized with nullpointers.

4.6.1.2 Controller::∼Controller ( ) [virtual]

4.6.2 Member Function Documentation

4.6.2.1 void Controller::algorithm ( )

The main part of the program.

The program will stay in the while-loop until the expected score drops below 0.

4.6.2.2 void Controller::calculateProfit (std::vector< **Run** ∗ > & *runs*)

Calculates the expected profit for all runs.

4.6.2.3 **Run** ∗ Controller::chooseNextRun ( )

This function returns a pointer to the Run that should be downloaded next.

If no estimator is selected, the next run is choosen randomly. If two or more Runs yield equal scores the Run is choosen randomly among the runs with the highest score.

4.6.2.4 bool Controller::continuing ( )

This function implements a criteria for exiting the program.

Returns true if the program should be continued, false otherwise.

4.6.2.5 void Controller::createDiceFromRuns ( )

Creates dice from all runs.

It is first looked for a file "completedRUns.txt". In this file the runs that are allready creadted as a die are listed. Then a "Runlist.txt" is read in. All runs except the runs being listed on the "completedRuns.txt"-file are then created as dice.

4.6.2.6 void Controller::createDieFromRun (Run ∗ *r*)

Completely downloads the given run stepwise and alligns it.

The resulting observations are then printed into a csv-file, which can later be used for simulations.

4.6.2.7 void Controller::exportTotalObservationCSV (std::string *name*)

Exports the observations in csv-format.

The passed string "name" is the accession-id of the run that was chosen to be downloaded from. Also the observations and p-vectors of all runs are exported.

4.6.2.8 void Controller::exportTotalObservationCSVlessInfo (std::string *name*)

Exports the observations in csv-format with less information.

Only the total observations and the best score and accession-id of the best run is exported. In exportTotalObservationsCSV() also the observations and p-vectors of all runs are exported.

4.6.2.9 void Controller::initialize ( )

Initialization of the chromosomeinitializer. Also sets the seed for the random-engines.

4.6.2.10 void Controller::printRun2File (Run ∗ *r*)

Exports the observations of the run into a csv-file.

This function is mainly used for creating the dice from real runs. These dice can later be used for simulations. In order to do that all reads in the run are downloaded and then alligned. Based on the observations it is possible to make probability distributions.

4.6.2.11 void Controller::profit (Run ∗ *d*)

Calculates the expected profit that will be achieved by downloading from the run d∗.

4.6.2.12 double Controller::score (const double *v*)

Returns the contribution of a transcript.

4.6.2.13 double Controller::score (UUmap & *obs*)

Calculates the score for the observations of a run or the totalObservations.

4.6.2.14 void Controller::updateDownloadableRuns ( )

Checks if all reads from a run are downloaded, and deletes it from downloadalbeRuns if it is the case.

Checks all runs. Should be improved by only checking the run that was downloaded instead of checking all runs.

4.6.3 Member Data Documentation

4.6.3.1 Alligner∗ Controller::allign

4.6.3.2 unsigned int Controller::batchCount

4.6.3.3 **ChromosomeInitializer**∗ Controller::chrom

4.6.3.4 **Downloader**∗ Controller::down

4.6.3.5 std::vector< **Run**∗ > Controller::downloadableRuns

4.6.3.6 **Estimator**∗ Controller::est

4.6.3.7 double Controller::maxProfit

4.6.3.8 **ParameterHandler**∗ Controller::param

Controlls the other objects.

The main-algorithm is implemented in algorithm().

4.6.3.9 **myRandomEngine**∗ Controller::ran

4.6.3.10 std::vector< **Run**∗ > Controller::runs

4.6.3.11 **Simulator**∗ Controller::sim

4.6.3.12 **UUmap** Controller::totalObservations

4.6.3.13 double Controller::totalProfit

4.6.3.14 double Controller::totalScore

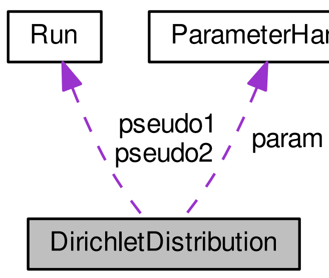
The documentation for this class was generated from the following files:

- /home/willy/workspace/VARUS/Implementation/Controller.h
- /home/willy/workspace/VARUS/Implementation/Controller.cpp

4.7 DirichletDistribution Class Reference

```
#include <DirichletDistribution.h>
```

Collaboration diagram for DirichletDistribution:



Public Member Functions

- DirichletDistribution (ParameterHandler ∗p, const double w)
- DirichletDistribution ()
- ∼DirichletDistribution ()
- void calculateWeight (const unsigned int n)
- void reset ()
- void calculateCsum ()
- void calculateCj ()
- void calculateCCountsAndQij ()
- void calculateMixtureParameters (const unsigned int num)
- double EalphaSum ()
- double calculateLikelihood (Run ∗r)
- void resetComplete (unsigned int numOfComps)
- void estimateP (Run ∗r)
- double calculateDistance (Run ∗r)
- void calculateSimpleP ()

Public Attributes

- ParameterHandler ∗ param

  Part of the Dirichlet-Mixture.
- Run ∗ pseudo1
- Run ∗ pseudo2
- UDmap alpha
- double weight
- double alphaSum
- unsigned int cSum
- unsigned int size
- std::vector< Run ∗ > runs
- UUmap aggregateCCounts
- UDmap Qij
- std::unordered_map< U32, UUmap > cj

4.7.1 Constructor & Destructor Documentation

4.7.1.1 DirichletDistribution::DirichletDistribution (ParameterHandler ∗ *p*, const double *w*)

4.7.1.2 DirichletDistribution::DirichletDistribution ( )

4.7.1.3 DirichletDistribution::∼DirichletDistribution ( )

4.7.2 Member Function Documentation

4.7.2.1 void DirichletDistribution::calculateCCountsAndQij ( )

Calculates the CCounts and Qijs.

4.7.2.2 void DirichletDistribution::calculateCj ( )

The j-th element of Cj gives the number of reads for all runs on that specific transcript-unit.

4.7.2.3 void DirichletDistribution::calculateCsum ( )

Calculates the total number of reads of all runs in this component.

4.7.2.4 double DirichletDistribution::calculateDistance (Run $*$ *r*)

4.7.2.5 double DirichletDistribution::calculateLikelihood (Run $*$ *r*)

Calculates the likelihood for a DirichletDistribution that it created the data in an Experiment. We use the logarithm of the gamma-function because tgamma is growing too fast. gamma(100) ∼= 10^155. We later transform back with exp().

$$L = \sum_{k=1}^{n_i} \log[\frac{\Gamma(\alpha_i^*)}{\Gamma(\alpha_i^*+c_i^{*k})} \cdot \prod_{j=1}^{T} \frac{\Gamma(\alpha_i^*+c_{j_i}^k)}{\Gamma(\alpha_i^*)}]$$
$$= \sum_{k=1}^{n_i} \{\log(\Gamma(\alpha_i^*)) - \log(\Gamma(\alpha_i^* + c_i^{*k})) + \sum_{j=1}^{T} [\log(\Gamma(\alpha_i^* q_{j_i} + c_{j_i}^k)) - \log(\Gamma(\alpha_i^* q_{j_i}))]\}$$

4.7.2.6 void DirichletDistribution::calculateMixtureParameters (const unsigned int *num*)

Calculates the parameters according to the runs sampled in the bins.

4.7.2.7 void DirichletDistribution::calculateSimpleP ( )

Calculates estimation for a one-component dirichlet-mixture.

We can use this insted of the more complex calculation because we sampled the runs into bins. The sampling reduced the problem to this very easy estimation.

4.7.2.8 void DirichletDistribution::calculateWeight (const unsigned int *n*)

Calculates the weight of the component based on how many runs have been sampled into this component in relation to how many runs exist in total.

4.7.2.9 double DirichletDistribution::EalphaSum ( )

Calculates the start-value for the newtons-method.

4.7.2.10 void DirichletDistribution::estimateP (Run $*$ *r*)

4.7.2.11 void DirichletDistribution::reset ( )

At each gibbs-sampling-step this function is called.

The runs are released from this component. The aggregateCCounts are set to 0.

4.7.2.12 void DirichletDistribution::resetComplete (unsigned int *numOfComps*)

This function is called at the beginning of a new training.

4.7.3 Member Data Documentation

4.7.3.1 UUmap DirichletDistribution::aggregateCCounts

4.7.3.2 UDmap DirichletDistribution::alpha

4.7.3.3 double DirichletDistribution::alphaSum

4.7.3.4 std::unordered_map<U32, UUmap > DirichletDistribution::cj

4.7.3.5 unsigned int DirichletDistribution::cSum

4.7.3.6 ParameterHandler* DirichletDistribution::param

Part of the Dirichlet-Mixture.

Each Dirichletdistribution can be imagined as a die-factory that produces a certain type of dice. However the actual dice may vary around this distributio.

4.7.3.7 Run* DirichletDistribution::pseudo1

4.7.3.8 Run* DirichletDistribution::pseudo2

4.7.3.9 UDmap DirichletDistribution::Qij

4.7.3.10 std::vector<Run*> DirichletDistribution::runs

4.7.3.11 unsigned int DirichletDistribution::size

4.7.3.12 double DirichletDistribution::weight

The documentation for this class was generated from the following files:

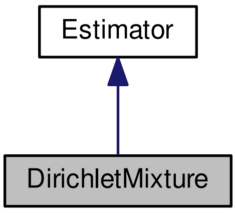
- /home/willy/workspace/VARUS/Implementation/DirichletDistribution.h
- /home/willy/workspace/VARUS/Implementation/DirichletDistribution.cpp

4.8 DirichletMixture Class Reference

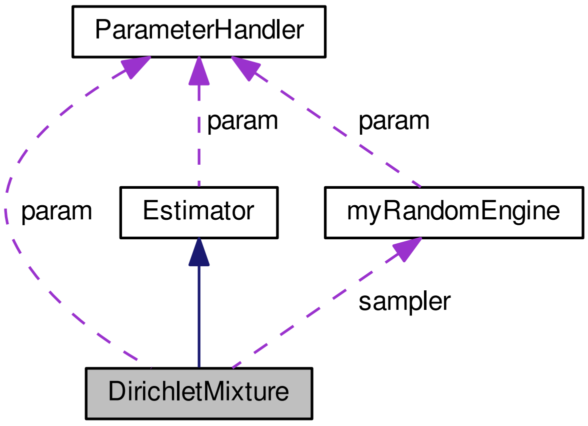
The dirichlet-mixture is used, when it is assumed that the runs are not independently distributed, but instead a handfull of distributions represent all runs.

`#include <DirichletMixture.h>`

Inheritance diagram for DirichletMixture:



Collaboration diagram for DirichletMixture:



Public Member Functions

- double logLikelihood (DirichletDistribution &comp)

  Calculates the likelihood for a DirichletDistribution that it created the data in an Experiment.
- double logLikelihood (DirichletDistribution &comp, const double alphaSum)
- double dL (DirichletDistribution &comp, const double x)
- double ddL (DirichletDistribution &comp, const double x)
- double newtonsMethod (unsigned int compNum, DirichletDistribution &c, const double start, unsigned int maxIterations, const double tol)
- double XiLog (std::vector< double > &logBComponents2, Run ∗r, const U32 i)

  Karplus formula (39).
- double myLogBeta (UDmap &v)
- double myLogBeta (UDmap &v, UUmap &v2)
- void calculateMixtureParameters (const unsigned int num)
- void train (std::vector< Run ∗ > &runs)
- void calculateLikelihoods (Run ∗r, std::vector< double > &likelihoods)
- bool checkComponentsValid ()
- void estimateP (std::vector< Run ∗ > &runs, const unsigned int IterationNumber)
- void initializeP (std::vector< Run ∗ > &runs)
- void calculateP (std::vector< Run ∗ > &runs)
- void exportLikelihoodCSV (const unsigned int it, const unsigned int compNum, DirichletDistribution &comp, const double a, const double b, const int n, std::vector< double > &newton)
- void exportMixture (const unsigned int iterationNumber, USmap &translate2Str)
- void setTranslate2Str (USmap &translate2Str)
- void exportRunNames ()
- void calculateSimpleP ()
- DirichletMixture (ParameterHandler ∗p)
- DirichletMixture ()
- virtual ∼DirichletMixture ()

Public Attributes

- ParameterHandler ∗ param
- myRandomEngine ∗ sampler
- std::vector
  < DirichletDistribution > components
- unsigned int current_trainingsIteration
- unsigned int iterationNumber
- std::unordered_map< double,
  double > lgammaHash
- USmap translate2Str

Additional Inherited Members

4.8.1 Detailed Description

The dirichlet-mixture is used, when it is assumed that the runs are not independently distributed, but instead a handfull of distributions represent all runs.

Detailed description starts here.

4.8.2 Constructor & Destructor Documentation

4.8.2.1 DirichletMixture::DirichletMixture (ParameterHandler ∗ *p*)

Creates the components and initializes other values.

The components are containers, into which the runs are sampled. Based on the samplings the parameters are then estimated.

4.8.2.2 DirichletMixture::DirichletMixture ( )

4.8.2.3 DirichletMixture::∼DirichletMixture ( ) [virtual]

4.8.3 Member Function Documentation

4.8.3.1 void DirichletMixture::calculateLikelihoods (Run ∗ *r*, std::vector< double > & *likelihoods*)

4.8.3.2 void DirichletMixture::calculateMixtureParameters (const unsigned int *num*)

Calculates the parameters according to the runs sampled in the bins.

4.8.3.3 void DirichletMixture::calculateP (std::vector< Run ∗ > & *runs*)

calculates the estimators p for all runs after training.

4.8.3.4 void DirichletMixture::calculateSimpleP ( )

4.8.3.5 bool DirichletMixture::checkComponentsValid ( )

4.8.3.6 double DirichletMixture::ddL (DirichletDistribution & *comp*, const double *x*)

The second derivate of the likelihood needed for newtons-method.

Formula (11) in Altschull-paper.

4.8.3.7 double DirichletMixture::dL (DirichletDistribution & *comp*, const double *x*)

The derivate of the likelihood needed for newtons-method.

Formula (10) in Altschull-paper.

4.8.3.8 void DirichletMixture::estimateP (std::vector< Run * > & *runs*, const unsigned int *IterationNumber*) [virtual]

Estimates the p for all Experiments.

Implements Estimator.

4.8.3.9 void DirichletMixture::exportLikelihoodCSV (const unsigned int *it*, const unsigned int *compNum*, DirichletDistribution & *comp*, const double *a*, const double *b*, const int *n*, std::vector< double > & *newton*)

Exports the likelihood as csv-file in the range from a to b with n equaly long steps.

A folder is created if necessary in which all the data for this component will go. The csv-file has four columns "steps;L;dL;newtons". L is the likelihood, dL is its derivate and newtons holds the values that have been produced by the newtons-method. The last value is the return-value of the newtons-method.

4.8.3.10 void DirichletMixture::exportMixture (const unsigned int *iterationNumber*, USmap & *translate2Str*)

Exports the parameters of the mixture as a csv-file.

4.8.3.11 void DirichletMixture::exportRunNames ( )

Exports the names of the runs sampled into the bins as txt-files.

Run-names are written to file line-wise.

4.8.3.12 void DirichletMixture::initializeP (std::vector< Run * > & *runs*)

4.8.3.13 double DirichletMixture::logLikelihood (DirichletDistribution & *comp*)

Calculates the likelihood for a DirichletDistribution that it created the data in an Experiment.

Formula for calculation can be found in Altschull...

4.8.3.14 double DirichletMixture::logLikelihood (DirichletDistribution & *comp*, const double *alphaSum*)

Calculates the likelihood for a DirichletDistribution that it created the data in an Experiment.

Formula (8) in Altschull-paper.

4.8.3.15 double DirichletMixture::myLogBeta (UDmap & *v*)

4.8.3.16 double DirichletMixture::myLogBeta (UDmap & *v*, UUmap & *v2*)

Calculates the logarithm of the beta-function of the vector v.

4.8.3.17 double DirichletMixture::newtonsMethod (unsigned int *compNum,* DirichletDistribution & *c,* const double *start,* unsigned int *maxIterations,* const double *tol*)

Newtons method calculates the expected alphasum.

The likelihood depending on alphasum is maximized. This is done by calculating the arguments at which the first derivate is zero.

4.8.3.18 void DirichletMixture::setTranslate2Str (USmap & *translate2Str*)

4.8.3.19 void DirichletMixture::train (std::vector< Run ∗ > & *runs*)

The components are trained with gibbs-samplin.

4.8.3.20 double DirichletMixture::XiLog (std::vector< double > & *BComponents,* Run ∗ *e,* const U32 *i*)

Karplus formula (39).

Karplus formula (39)

4.8.4 Member Data Documentation

4.8.4.1 std::vector<DirichletDistribution> DirichletMixture::components

4.8.4.2 unsigned int DirichletMixture::current_trainingsIteration

4.8.4.3 unsigned int DirichletMixture::iterationNumber

4.8.4.4 std::unordered_map<double,double> DirichletMixture::lgammaHash

4.8.4.5 ParameterHandler∗ DirichletMixture::param

4.8.4.6 myRandomEngine∗ DirichletMixture::sampler

4.8.4.7 USmap DirichletMixture::translate2Str

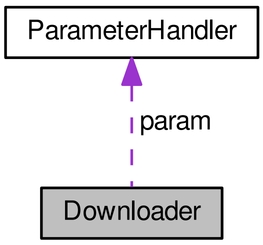
The documentation for this class was generated from the following files:

- /home/willy/workspace/VARUS/Implementation/DirichletMixture.h
- /home/willy/workspace/VARUS/Implementation/DirichletMixture.cpp

4.9 Downloader Class Reference

```
#include <Downloader.h>
```

Collaboration diagram for Downloader:



Public Member Functions

- Downloader (ParameterHandler ∗p)
- virtual ∼Downloader ()
- std::string shellCommand (Run ∗r)
- void nextBatchIndices (Run ∗r)
- void getBatch (Run ∗r, bool all=false)

Public Attributes

- ParameterHandler ∗ param

4.9.1 Constructor & Destructor Documentation

4.9.1.1 Downloader::Downloader (**ParameterHandler** ∗ *p*)

4.9.1.2 Downloader::∼Downloader ( ) `[virtual]`

4.9.2 Member Function Documentation

4.9.2.1 void Downloader::getBatch (**Run** ∗ *r*, bool *all* = `false`)

The next batch will be downloaded according to the sigma-vector. The command for the download is invoked throught the shell and calls fastq-dump.

4.9.2.2 void Downloader::nextBatchIndices (**Run** ∗ *r*)

Calculates the batch-indices N and X.

4.9.2.3 std::string Downloader::shellCommand (**Run** ∗ *r*)

4.9.3 Member Data Documentation

4.9.3.1 **ParameterHandler**∗ Downloader::param

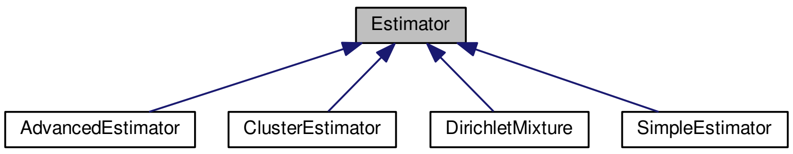
The documentation for this class was generated from the following files:

- /home/willy/workspace/VARUS/Implementation/Downloader.h
- /home/willy/workspace/VARUS/Implementation/Downloader.cpp

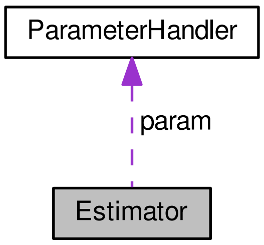
4.10 Estimator Class Reference

```
#include <Estimator.h>
```

Inheritance diagram for Estimator:



Collaboration diagram for Estimator:



Public Member Functions

- Estimator ()
- Estimator (ParameterHandler ∗p)
- virtual ∼Estimator ()
- virtual void estimateP (std::vector< Run ∗ > &runs, unsigned int iterationNumber)=0
- void initializeRuns (std::vector< Run ∗ > &runs, UUmap &transcriptUnits)

Public Attributes

- ParameterHandler ∗ param

Static Public Attributes

- static constexpr double precision = 0.0001

4.10.1 Constructor & Destructor Documentation

4.10.1.1 Estimator::Estimator ( )

4.10.1.2 Estimator::Estimator (**ParameterHandler** ∗ *p*)

4.10.1.3 Estimator::∼Estimator ( ) `[virtual]`

4.10.2 Member Function Documentation

4.10.2.1 virtual void Estimator::estimateP (std::vector< **Run** ∗ > & *runs,* unsigned int *iterationNumber*) `[pure virtual]`

Implemented in DirichletMixture, ClusterEstimator, AdvancedEstimator, and SimpleEstimator.

4.10.2.2 void Estimator::initializeRuns (std::vector< **Run** ∗ > & *runs,* **UUmap** & *transcriptUnits*)

4.10.3 Member Data Documentation

4.10.3.1 **ParameterHandler**∗ Estimator::param

4.10.3.2 constexpr double Estimator::precision = 0.0001 `[static]`

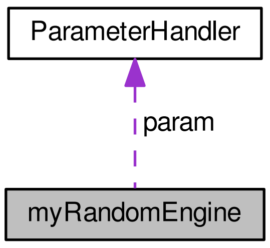
The documentation for this class was generated from the following files:

- /home/willy/workspace/VARUS/Implementation/Estimator.h
- /home/willy/workspace/VARUS/Implementation/Estimator.cpp

4.11 myRandomEngine Class Reference

`#include <myRandomEngine.h>`

Collaboration diagram for myRandomEngine:



Public Member Functions

- myRandomEngine ()
- myRandomEngine (ParameterHandler ∗p)

- virtual ~myRandomEngine ()
- template<typename V >
  V select_randomly (std::vector< V > &vec)
- template<typename V >
  void shuffle (std::vector< V > &vec)
- unsigned int select_randomly (unsigned int max)
- unsigned int select_randomly_p (std::vector< double > &vec)
- void seed (unsigned int t)

Public Attributes

- std::mt19937 gen
- ParameterHandler ∗ param

4.11.1 Constructor & Destructor Documentation

4.11.1.1 myRandomEngine::myRandomEngine ( )

Seed is set randomly depending on the time.

4.11.1.2 myRandomEngine::myRandomEngine (**ParameterHandler** ∗ *p*)

Seed is set randomly depending on the time or according to the value randomSeed in ParameterHandler param.

4.11.1.3 myRandomEngine::~myRandomEngine ( ) `[virtual]`

4.11.2 Member Function Documentation

4.11.2.1 void myRandomEngine::seed (unsigned int *t*)

Set the seed of the engine. Only needed for unit-testing to make things reproduce-able.

4.11.2.2 template<typename V > V myRandomEngine::select_randomly (std::vector< V > & *vec*) `[inline]`

4.11.2.3 unsigned int myRandomEngine::select_randomly (unsigned int *max*)

4.11.2.4 unsigned int myRandomEngine::select_randomly_p (std::vector< double > & *vec*)

4.11.2.5 template<typename V > void myRandomEngine::shuffle (std::vector< V > & *vec*) `[inline]`

4.11.3 Member Data Documentation

4.11.3.1 std::mt19937 myRandomEngine::gen

4.11.3.2 **ParameterHandler**∗ myRandomEngine::param

The documentation for this class was generated from the following files:

- /home/willy/workspace/VARUS/Implementation/myRandomEngine.h
- /home/willy/workspace/VARUS/Implementation/myRandomEngine.cpp

4.12 ParameterHandler Class Reference

This class holds all the data, that is read in as command-line arguments.

`#include <ParameterHandler.h>`

Public Member Functions

- void read_parameters_from_file (std::string path)
- void export_parameters ()
- void readArguments (int argc, char ∗argv[])
- void readInputRuns ()
- std::ostream & printParameters (std::ostream &os)
- void print_usage ()
- std::string lineLength (const std::string s, const unsigned int l, const unsigned int maxS)
- template<typename K >
  void add_parameter (std::string name, K a)
- ParameterHandler ()
- ∼ParameterHandler ()
- template<typename K >
  void add_parameter (string name, K a)

Public Attributes

- std::string pathToSTAR
- std::string pathToRuns
- std::string readFilesIn
- std::string genomeDir
- std::string outFileNamePrefix
- ul runThreadN
- ul blockSize
- ul batchSize
- double pseudoCount
- double lambda
- int logScore
- double cost
- int createDice
- int simulation
- int estimator
- int dieList
- int lessInfo = 1
- ul components
- ul numOfBlocks
- ul trainingsIterations
- int loadAllOnce
- int verbosity
- int verbosityDebug
- int randomSeed
- int profitCondition
- ul newtonIterations
- double newtonPercission
- int exportObservationsToFile
- int readParametersFromFile
- std::string pathToParameters

- int exportParametersToFile
- std::string pathToDice
- int deleteLater
- unsigned int maxBatches
- int ignoreReadNum
- int simpleDM
- int kMeansIterations
- int exportNewtons
- bool readAllready
- int argc_saved
- char ** argv_saved
- std::map< std::string,
  std::string > parameters
- std::map< std::string,
  std::string > usage

4.12.1 Detailed Description

This class holds all the data, that is read in as command-line arguments.

A raw pointer is passed to all other instances, that need to have access to the input data. A std::unique_ptr is created in main.cpp that is responsible for this instance.

4.12.2 Constructor & Destructor Documentation

4.12.2.1 ParameterHandler::ParameterHandler ( )

All parameters are initialized with default-values. To change these values call readArguments().

4.12.2.2 ParameterHandler::∼ParameterHandler ( )

4.12.3 Member Function Documentation

4.12.3.1 template<typename K > void ParameterHandler::add_parameter (std::string *name*, K *a*)

4.12.3.2 template<typename K > void ParameterHandler::add_parameter (string *name*, K *a*)

This function is called when a new parameter should be added.

With this function and the map parameters it is possible to import and export the values of the parameters easily.

4.12.3.3 void ParameterHandler::export_parameters ( )

The parameters are exported into a file.

The parameters can be read in from this file with read_parameters_from_file().

4.12.3.4 string ParameterHandler::lineLength (const std::string *s*, const unsigned int *l*, const unsigned int *maxS*)

4.12.3.5 void ParameterHandler::print_usage ( )

Prints the usage of the program.

The map<string,string> usage holds the name and its description for each parameter.

4.12.3.6 std::ostream & ParameterHandler::printParameters (std::ostream & *os*)

The parameters are appended to the std::ostream os.

The format is in such a way, that a call of read_parameters_from_file() with an output-file created with this method can be read in again. For example: Let lambda=1.2

outFile:–lambda 1.2

4.12.3.7 void ParameterHandler::read_parameters_from_file (std::string *path*)

The parameters are read from a file.

The parameters can be written to a file with exportParameters().

readArguments() is called twice, because in the first call the parameters form the file are read, in the second call the parameters from the command-line are added.

4.12.3.8 void ParameterHandler::readArguments (int *argc,* char ∗ *argv[]*)

Reads the command line arguments.

if the user selects the simulation, no real runs will be used

If the user selected a path readParameters from file, we have to call read_parameters_from_file(), which in turn will call readArguments. So we need to set a flag in order to not get stuck in a loop.

4.12.3.9 void ParameterHandler::readInputRuns ( )

4.12.4 Member Data Documentation

4.12.4.1 int ParameterHandler::argc_saved

4.12.4.2 char∗∗ ParameterHandler::argv_saved

4.12.4.3 ul ParameterHandler::batchSize

4.12.4.4 ul ParameterHandler::blockSize

4.12.4.5 ul ParameterHandler::components

4.12.4.6 double ParameterHandler::cost

4.12.4.7 int ParameterHandler::createDice

4.12.4.8 int ParameterHandler::deleteLater

4.12.4.9 int ParameterHandler::dieList

4.12.4.10 int ParameterHandler::estimator

4.12.4.11 int ParameterHandler::exportNewtons

4.12.4.12 int ParameterHandler::exportObservationsToFile

4.12.4.13 int ParameterHandler::exportParametersToFile

4.12.4.14 std::string ParameterHandler::genomeDir

4.12.4.15 int ParameterHandler::ignoreReadNum

4.12.4.16 int ParameterHandler::kMeansIterations

4.12.4.17 double ParameterHandler::lambda

4.12.4.18 int ParameterHandler::lessInfo = 1

4.12.4.19 int ParameterHandler::loadAllOnce

4.12.4.20 int ParameterHandler::logScore

4.12.4.21 unsigned int ParameterHandler::maxBatches

4.12.4.22 ul ParameterHandler::newtonIterations

4.12.4.23 double ParameterHandler::newtonPercission

4.12.4.24 ul ParameterHandler::numOfBlocks

4.12.4.25 std::string ParameterHandler::outFileNamePrefix

4.12.4.26 std::map< std::string, std::string > ParameterHandler::parameters

4.12.4.27 std::string ParameterHandler::pathToDice

4.12.4.28 std::string ParameterHandler::pathToParameters

4.12.4.29 std::string ParameterHandler::pathToRuns

4.12.4.30 std::string ParameterHandler::pathToSTAR

4.12.4.31 int ParameterHandler::profitCondition

4.12.4.32 double ParameterHandler::pseudoCount

4.12.4.33 int ParameterHandler::randomSeed

4.12.4.34 bool ParameterHandler::readAllready

4.12.4.35 std::string ParameterHandler::readFilesIn

4.12.4.36 int ParameterHandler::readParametersFromFile

4.12.4.37 ul ParameterHandler::runThreadN

4.12.4.38 int ParameterHandler::simpleDM

4.12.4.39 int ParameterHandler::simulation

4.12.4.40 ul ParameterHandler::trainingsIterations

4.12.4.41 std::map< std::string, std::string > ParameterHandler::usage

4.12.4.42 int ParameterHandler::verbosity

4.12.4.43 int ParameterHandler::verbosityDebug

The documentation for this class was generated from the following files:

- /home/willy/workspace/VARUS/Implementation/ParameterHandler.h
- /home/willy/workspace/VARUS/Implementation/ParameterHandler.cpp

4.13 RNAread Class Reference

```
#include <RNAread.h>
```

Public Member Functions

- bool UMR ()
- RNAread ()
- ∼RNAread ()

Public Attributes

- std::unordered_map
  < std::string, unsigned int > transcriptUnits

4.13.1 Constructor & Destructor Documentation

4.13.1.1 RNAread::RNAread ( )

4.13.1.2 RNAread::∼RNAread ( )

4.13.2 Member Function Documentation

4.13.2.1 bool RNAread::UMR ( )

4.13.3 Member Data Documentation

4.13.3.1 std::unordered_map<std::string,unsigned int> RNAread::transcriptUnits

The documentation for this class was generated from the following files:

- /home/willy/workspace/VARUS/Implementation/RNAread.h
- /home/willy/workspace/VARUS/Implementation/RNAread.cpp

4.14 Run Class Reference

```
#include <Run.h>
```

Public Member Functions

- Run ()
- Run (std::string accesionId, const unsigned int transcriptBlocks, const unsigned int numOfSpots, const unsigned int batchSize)
- ∼Run ()

Public Attributes

- UUmap observations
- UDmap p
- double pNoObs
- unsigned int N
- unsigned int X
- std::string accesionId
- bool paired
- unsigned int numOfSpots
- unsigned int maxNumOfBatches
- unsigned int sigmaIndex
- unsigned int batchSize
- std::vector< int > sigma
- unsigned int timesDownloaded
- unsigned int observationSum
- U32 transcriptBlocks
- double expectedProfit
- UDmap q

4.14.1 Constructor & Destructor Documentation

4.14.1.1 Run::Run ( )

4.14.1.2 Run::Run (std::string *accesionId,* const unsigned int *transcriptBlocks,* const unsigned int *numOfSpots,* const unsigned int *batchSize*)

4.14.1.3 Run::∼Run ( )

4.14.2 Member Data Documentation

4.14.2.1 std::string Run::accesionId

4.14.2.2 unsigned int Run::batchSize

4.14.2.3 double Run::expectedProfit

4.14.2.4 unsigned int Run::maxNumOfBatches

4.14.2.5 unsigned int Run::N

4.14.2.6 unsigned int Run::numOfSpots

4.14.2.7 UUmap Run::observations

4.14.2.8 unsigned int Run::observationSum

4.14.2.9 UDmap Run::p

4.14.2.10 bool Run::paired

4.14.2.11 double Run::pNoObs

4.14.2.12 UDmap Run::q

4.14.2.13 std::vector<int> Run::sigma

4.14.2.14 unsigned int Run::sigmaIndex

4.14.2.15 unsigned int Run::timesDownloaded

4.14.2.16 U32 Run::transcriptBlocks

4.14.2.17 unsigned int Run::X

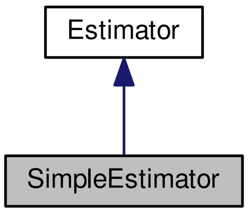
The documentation for this class was generated from the following files:

- /home/willy/workspace/VARUS/Implementation/Run.h
- /home/willy/workspace/VARUS/Implementation/Run.cpp

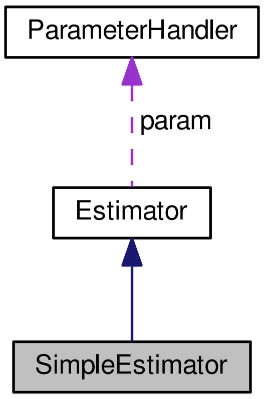
4.15 SimpleEstimator Class Reference

`#include <SimpleEstimator.h>`

Inheritance diagram for SimpleEstimator:



Collaboration diagram for SimpleEstimator:



Public Member Functions

- SimpleEstimator (ParameterHandler ∗p)
- virtual ∼SimpleEstimator ()
- void estimateP (std::vector< Run ∗ > &runs, unsigned int IterationNumber)

Additional Inherited Members

4.15.1 Constructor & Destructor Documentation

4.15.1.1 SimpleEstimator::SimpleEstimator (**ParameterHandler** ∗ *p*)

4.15.1.2 SimpleEstimator::∼SimpleEstimator ( ) `[virtual]`

4.15.2 Member Function Documentation

4.15.2.1 void SimpleEstimator::estimateP (std::vector< **Run** ∗ > & *runs*, unsigned int *IterationNumber*) `[virtual]`

Implements Estimator.

The documentation for this class was generated from the following files:

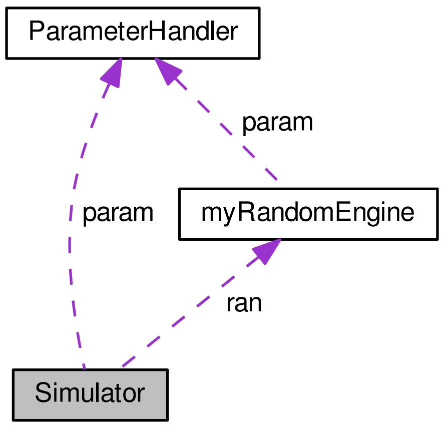
- /home/willy/workspace/VARUS/Implementation/SimpleEstimator.h
- /home/willy/workspace/VARUS/Implementation/SimpleEstimator.cpp

4.16 Simulator Class Reference

This class simulates the download and allignment-steps.

```
#include <Simulator.h>
```

Collaboration diagram for Simulator:



Public Member Functions

- Simulator (ParameterHandler ∗p)
- ∼Simulator ()
- void readInputRuns ()
- void initializeRuns (std::vector< Run ∗ > &runs)
- void simulateObservations (Run ∗r, UUmap &totalObservations)

Public Attributes

- ParameterHandler ∗ param
- myRandomEngine ∗ ran
- std::vector< std::string > inputRuns
- std::unordered_map
  < std::string, std::vector
  < double > > p_hidden
- SUmap translate2int
- USmap translate2str
- std::vector< U32 > order

4.16.1 Detailed Description

This class simulates the download and allignment-steps.

The simulation basically consists of producing vectors according to a random-distribution of the runs. These vectors are then interpreted as actual observations of reads mapping to transcripts.

4.16.2 Constructor & Destructor Documentation

4.16.2.1 Simulator::Simulator (**ParameterHandler** ∗ *p*)

4.16.2.2 Simulator::∼Simulator ( )

4.16.3 Member Function Documentation

4.16.3.1 void Simulator::initializeRuns (std::vector< Run ∗ > & *runs*)

4.16.3.2 void Simulator::readInputRuns ( )

Reads in the accession-ids of the Runs to download from.

It is read from "Runlist.txt"

4.16.3.3 void Simulator::simulateObservations (Run ∗ *r*, UUmap & *totalObservations*)

4.16.4 Member Data Documentation

4.16.4.1 std::vector<std::string> Simulator::inputRuns

4.16.4.2 std::vector<U32> Simulator::order

4.16.4.3 std::unordered_map<std::string,std::vector<double> > Simulator::p_hidden

4.16.4.4 ParameterHandler∗ Simulator::param

4.16.4.5 myRandomEngine∗ Simulator::ran

4.16.4.6 SUmap Simulator::translate2int

4.16.4.7 USmap Simulator::translate2str

The documentation for this class was generated from the following files:

- /home/willy/workspace/VARUS/Implementation/Simulator.h
- /home/willy/workspace/VARUS/Implementation/Simulator.cpp

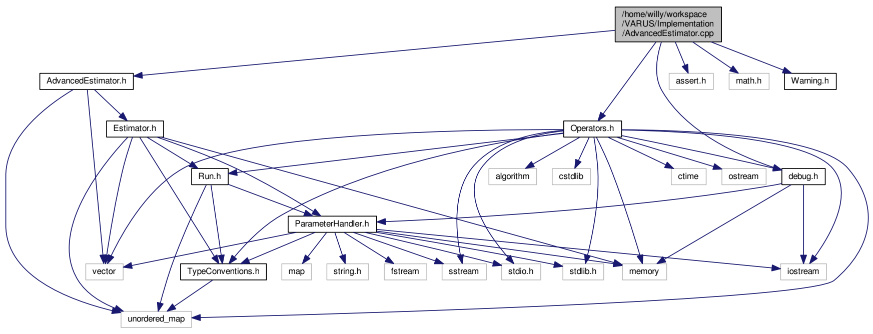
Chapter 5

File Documentation

5.1 /home/willy/workspace/VARUS/Implementation/AdvancedEstimator.cpp File Reference

```
#include "AdvancedEstimator.h"
#include "debug.h"
#include "Operators.h"
#include "assert.h"
#include "math.h"
#include "Warning.h"
```

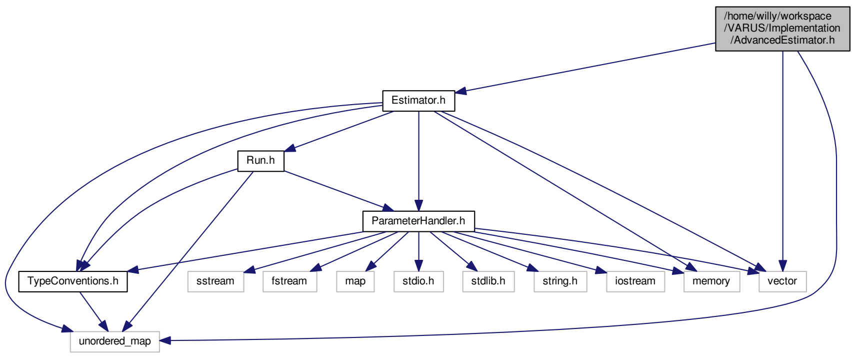
Include dependency graph for AdvancedEstimator.cpp:



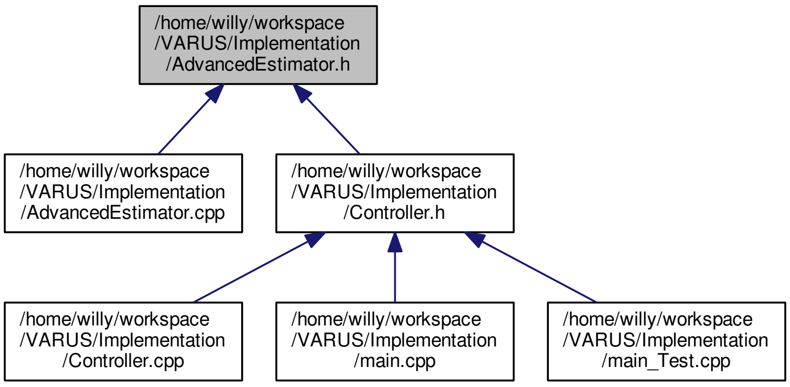
5.2 /home/willy/workspace/VARUS/Implementation/AdvancedEstimator.h File Reference

```
#include "Estimator.h"
#include <unordered_map>
#include <vector>
```

Include dependency graph for AdvancedEstimator.h:



This graph shows which files directly or indirectly include this file:



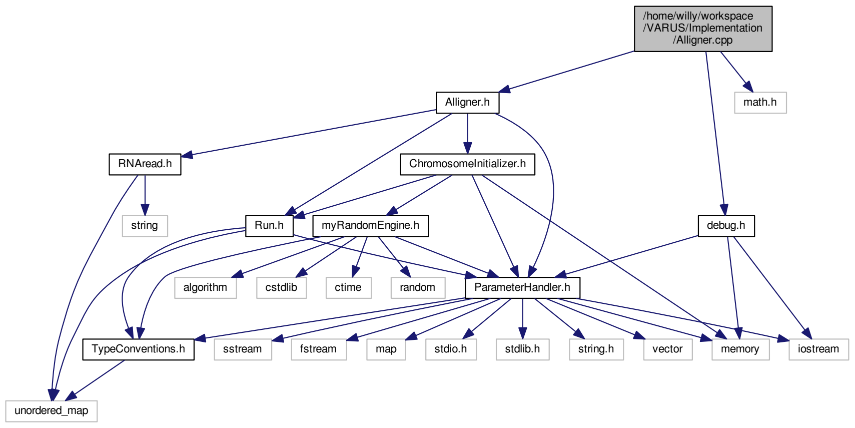
Classes

- class AdvancedEstimator

5.3 /home/willy/workspace/VARUS/Implementation/Alligner.cpp File Reference

```
#include "Alligner.h"
#include "debug.h"
#include "math.h"
```

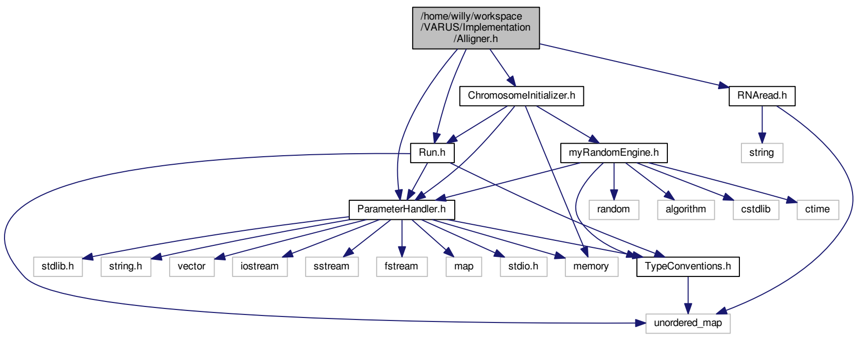
Include dependency graph for Alligner.cpp:



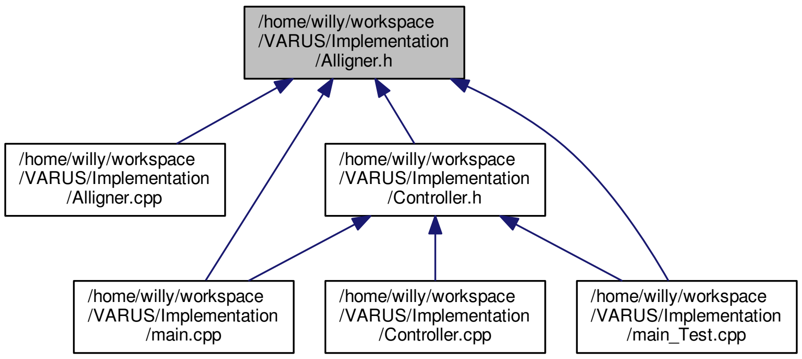
5.4 /home/willy/workspace/VARUS/Implementation/Alligner.h File Reference

```
#include "ParameterHandler.h"
#include "Run.h"
#include "RNAread.h"
#include "ChromosomeInitializer.h"
```

Include dependency graph for Alligner.h:



This graph shows which files directly or indirectly include this file:



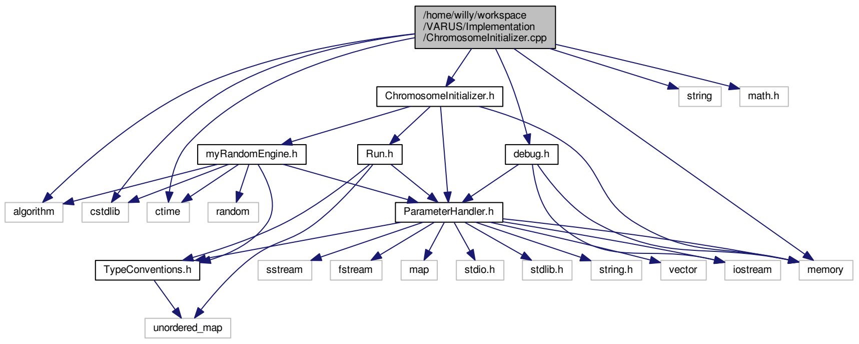
Classes

- class Alligner

5.5 /home/willy/workspace/VARUS/Implementation/ChromosomeInitializer.cpp File Reference

```
#include "ChromosomeInitializer.h"
#include <string>
#include "debug.h"
#include <math.h>
#include <memory>
#include <algorithm>
#include <cstdlib>
#include <ctime>
```

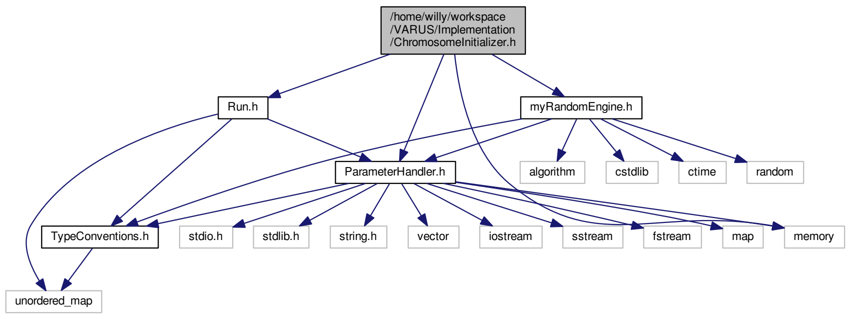
Include dependency graph for ChromosomeInitializer.cpp:



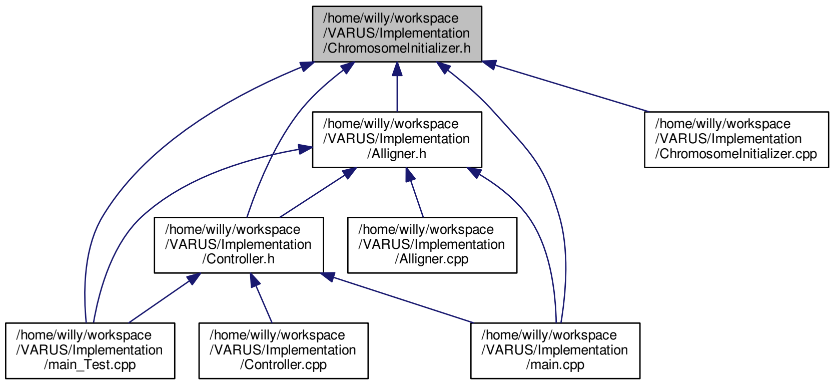
5.6 /home/willy/workspace/VARUS/Implementation/ChromosomeInitializer.h File Reference

```
#include "Run.h"
#include "ParameterHandler.h"
#include <memory>
#include "myRandomEngine.h"
```

Include dependency graph for ChromosomeInitializer.h:



This graph shows which files directly or indirectly include this file:



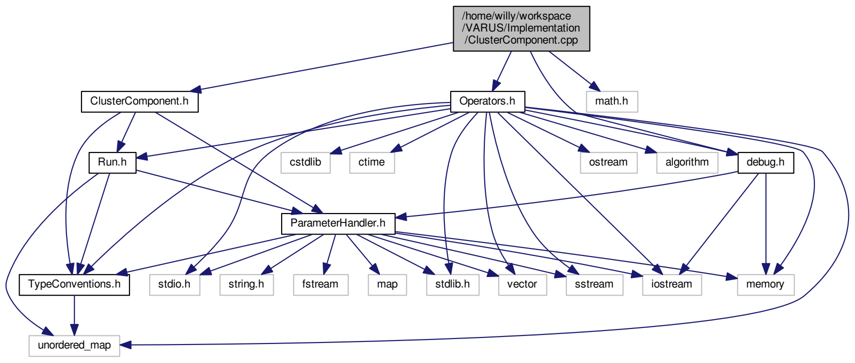
Classes

- class ChromosomeInitializer

5.7 /home/willy/workspace/VARUS/Implementation/ClusterComponent.cpp File Reference

```
#include "ClusterComponent.h"
#include "debug.h"
#include "Operators.h"
#include "math.h"
```

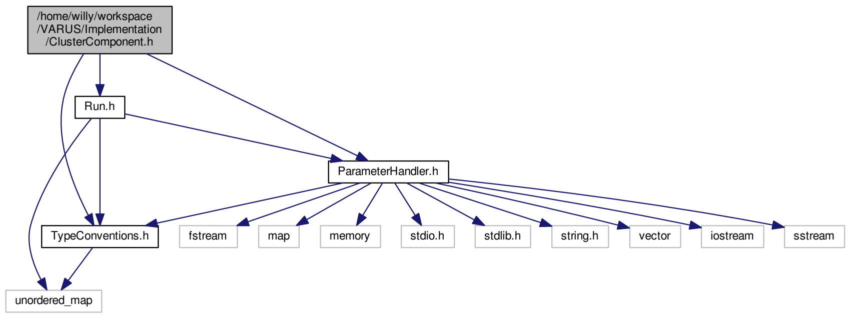
Include dependency graph for ClusterComponent.cpp:



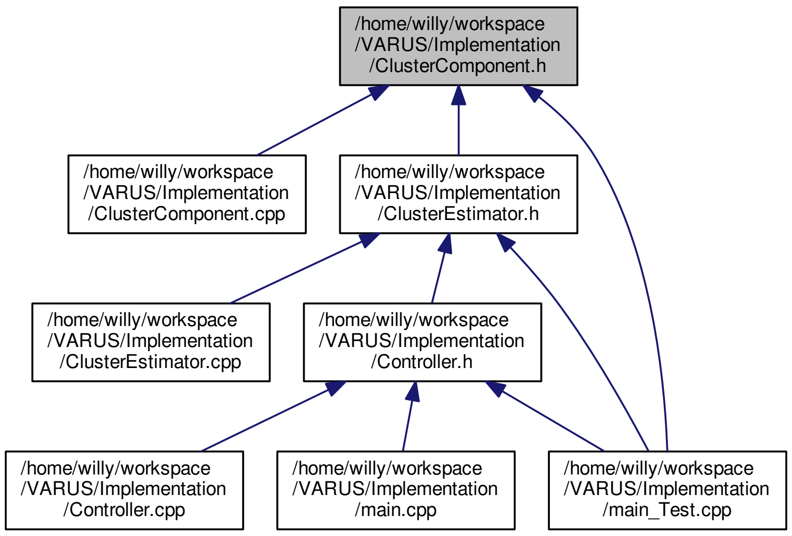
5.8 /home/willy/workspace/VARUS/Implementation/ClusterComponent.h File Reference

```
#include "TypeConventions.h"
#include "Run.h"
#include "ParameterHandler.h"
```

Include dependency graph for ClusterComponent.h:



This graph shows which files directly or indirectly include this file:


/home/willy/workspace /VARUS/Implementation /ClusterComponent.h
/home/willy/workspace /VARUS/Implementation /ClusterComponent.cpp
/home/willy/workspace /VARUS/Implementation /ClusterEstimator.h
/home/willy/workspace /VARUS/Implementation /ClusterEstimator.cpp
/home/willy/workspace /VARUS/Implementation /Controller.h
/home/willy/workspace /VARUS/Implementation /Controller.cpp
/home/willy/workspace /VARUS/Implementation /main.cpp
/home/willy/workspace /VARUS/Implementation /main_Test.cpp

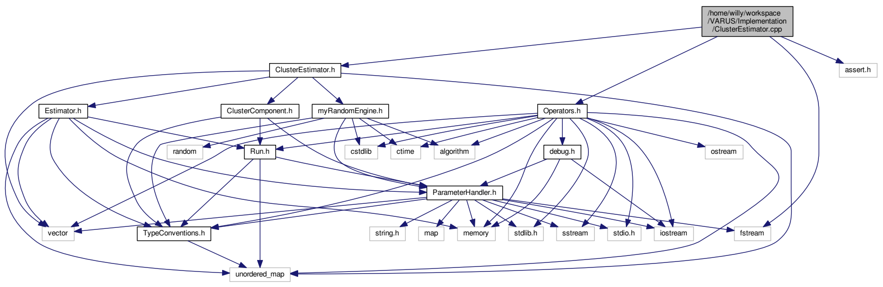
Classes

- class ClusterComponent

5.9 /home/willy/workspace/VARUS/Implementation/ClusterEstimator.cpp File Reference

```
#include "ClusterEstimator.h"
#include <assert.h>
#include "Operators.h"
#include <fstream>
```

Include dependency graph for ClusterEstimator.cpp:



Functions

- bool pred (Run ∗s)

5.9.1 Function Documentation

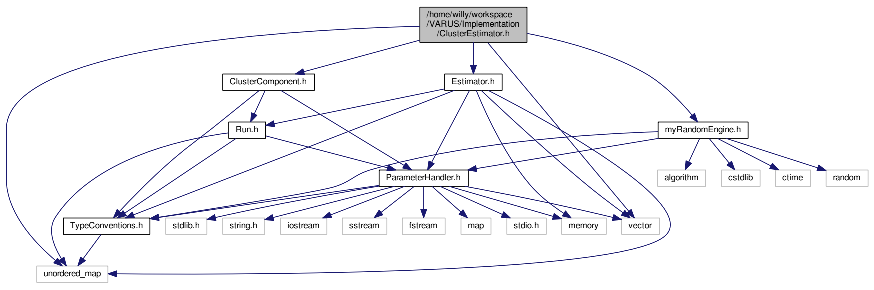
5.9.1.1 bool pred (**Run** ∗ ***s***)

Checks if the observationSum of a given run is equal to 0.

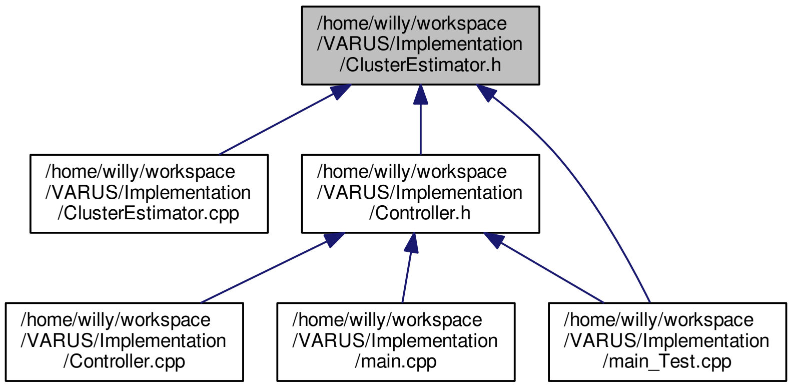
5.10 /home/willy/workspace/VARUS/Implementation/ClusterEstimator.h File Reference

```
#include "Estimator.h"
#include <unordered_map>
#include <vector>
#include "ClusterComponent.h"
#include "myRandomEngine.h"
```

Include dependency graph for ClusterEstimator.h:



This graph shows which files directly or indirectly include this file:


/home/willy/workspace /VARUS/Implementation /ClusterEstimator.h
/home/willy/workspace /VARUS/Implementation /ClusterEstimator.cpp
/home/willy/workspace /VARUS/Implementation /Controller.h
/home/willy/workspace /VARUS/Implementation /Controller.cpp
/home/willy/workspace /VARUS/Implementation /main.cpp
/home/willy/workspace /VARUS/Implementation /main_Test.cpp

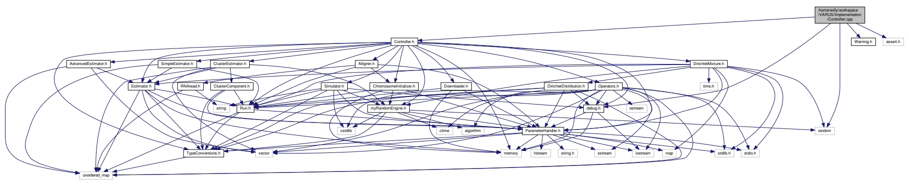
Classes

- class ClusterEstimator

5.11 /home/willy/workspace/VARUS/Implementation/Controller.cpp File Reference

```
#include "Controller.h"
#include <random>
#include "Warning.h"
#include "debug.h"
#include <assert.h>
```

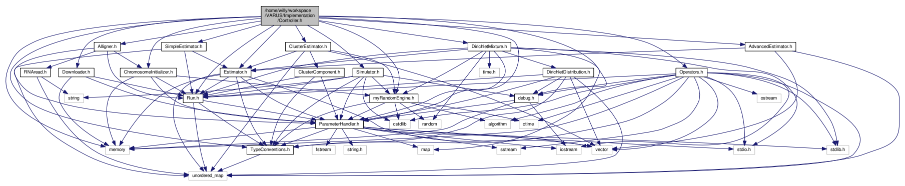
Include dependency graph for Controller.cpp:



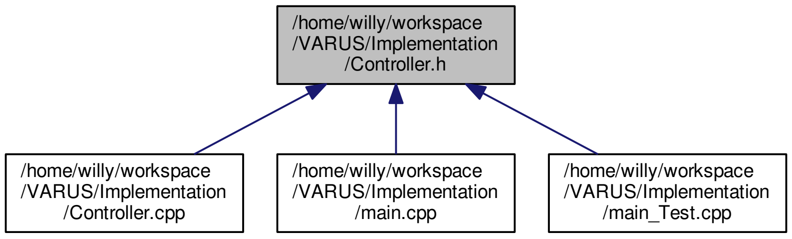
5.12 /home/willy/workspace/VARUS/Implementation/Controller.h File Reference

```
#include "Run.h"
#include "TypeConventions.h"
#include "ChromosomeInitializer.h"
#include "Alligner.h"
#include "Downloader.h"
#include "ParameterHandler.h"
#include "debug.h"
#include "Operators.h"
#include <memory>
#include "myRandomEngine.h"
#include "Estimator.h"
#include "SimpleEstimator.h"
#include "AdvancedEstimator.h"
#include "DirichletMixture.h"
#include "Simulator.h"
#include "ClusterEstimator.h"
```

Include dependency graph for Controller.h:



This graph shows which files directly or indirectly include this file:



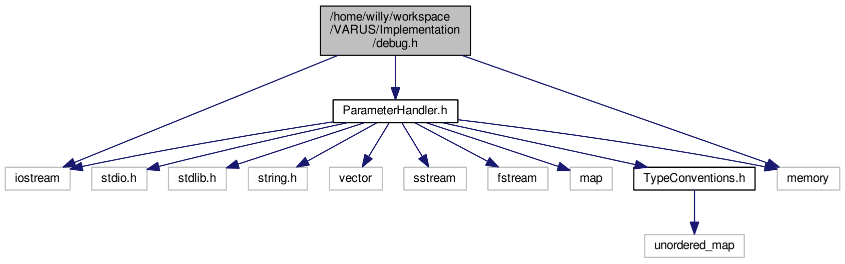
Classes

- class Controller

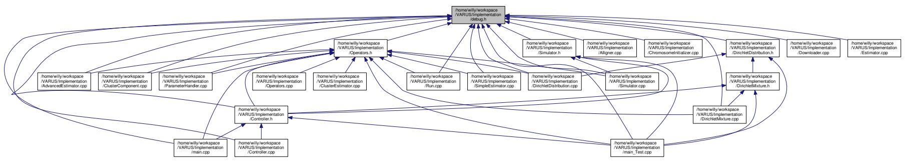
5.13 /home/willy/workspace/VARUS/Implementation/debug.h File Reference

```
#include <iostream>
#include "ParameterHandler.h"
#include <memory>
```

Include dependency graph for debug.h:



This graph shows which files directly or indirectly include this file:



Macros

- #define DEBUG(lvl, msg) if(lvl <= param->verbosityDebug) {std::cerr<<__FILE__ <<':'<<__LINE__-<<';'<<msg; std::cerr<<std::endl;}

Variables

- ParameterHandler ∗ param

    This macro is used for Debuging-purposes.

5.13.1 Macro Definition Documentation

5.13.1.1 #define DEBUG(*lvl,* *msg*) if(lvl <= param->verbosityDebug) {std::cerr<<__FILE__ <<':'<<__LINE__<<';'<<msg; std::cerr<<std::endl;}

5.13.2 Variable Documentation

5.13.2.1 ParameterHandler∗ param

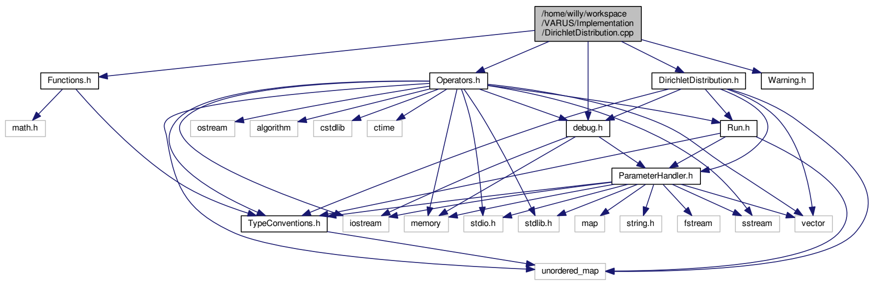
This macro is used for Debuging-purposes.

A lvl can be passed alongside a string. The message is only printed if the lvl is lower or equal to a globaly set verbosity-lvl specified int param->verbosityDebug. Only Classes that have a ParameterHandler-object called param can used this macro.

5.14 /home/willy/workspace/VARUS/Implementation/DirichletDistribution.cpp File Reference

```
#include "DirichletDistribution.h"
#include "Operators.h"
#include "debug.h"
#include "Functions.h"
#include "Warning.h"
```

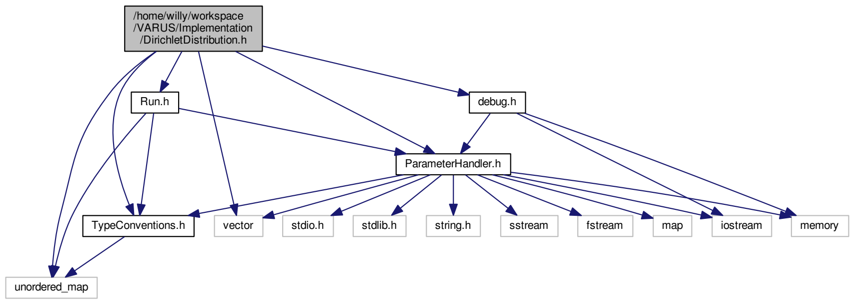
Include dependency graph for DirichletDistribution.cpp:



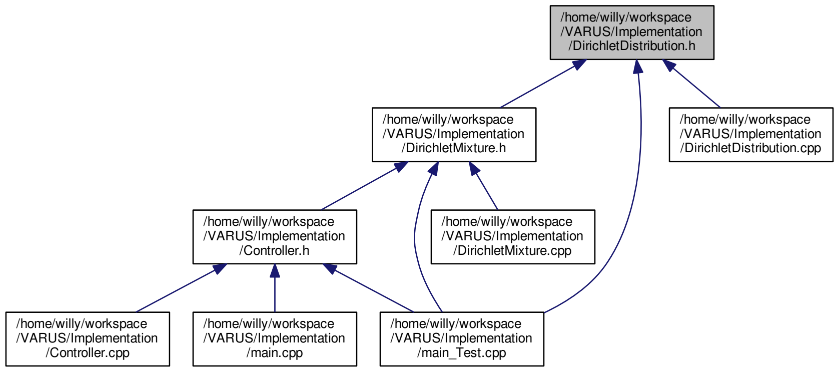
5.15 /home/willy/workspace/VARUS/Implementation/DirichletDistribution.h File Reference

```
#include <unordered_map>
#include <vector>
#include "Run.h"
#include "TypeConventions.h"
#include "ParameterHandler.h"
#include "debug.h"
```

Include dependency graph for DirichletDistribution.h:



This graph shows which files directly or indirectly include this file:



/home/willy/workspace /VARUS/Implementation /DirichletDistribution.h
/home/willy/workspace /VARUS/Implementation /DirichletMixture.h
/home/willy/workspace /VARUS/Implementation /DirichletDistribution.cpp
/home/willy/workspace /VARUS/Implementation /Controller.h
/home/willy/workspace /VARUS/Implementation /DirichletMixture.cpp
/home/willy/workspace /VARUS/Implementation /Controller.cpp
/home/willy/workspace /VARUS/Implementation /main.cpp
/home/willy/workspace /VARUS/Implementation /main_Test.cpp

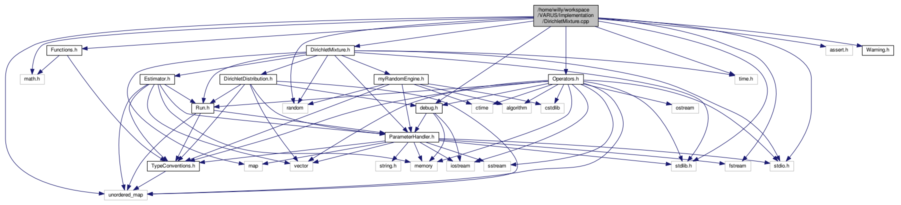
Classes

- class DirichletDistribution

5.16 /home/willy/workspace/VARUS/Implementation/DirichletMixture.cpp File Reference

```
#include "DirichletMixture.h"
#include <math.h>
#include "debug.h"
#include "Operators.h"
#include <unordered_map>
#include <fstream>
#include <random>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <assert.h>
#include "Functions.h"
#include "Warning.h"
```

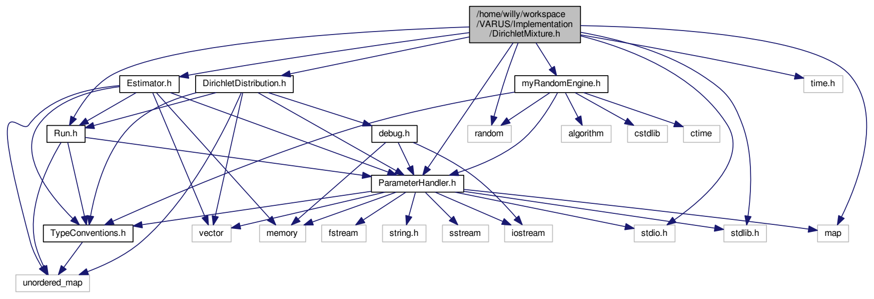
Include dependency graph for DirichletMixture.cpp:



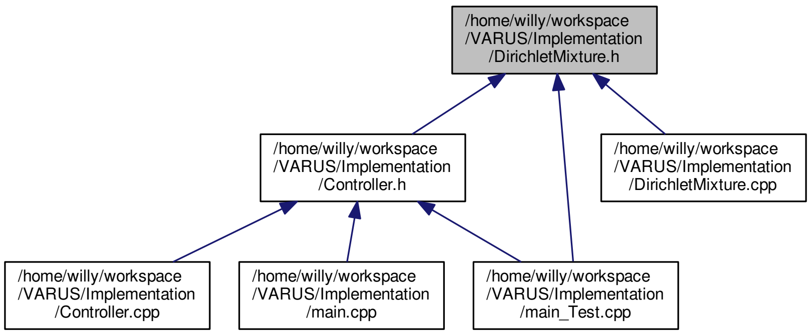
5.17 /home/willy/workspace/VARUS/Implementation/DirichletMixture.h File Reference

```
#include "DirichletDistribution.h"
#include "Estimator.h"
#include "Run.h"
#include "ParameterHandler.h"
#include <map>
#include <random>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include "myRandomEngine.h"
```

Include dependency graph for DirichletMixture.h:



This graph shows which files directly or indirectly include this file:



Classes

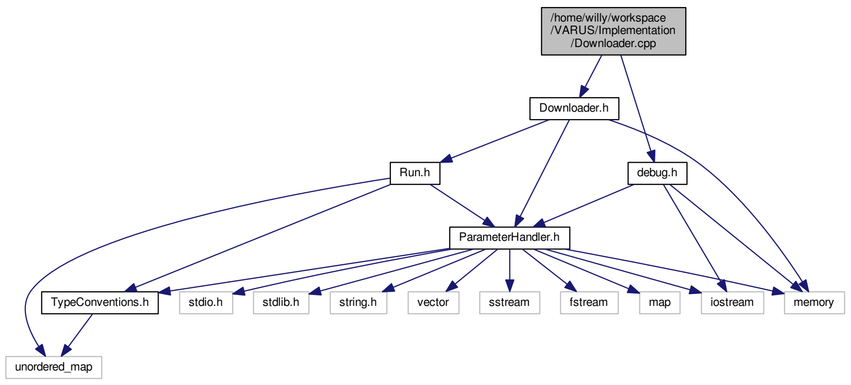
- class DirichletMixture

  The dirichlet-mixture is used, when it is assumed that the runs are not independently distributed, but instead a handfull of distributions represent all runs.

5.18 /home/willy/workspace/VARUS/Implementation/Downloader.cpp File Reference

```
#include "Downloader.h"
#include "debug.h"
```

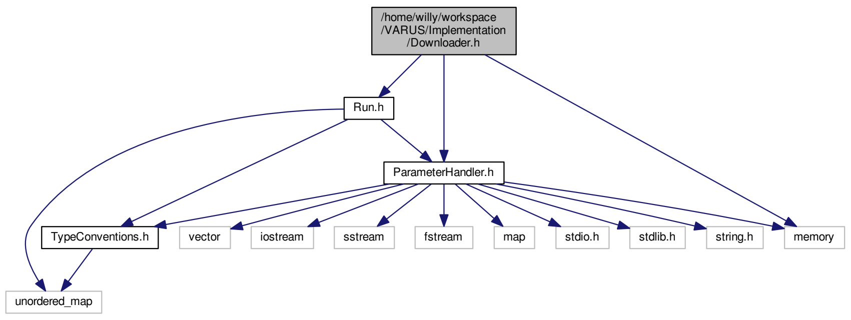
Include dependency graph for Downloader.cpp:



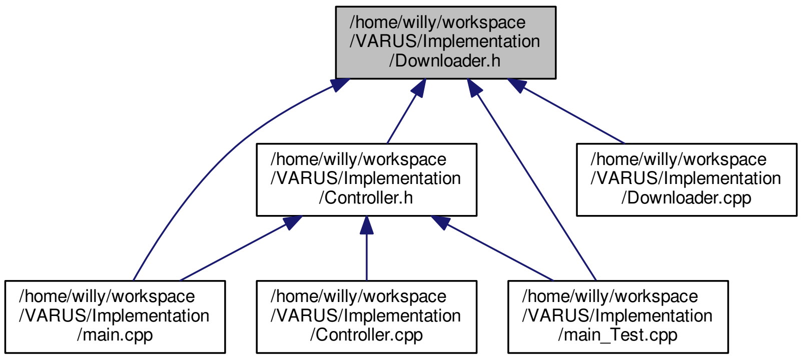
5.19 /home/willy/workspace/VARUS/Implementation/Downloader.h File Reference

```
#include "Run.h"
#include <memory>
#include "ParameterHandler.h"
```

Include dependency graph for Downloader.h:



This graph shows which files directly or indirectly include this file:


/home/willy/workspace /VARUS/Implementation /Downloader.h
/home/willy/workspace /VARUS/Implementation /Controller.h
/home/willy/workspace /VARUS/Implementation /Downloader.cpp
/home/willy/workspace /VARUS/Implementation /main.cpp
/home/willy/workspace /VARUS/Implementation /Controller.cpp
/home/willy/workspace /VARUS/Implementation /main_Test.cpp

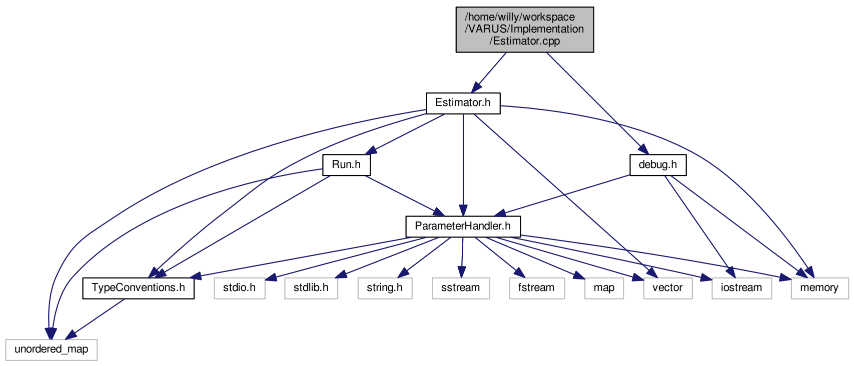
Classes

- class Downloader

5.20 /home/willy/workspace/VARUS/Implementation/Estimator.cpp File Reference

```
#include "Estimator.h"
#include "debug.h"
```

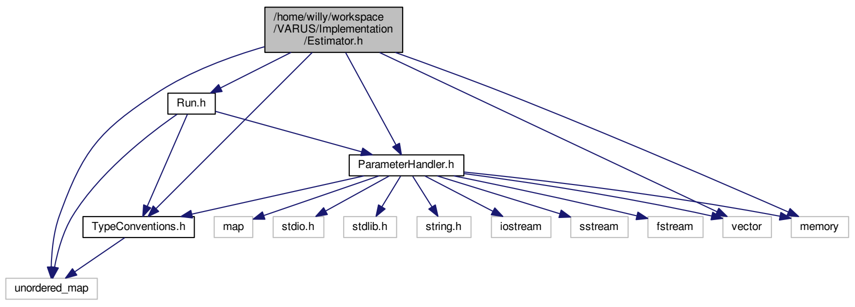
Include dependency graph for Estimator.cpp:



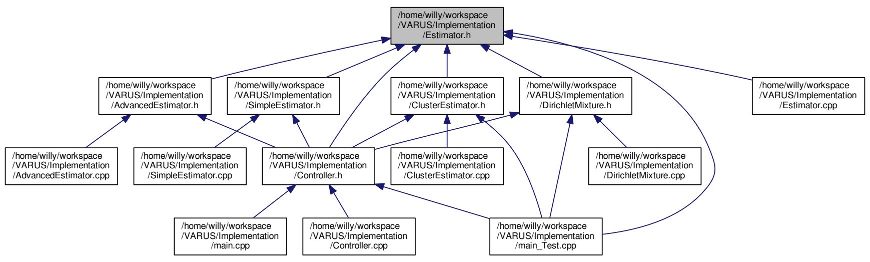
5.21 /home/willy/workspace/VARUS/Implementation/Estimator.h File Reference

```
#include <unordered_map>
#include "Run.h"
#include "ParameterHandler.h"
#include <vector>
#include <memory>
#include "TypeConventions.h"
```

Include dependency graph for Estimator.h:



This graph shows which files directly or indirectly include this file:


/home/willy/workspace/VARUS/Implementation/Estimator.h
/home/willy/workspace/VARUS/Implementation/AdvancedEstimator.h
/home/willy/workspace/VARUS/Implementation/SimpleEstimator.h
/home/willy/workspace/VARUS/Implementation/ClusterEstimator.h
/home/willy/workspace/VARUS/Implementation/DirichletMixture.h
/home/willy/workspace/VARUS/Implementation/Estimator.cpp
/home/willy/workspace/VARUS/Implementation/AdvancedEstimator.cpp
/home/willy/workspace/VARUS/Implementation/SimpleEstimator.cpp
/home/willy/workspace/VARUS/Implementation/Controller.h
/home/willy/workspace/VARUS/Implementation/ClusterEstimator.cpp
/home/willy/workspace/VARUS/Implementation/DirichletMixture.cpp
/home/willy/workspace/VARUS/Implementation/main.cpp
/home/willy/workspace/VARUS/Implementation/Controller.cpp
/home/willy/workspace/VARUS/Implementation/main_Test.cpp

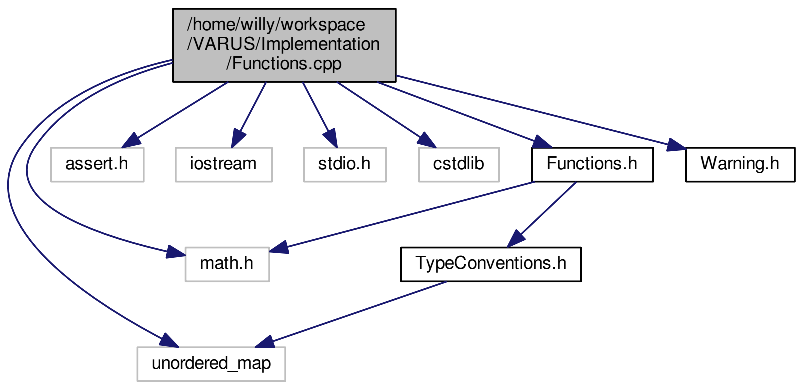
Classes

- class Estimator

5.22 /home/willy/workspace/VARUS/Implementation/Functions.cpp File Reference

```
#include <unordered_map>
#include <math.h>
#include <assert.h>
#include <iostream>
#include <stdio.h>
#include <cstdlib>
#include "Functions.h"
#include "Warning.h"
```

Include dependency graph for Functions.cpp:



Functions

- double digamma (const double &x)
- double trigamma (const double &x)
- double Emean (UUmap &c)
- double Evariance (UUmap &c)
- double Vj (UUmap &cj, UUmap &c, const double qj)
- double newtonsMethod (double start, double(∗f)(double), double(∗df)(double))

5.22.1 Function Documentation

5.22.1.1 double digamma (const double & *x*)

Implementation of the digamma-function.

ALGORITHM AS 103 APPL. STATIST. (1976) VOL.25, NO.3

5.22.1.2 double Emean (UUmap & *c*)

Calculates the artihmetic mean of the values in c.

5.22.1.3 double Evariance (UUmap & *c*)

Calculates the variance in the values of c.

5.22.1.4 double newtonsMethod (double *start,* double(∗)(double) *f,* double(∗)(double) *df*)

5.22.1.5 double trigamma (const double & *x*)

Implementation of the trigamma-function.

algorithm as121 Appl. Statist. (1978) vol 27, no. 1

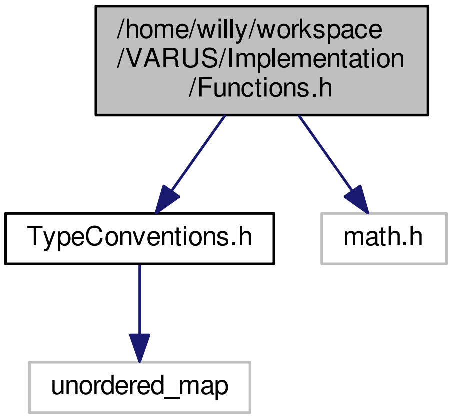
5.22.1.6 double Vj (**UUmap** & *cj*, **UUmap** & *c*, const double *qj*)

Below formula (12) in Altschull.

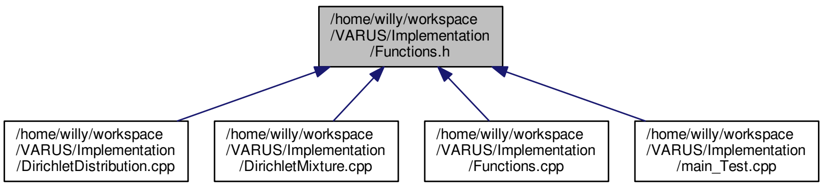
5.23 /home/willy/workspace/VARUS/Implementation/Functions.h File Reference

```
#include "TypeConventions.h"
#include "math.h"
```

Include dependency graph for Functions.h:



This graph shows which files directly or indirectly include this file:



Functions

- double digamma (const double &x)
- double trigamma (const double &x)
- double Emean (UUmap &c)
- double Evariance (UUmap &c)
- double Vj (UUmap &cj, UUmap &c, const double qj)
- double newtonsMethod (double start, double(∗f)(double), double(∗df)(double))

5.23.1 Function Documentation

5.23.1.1 double digamma (const double & *x*)

Implementation of the digamma-function.

ALGORITHM AS 103 APPL. STATIST. (1976) VOL.25, NO.3

5.23.1.2 double Emean (UUmap & *c*)

Calculates the artihmetic mean of the values in c.

5.23.1.3 double Evariance (UUmap & *c*)

Calculates the variance in the values of c.

5.23.1.4 double newtonsMethod (double *start,* double(∗)(double) *f,* double(∗)(double) *df*)

5.23.1.5 double trigamma (const double & *x*)

Implementation of the trigamma-function.

algorithm as121 Appl. Statist. (1978) vol 27, no. 1

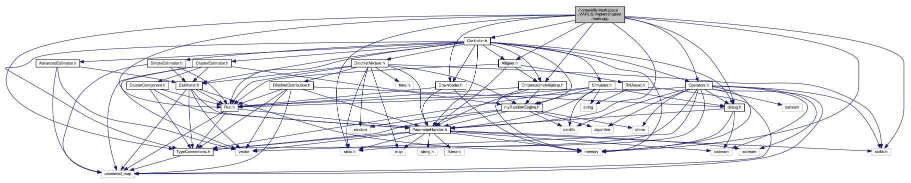
5.23.1.6 double Vj (UUmap & *cj,* UUmap & *c,* const double *qj*)

Below formula (12) in Altschull.

5.24 /home/willy/workspace/VARUS/Implementation/main.cpp File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include "Run.h"
#include "TypeConventions.h"
#include "ChromosomeInitializer.h"
#include "Alligner.h"
#include "Downloader.h"
#include "ParameterHandler.h"
#include "debug.h"
#include "Operators.h"
#include <memory>
#include "Controller.h"
```

Include dependency graph for main.cpp:



Functions

- int main (int argc, char ∗argv[])

Variables

- ParameterHandler ∗ param

  This macro is used for Debuging-purposes.

5.24.1 Function Documentation

5.24.1.1 int main (int *argc*, char ∗ *argv[]*)

5.24.2 Variable Documentation

5.24.2.1 ParameterHandler∗ param

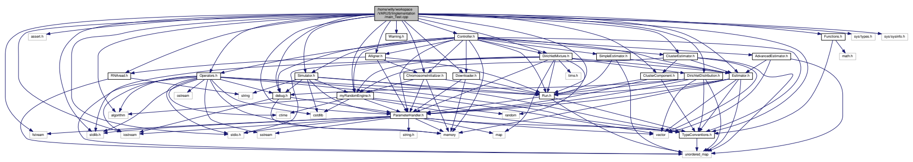
This macro is used for Debuging-purposes.

A lvl can be passed alongside a string. The message is only printed if the lvl is lower or equal to a globaly set verbosity-lvl specified int param->verbosityDebug. Only Classes that have a ParameterHandler-object called param can used this macro.

5.25 /home/willy/workspace/VARUS/Implementation/main_Test.cpp File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include <iostream>
#include <fstream>
#include "ChromosomeInitializer.h"
#include "TypeConventions.h"
#include "debug.h"
#include "Operators.h"
#include "ParameterHandler.h"
#include "Warning.h"
#include "Run.h"
#include "Downloader.h"
#include "Alligner.h"
#include "RNAread.h"
#include "Controller.h"
#include "myRandomEngine.h"
#include <algorithm>
#include "Estimator.h"
#include "Simulator.h"
#include "Functions.h"
#include "DirichletDistribution.h"
#include "DirichletMixture.h"
#include "ClusterComponent.h"
#include "ClusterEstimator.h"
#include "sys/types.h"
#include "sys/sysinfo.h"
```

Include dependency graph for main_Test.cpp:



Macros

- #define ASSRT(val)

Functions

- void printToFile (string path, string content)
- string read_file (string path)
- void test_map_plus ()
- void test_map_plus_equal ()
- void test_initMap ()
- void test_DotProduct ()
- void test_ScaleMap ()
- void test_operators ()
- void test_myRandomEnginge_vectorSelect ()
- void test_myRandomEngine_shuffle ()
- void test_myRandomEngine_selectRandomly_p ()
- void test_myRandomEngine ()
- void test_ParameterHandler_read_parameters ()
- void test_ParameterHandler_commandPrompt (string commandPrompt)
- void test_ParameterHandler_start_script (std::string path)
- void test_ParameterHandler ()
- void test_Run ()
- void test_ChromosomeInitializer_readInputRuns ()
- void test_ChromosomeInitializer_getChromosomeLengths ()
- void test_ChromosomeInitializer_create_transcript_units ()
- void test_ChromosomeInitializer_initializeSigma ()
- void test_ChromosomeInitializer_initializeRuns ()
- void test_ChromosomeInitializer ()
- void test_Downloader_nextBatchIndices ()
- void test_Downloader_shellCommand ()
- void test_Downloader_getBatch ()
- void test_Downloader ()
- void test_RNAread_UMR ()
- void test_RNAread ()
- void test_Alligner_shellCommand ()
- void test_Alligner_mapReads ()
- void test_Alligner_getAllignedReads ()
- void test_Alligner_updateObservations ()
- void test_Alligner_update ()
- void test_Alligner ()
- void test_Controller_allgorithm ()
- void test_Controller_allgorithm2 ()
- void test_Controller_allgorithmSimulation ()

- void test_Controller_runsEmpty ()
- void test_Controller_createDice ()
- void test_Controller_exitCondition ()
- void test_Controller ()
- void test_DirichletDistribution_const ()
- void test_DirichletDistribution_reset ()
- void test_DirichletDistribution_calculateMixtureParameters ()
- string test_DirichletDistribution_Mixtures2runs2dim (int x_1, int x_2, int y_1, int y_2, string path, bool print=false)
- void test_DirichletDistribution_MixturesNruns_n_dim (vector< UUmap > &obs, bool print=false)
- void test_DirichletDistribution_Mixtures ()
- void test_DirichletDistribution_EalphaSum ()
- void test_DirichletDistribution_EalphaSum2 ()
- void test_DirichletDistribution ()
- void test_DirichletMixture_train ()
- void test_DirichletMixture_calculateLikelihood ()
- void test_DirichletMixture_calculateP ()
- void test_DirichletMixture ()
- void test_simpleEstimator1Run (UUmap obs, double pseudoCount, string solution, bool shouldPrint=false)
- void test_simpleEstimator ()
- void test_AdvancedEstimator2Run (UUmap obs, UUmap obs2, double pseudoCount, double lambda, string solution, bool shouldPrint=false)
- void test_AdvancedEstimator ()
- void test_Estimators ()
- void test_Simulator_readInputRuns ()
- void test_Simulator_initializeRuns ()
- void test_Simulator_simulateObservations ()
- void test_Simulator ()
- void test_Digamma ()
- void test_Trigamma ()
- void test_Emean ()
- void test_Evariance ()
- void test_Functions ()
- void test_ClusterEstimator_const ()
- void test_ClusterRemoveRuns ()
- void test_ClusterEstimator_calculateP ()
- void test_ClusterEstimator_estimateP ()
- void test_ClusterEstimator_calculateVariance ()
- void test_ClusterEstimator ()
- void test_MapSize ()
- int parseLine (char ∗line)
- int getValue ()
- void RunCrashTest ()
- int main (int argc, char ∗argv[])

Variables

- ParameterHandler ∗ param = new ParameterHandler()

  This macro is used for Debuging-purposes.
- unsigned int error_count = 0
- struct sysinfo memInfo

5.25.1 Macro Definition Documentation

5.25.1.1 #define ASSRT(*val*)

Value:

```
if (val == false) {                    \
    cerr << "->\terror in " << __FUNCTION__ << "()"<< endl; \
    error_count++;                                      \
}                                                       \
```

5.25.2 Function Documentation

5.25.2.1 int getValue ( )

5.25.2.2 int main (int *argc,* char ∗ *argv[ ]*)

5.25.2.3 int parseLine (char ∗ *line*)

5.25.2.4 void printToFile (string *path,* string *content*)

5.25.2.5 string read_file (string *path*)

5.25.2.6 void RunCrashTest ( )

5.25.2.7 void test_AdvancedEstimator ( )

5.25.2.8 void test_AdvancedEstimator2Run (UUmap *obs,* UUmap *obs2,* double *pseudoCount,* double *lambda,* string *solution,* bool *shouldPrint* = `false`)

5.25.2.9 void test_Alligner ( )

5.25.2.10 void test_Alligner_getAllignedReads ( )

5.25.2.11 void test_Alligner_mapReads ( )

5.25.2.12 void test_Alligner_shellCommand ( )

5.25.2.13 void test_Alligner_update ( )

5.25.2.14 void test_Alligner_updateObservations ( )

5.25.2.15 void test_ChromosomeInitializer ( )

5.25.2.16 void test_ChromosomeInitializer_create_transcript_units ( )

5.25.2.17 void test_ChromosomeInitializer_getChromosomeLengths ( )

5.25.2.18 void test_ChromosomeInitializer_initializeRuns ( )

5.25.2.19 void test_ChromosomeInitializer_initializeSigma ( )

5.25.2.20 void test_ChromosomeInitializer_readInputRuns ( )

5.25.2.21 void test_ClusterEstimator ( )

5.25.2.22 void test_ClusterEstimator_calculateP ( )

5.25.2.23 void test_ClusterEstimator_calculateVariance ( )

5.25.2.24 void test_ClusterEstimator_const ( )

5.25.2.25 void test_ClusterEstimator_estimateP ( )

5.25.2.26 void test_ClusterRemoveRuns ( )

5.25.2.27 void test_Controller ( )

5.25.2.28 void test_Controller_allgorithm ( )

5.25.2.29 void test_Controller_allgorithm2 ( )

5.25.2.30 void test_Controller_allgorithmSimulation ( )

5.25.2.31 void test_Controller_createDice ( )

5.25.2.32 void test_Controller_exitCondition ( )

5.25.2.33 void test_Controller_runsEmpty ( )

5.25.2.34 void test_Digamma ( )

5.25.2.35 void test_DirichletDistribution ( )

5.25.2.36 void test_DirichletDistribution_calculateMixtureParameters ( )

5.25.2.37 void test_DirichletDistribution_const ( )

5.25.2.38 void test_DirichletDistribution_EalphaSum ( )

5.25.2.39 void test_DirichletDistribution_EalphaSum2 ( )

5.25.2.40 void test_DirichletDistribution_Mixtures ( )

5.25.2.41 string test_DirichletDistribution_Mixtures2runs2dim (int *x_1*, int *x_2*, int *y_1*, int *y_2*, string *path*, bool *print* = `false`)

5.25.2.42 void test_DirichletDistribution_MixturesNruns_n_dim (vector< UUmap > & *obs*, bool *print* = `false`)

5.25.2.43 void test_DirichletDistribution_reset ( )

5.25.2.44 void test_DirichletMixture ( )

5.25.2.45 void test_DirichletMixture_calculateLikelihood ( )

5.25.2.46 void test_DirichletMixture_calculateP ( )

5.25.2.47 void test_DirichletMixture_train ( )

5.25.2.48 void test_DotProduct ( )

5.25.2.49 void test_Downloader ( )

5.25.2.50 void test_Downloader_getBatch ( )

5.25.2.51 void test_Downloader_nextBatchIndices ( )

5.25.2.52 void test_Downloader_shellCommand ( )

5.25.2.53 void test_Emean ( )

5.25.2.54 void test_Estimators ( )

5.25.2.55 void test_Evariance ( )

5.25.2.56 void test_Functions ( )

5.25.2.57 void test_initMap ( )

5.25.2.58 void test_map_plus ( )

5.25.2.59 void test_map_plus_equal ( )

5.25.2.60 void test_MapSize ( )

5.25.2.61 void test_myRandomEngine ( )

5.25.2.62 void test_myRandomEngine_selectRandomly_p ( )

5.25.2.63 void test_myRandomEngine_shuffle ( )

5.25.2.64 void test_myRandomEnginge_vectorSelect ( )

5.25.2.65 void test_operators ( )

5.25.2.66 void test_ParameterHandler ( )

5.25.2.67 void test_ParameterHandler_commandPrompt (string *commandPrompt*)

5.25.2.68 void test_ParameterHandler_read_parameters ( )

5.25.2.69 void test_ParameterHandler_start_script (std::string *path*)

5.25.2.70 void test_RNAread ( )

5.25.2.71 void test_RNAread_UMR ( )

5.25.2.72 void test_Run ( )

5.25.2.73 void test_ScaleMap ( )

5.25.2.74 void test_simpleEstimator ( )

5.25.2.75 void test_simpleEstimator1Run (UUmap *obs,* double *pseudoCount,* string *solution,* bool *shouldPrint =* `false`)

5.25.2.76 void test_Simulator ( )

5.25.2.77 void test_Simulator_initializeRuns ( )

5.25.2.78 void test_Simulator_readInputRuns ( )

5.25.2.79 void test_Simulator_simulateObservations ( )

5.25.2.80 void test_Trigamma ( )

5.25.3 Variable Documentation

5.25.3.1 unsigned int error_count = 0

5.25.3.2 struct sysinfo memInfo

5.25.3.3 ParameterHandler∗ param = new ParameterHandler()

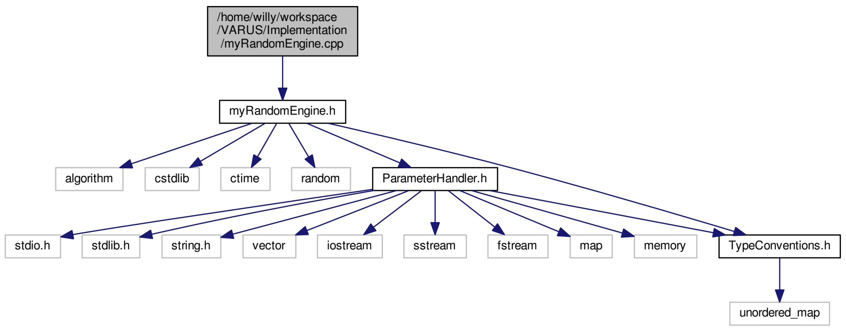
This macro is used for Debuging-purposes.

A lvl can be passed alongside a string. The message is only printed if the lvl is lower or equal to a globaly set verbosity-lvl specified int param->verbosityDebug. Only Classes that have a ParameterHandler-object called param can used this macro.

5.26 /home/willy/workspace/VARUS/Implementation/myRandomEngine.cpp File Reference

```
#include "myRandomEngine.h"
```

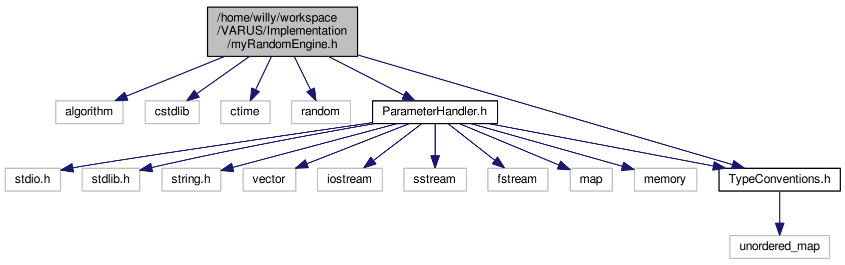
Include dependency graph for myRandomEngine.cpp:



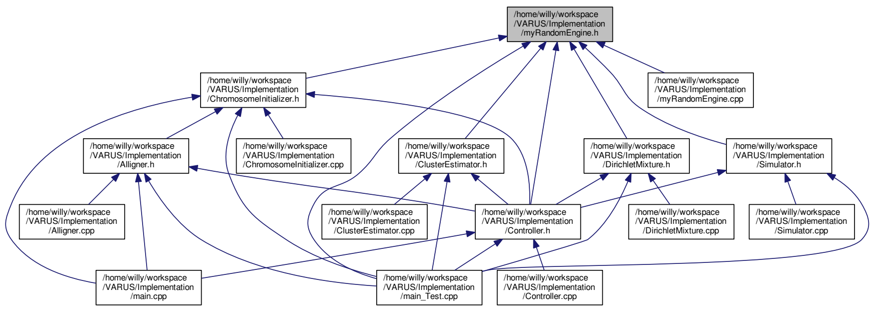
5.27 /home/willy/workspace/VARUS/Implementation/myRandomEngine.h File Reference

```
#include <algorithm>
#include <cstdlib>
#include <ctime>
#include <random>
#include "ParameterHandler.h"
#include "TypeConventions.h"
```

Include dependency graph for myRandomEngine.h:



This graph shows which files directly or indirectly include this file:


/home/willy/workspace /VARUS/Implementation /myRandomEngine.h
/home/willy/workspace /VARUS/Implementation /ChromosomeInitializer.h
/home/willy/workspace /VARUS/Implementation /myRandomEngine.cpp
/home/willy/workspace /VARUS/Implementation /Alligner.h
/home/willy/workspace /VARUS/Implementation /ChromosomeInitializer.cpp
/home/willy/workspace /VARUS/Implementation /ClusterEstimator.h
/home/willy/workspace /VARUS/Implementation /DirichletMixture.h
/home/willy/workspace /VARUS/Implementation /Simulator.h
/home/willy/workspace /VARUS/Implementation /Alligner.cpp
/home/willy/workspace /VARUS/Implementation /ClusterEstimator.cpp
/home/willy/workspace /VARUS/Implementation /Controller.h
/home/willy/workspace /VARUS/Implementation /DirichletMixture.cpp
/home/willy/workspace /VARUS/Implementation /Simulator.cpp
/home/willy/workspace /VARUS/Implementation /main.cpp
/home/willy/workspace /VARUS/Implementation /main_Test.cpp
/home/willy/workspace /VARUS/Implementation /Controller.cpp

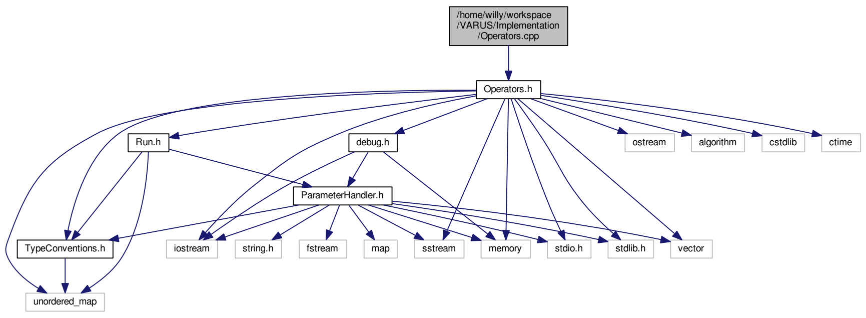
Classes

- class myRandomEngine

5.28 /home/willy/workspace/VARUS/Implementation/Operators.cpp File Reference

```
#include "Operators.h"
```

Include dependency graph for Operators.cpp:



Functions

- void remove_substr (const std::string &sub, std::string &str)
- void scaleMap (UDmap &m, const double scaleTo)

5.28.1 Function Documentation

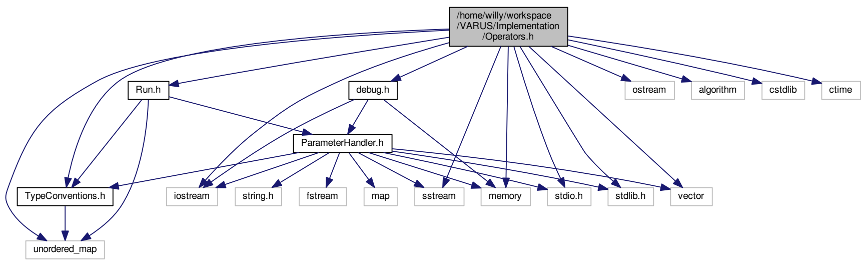
5.28.1.1 void remove_substr (const std::string & *sub*, std::string & *str*)

5.28.1.2 void scaleMap (UDmap & *m*, const double *scaleTo*)

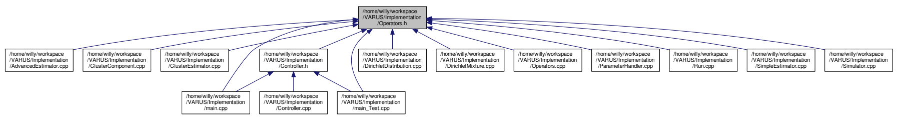
5.29 /home/willy/workspace/VARUS/Implementation/Operators.h File Reference

```
#include <unordered_map>
#include <ostream>
#include <iostream>
#include <sstream>
#include <memory>
#include <stdio.h>
#include <stdlib.h>
#include <vector>
#include "debug.h"
#include "TypeConventions.h"
#include "Run.h"
#include <algorithm>
#include <cstdlib>
#include <ctime>
```

Include dependency graph for Operators.h:



This graph shows which files directly or indirectly include this file:



Functions

- template<typename K , typename V >
  std::ostream & operator<< (std::ostream &os, std::unordered_map< K, V > &map)

- template<typename K , typename V >
  std::stringstream & operator<< (std::stringstream &os, std::unordered_map< K, V > &map)
- template<template< typename...> class Map, typename K , typename V , typename V2 >
  void operator+= (Map< K, V > &m1, const Map< K, V2 > &m2)
- template<template< typename...> class Map, typename K , typename V , typename V2 >
  void operator*= (Map< K, V > &m, const V2 scalar)
- template<template< typename...> class Map, typename K , typename V , typename V2 >
  void operator/= (Map< K, V > &m, const V2 scalar)
- template<template< typename...> class Map, typename K , typename V1 , typename V2 >
  Map< K, V1 > operator+ (Map< K, V1 > &m1, const Map< K, V2 > &m2)
- template<template< typename...> class Map, typename K , typename V , typename V2 >
  Map< K, V > operator/ (const Map< K, V > &m, V2 n)
- template<template< typename...> class Map, typename K , typename V >
  void initMap (Map< K, V > &m, const V a, const K n)
- template<template< typename...> class Map, typename V >
  double euklidNorm (Map< U32, V > &m)
- template<template< typename...> class Map, typename K , typename V >
  V sum (const Map< K, V > &m)
- void scaleMap (UDmap &m, const double scaleTo)
- template<template< typename...> class Map, typename K , typename V >
  void set0 (Map< K, V > &m)
- template<template< typename...> class Map, typename K , typename V >
  void dotProduct (Map< K, V > &m1)
- template<template< typename...> class Map, typename K , typename V >
  std::string toStr (Map< K, V > &m)
- template<typename T >
  unsigned int num_of_digits (T i)
- template<typename K , typename V >
  std::vector< V > map2vec (std::unordered_map< K, V > &m)
- template<typename V >
  std::ostream & operator<< (std::ostream &os, std::vector< V > &v)
- template<typename K , typename V >
  void normMap (std::unordered_map< K, V > &m)
- template<typename V >
  V sum (std::vector< V > &v)
- template<typename V >
  void normVector (std::vector< V > &v)
- void remove_substr (const std::string &sub, std::string &str)

5.29.1 Function Documentation

5.29.1.1 template<template< typename...> class Map, typename K , typename V > void dotProduct (Map< K, V > & *m1*)

5.29.1.2 template<template< typename...> class Map, typename V > double euklidNorm (Map< U32, V > & *m*)

5.29.1.3 template<template< typename...> class Map, typename K , typename V > void initMap (Map< K, V > & *m*, const V *a*, const K *n*)

5.29.1.4 template<typename K , typename V > std::vector<V> map2vec (std::unordered_map< K, V > & *m*)

5.29.1.5 template<typename K , typename V > void normMap (std::unordered_map< K, V > & *m*)

5.29.1.6 template<typename V > void normVector (std::vector< V > & *v*)

5.29.1.7 template<typename T > unsigned int num_of_digits (T *i*)

5.29.1.8 template<template< typename...> class Map, typename K , typename V , typename V2 > void operator*= (Map< K, V > & *m*, const V2 *scalar*)

5.29.1.9 template<template< typename...> class Map, typename K , typename V1 , typename V2 > Map<K, V1> operator+ (Map< K, V1 > & *m1*, const Map< K, V2 > & *m2*)

5.29.1.10 template<template< typename...> class Map, typename K , typename V , typename V2 > void operator+= (Map< K, V > & *m1*, const Map< K, V2 > & *m2*)

5.29.1.11 template<template< typename...> class Map, typename K , typename V , typename V2 > Map<K,V> operator/ (const Map< K, V > & *m*, V2 *n*)

5.29.1.12 template<template< typename...> class Map, typename K , typename V , typename V2 > void operator/= (Map< K, V > & *m*, const V2 *scalar*)

5.29.1.13 template<typename K , typename V > std::ostream& operator<< (std::ostream & *os*, std::unordered_map< K, V > & *map*)

5.29.1.14 template<typename K , typename V > std::stringstream& operator<< (std::stringstream & *os*, std::unordered_map< K, V > & *map*)

5.29.1.15 template<typename V > std::ostream& operator<< (std::ostream & *os*, std::vector< V > & *v*)

5.29.1.16 void remove_substr (const std::string & *sub*, std::string & *str*)

5.29.1.17 void scaleMap (UDmap & *m*, const double *scaleTo*)

5.29.1.18 template<template< typename...> class Map, typename K , typename V > void set0 (Map< K, V > & *m*)

5.29.1.19 template<template< typename...> class Map, typename K , typename V > V sum (const Map< K, V > & *m*)

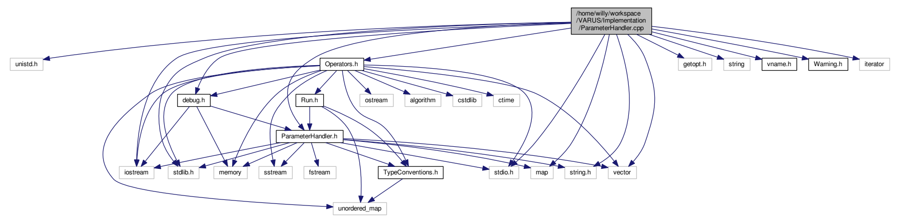
5.29.1.20 template<typename V > V sum (std::vector< V > & *v*)

5.29.1.21 template<template< typename...> class Map, typename K , typename V > std::string toStr (Map< K, V > & *m*)

5.30 /home/willy/workspace/VARUS/Implementation/ParameterHandler.cpp File Reference

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <getopt.h>
#include <string>
#include <iostream>
#include <map>
#include <vector>
#include "vname.h"
#include "debug.h"
#include "Warning.h"
#include "ParameterHandler.h"
#include "Operators.h"
#include <iterator>
#include <string.h>
```

Include dependency graph for ParameterHandler.cpp:



Macros

- #define PARAM(x)
- #define INITPARAM(x, use)

Variables

- int getoptCount = 0

5.30.1 Macro Definition Documentation

5.30.1.1 #define INITPARAM(*x*, *use*)

Value:

```
{           \
        string name = #x;    \
    add_parameter(name, x); \
    usage[name] = use;      \
}
```

5.30.1.2 #define PARAM(*x*)

Value:

```
{           \
        string name = #x;    \
    add_parameter(name, x); \
}
```

5.30.2 Variable Documentation

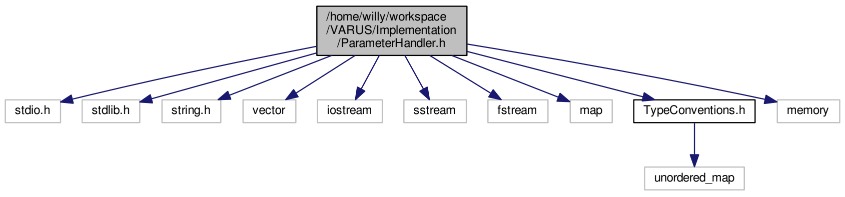
5.30.2.1 int getoptCount = 0

5.31 /home/willy/workspace/VARUS/Implementation/ParameterHandler.h File Reference

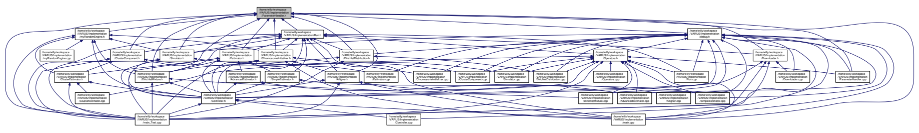
```
#include <stdio.h>
```

```
#include <stdlib.h>
#include <string.h>
#include <vector>
#include <iostream>
#include <sstream>
#include <fstream>
#include <map>
#include "TypeConventions.h"
#include <memory>
```

Include dependency graph for ParameterHandler.h:



This graph shows which files directly or indirectly include this file:



Classes

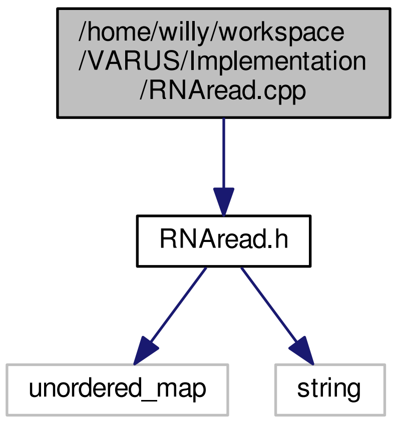
- class ParameterHandler

This class holds all the data, that is read in as command-line arguments.

5.32 /home/willy/workspace/VARUS/Implementation/RNAread.cpp File Reference

```
#include "RNAread.h"
```

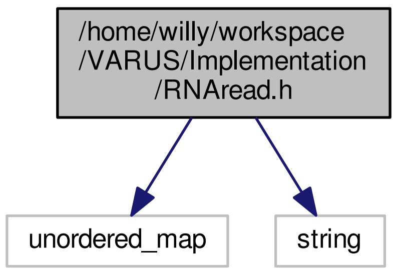
Include dependency graph for RNAread.cpp:



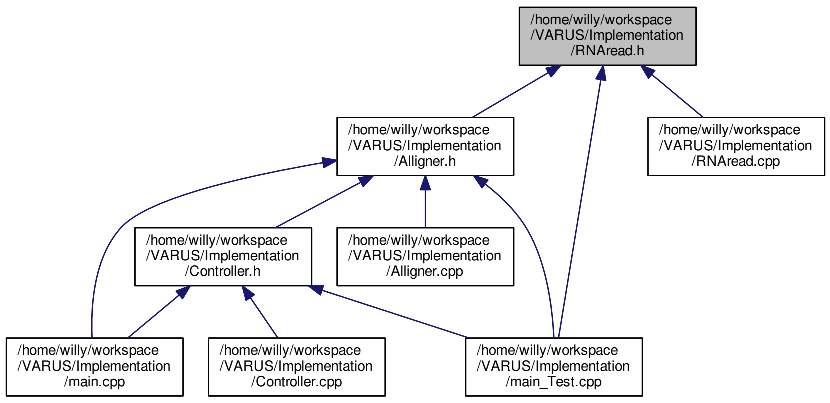
5.33 /home/willy/workspace/VARUS/Implementation/RNAread.h File Reference

```
#include <unordered_map>
#include <string>
```

Include dependency graph for RNAread.h:



This graph shows which files directly or indirectly include this file:


/home/willy/workspace /VARUS/Implementation /RNAread.h
/home/willy/workspace /VARUS/Implementation /Alligner.h
/home/willy/workspace /VARUS/Implementation /RNAread.cpp
/home/willy/workspace /VARUS/Implementation /Controller.h
/home/willy/workspace /VARUS/Implementation /Alligner.cpp
/home/willy/workspace /VARUS/Implementation /main.cpp
/home/willy/workspace /VARUS/Implementation /Controller.cpp
/home/willy/workspace /VARUS/Implementation /main_Test.cpp

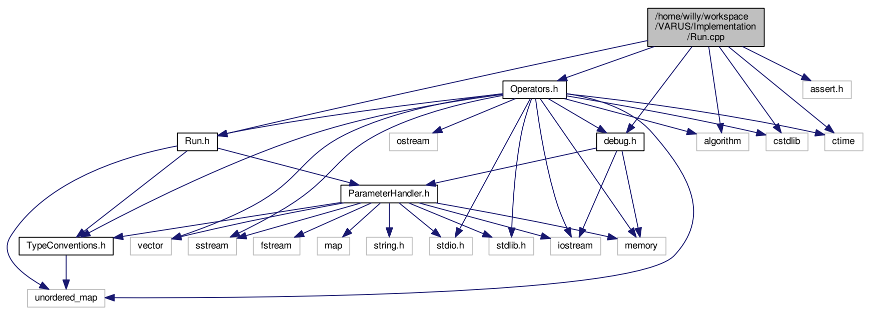
Classes

- class RNAread

5.34 /home/willy/workspace/VARUS/Implementation/Run.cpp File Reference

```
#include "Run.h"
#include "Operators.h"
#include "debug.h"
#include <algorithm>
#include <cstdlib>
#include <ctime>
#include <assert.h>
```

Include dependency graph for Run.cpp:



Variables

- ParameterHandler ∗ param

    This macro is used for Debuging-purposes.

5.34.1 Variable Documentation

5.34.1.1 ParameterHandler∗ param

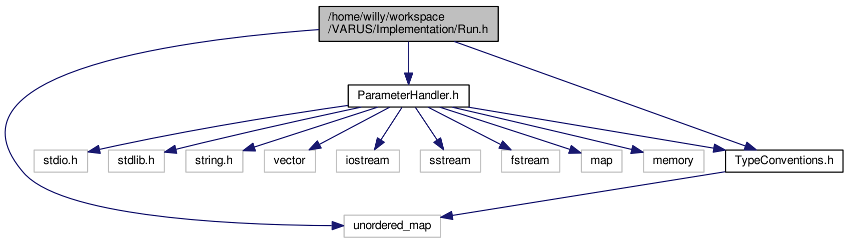
This macro is used for Debuging-purposes.

A lvl can be passed alongside a string. The message is only printed if the lvl is lower or equal to a globaly set verbosity-lvl specified int param->verbosityDebug. Only Classes that have a ParameterHandler-object called param can used this macro.

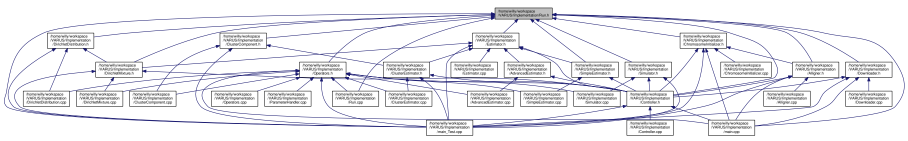
5.35 /home/willy/workspace/VARUS/Implementation/Run.h File Reference

```
#include <unordered_map>
#include "ParameterHandler.h"
#include "TypeConventions.h"
```

Include dependency graph for Run.h:



This graph shows which files directly or indirectly include this file:



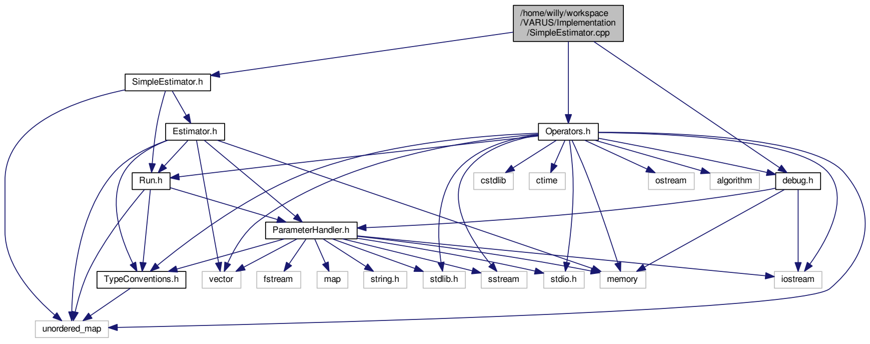
Classes

- class Run

5.36 /home/willy/workspace/VARUS/Implementation/SimpleEstimator.cpp File Reference

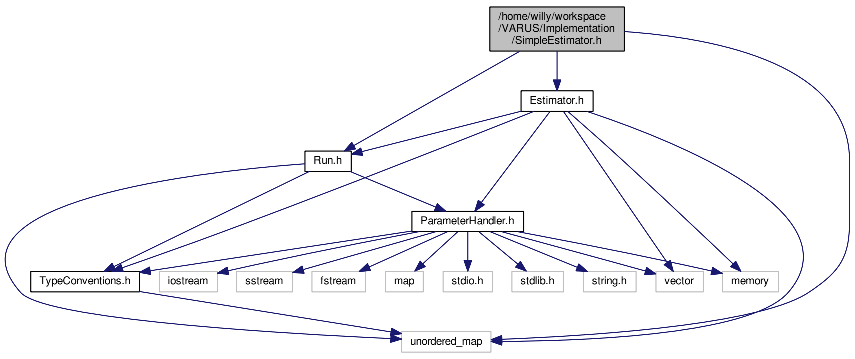
```
#include "SimpleEstimator.h"
#include "debug.h"
#include "Operators.h"
```

Include dependency graph for SimpleEstimator.cpp:

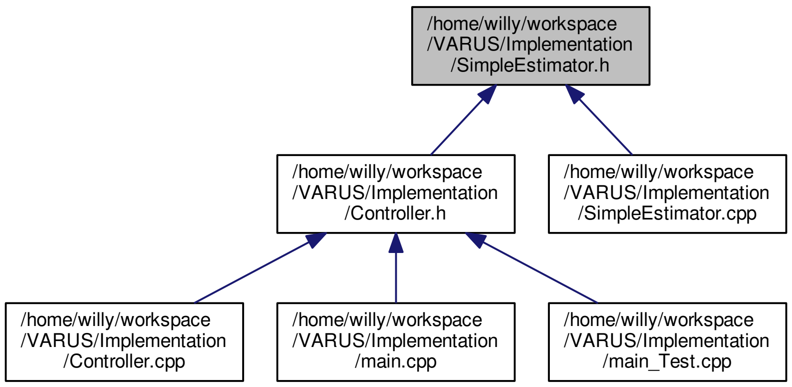


5.37 /home/willy/workspace/VARUS/Implementation/SimpleEstimator.h File Reference

```
#include "Estimator.h"
#include "Run.h"
#include <unordered_map>
```
Include dependency graph for SimpleEstimator.h:



This graph shows which files directly or indirectly include this file:


/home/willy/workspace /VARUS/Implementation /SimpleEstimator.h
/home/willy/workspace /VARUS/Implementation /Controller.h
/home/willy/workspace /VARUS/Implementation /SimpleEstimator.cpp
/home/willy/workspace /VARUS/Implementation /Controller.cpp
/home/willy/workspace /VARUS/Implementation /main.cpp
/home/willy/workspace /VARUS/Implementation /main_Test.cpp

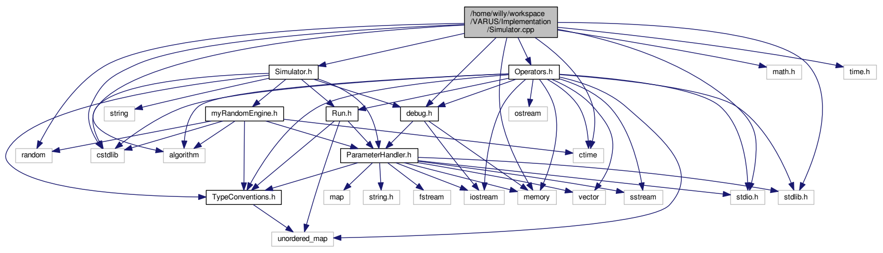
Classes

- class SimpleEstimator

5.38 /home/willy/workspace/VARUS/Implementation/Simulator.cpp File Reference

```
#include "Simulator.h"
#include "debug.h"
#include <math.h>
#include <memory>
#include <algorithm>
#include <cstdlib>
#include <ctime>
#include "Operators.h"
#include <random>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
```

Include dependency graph for Simulator.cpp:

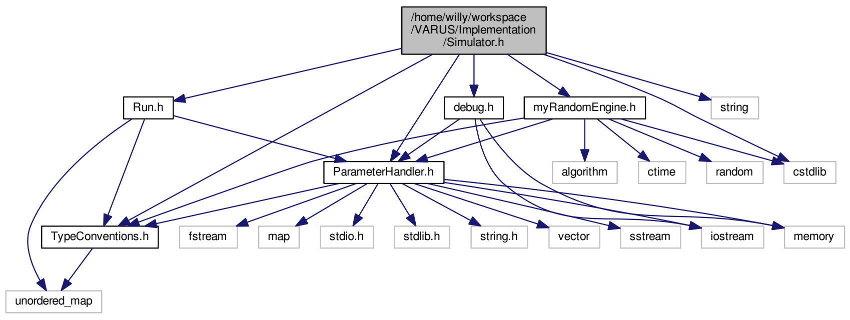


5.39 /home/willy/workspace/VARUS/Implementation/Simulator.h File Reference

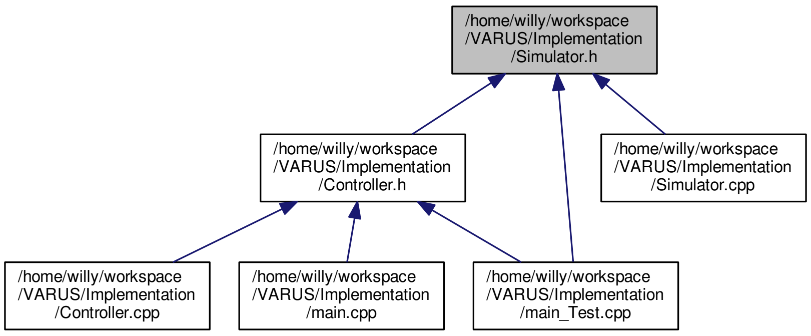
```
#include "TypeConventions.h"
```

```
#include "ParameterHandler.h"
#include "Run.h"
#include "myRandomEngine.h"
#include "debug.h"
#include <string>
#include <cstdlib>
```

Include dependency graph for Simulator.h:



This graph shows which files directly or indirectly include this file:



Classes

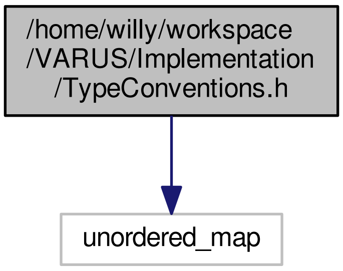
- class Simulator

  This class simulates the download and allignment-steps.

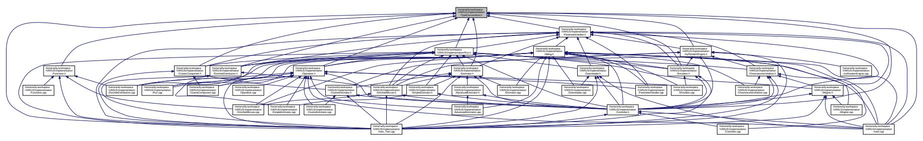
5.40 /home/willy/workspace/VARUS/Implementation/TypeConventions.h File Reference

```
#include <unordered_map>
```

Include dependency graph for TypeConventions.h:



This graph shows which files directly or indirectly include this file:



Typedefs

- typedef unsigned int U32
- typedef unsigned int ul
- typedef std::unordered_map
  < U32, U32 > UUmap
- typedef std::unordered_map
  < std::string, U32 > SUmap
- typedef std::unordered_map
  < U32, std::string > USmap
- typedef std::unordered_map
  < U32, double > UDmap
- typedef std::unordered_map
  < std::string, double > SDmap

5.40.1 Typedef Documentation

5.40.1.1 typedef std::unordered_map<std::string,double> **SDmap**

5.40.1.2 typedef std::unordered_map<std::string,U32> **SUmap**

5.40.1.3 typedef unsigned int **U32**

5.40.1.4 typedef std::unordered_map<U32,double> **UDmap**

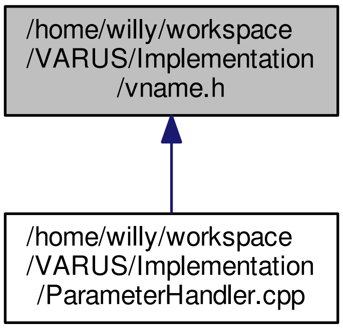
5.40.1.5 typedef unsigned int **ul**

5.40.1.6 typedef std::unordered_map<U32,std::string> USmap

5.40.1.7 typedef std::unordered_map<U32,U32> UUmap

5.41 /home/willy/workspace/VARUS/Implementation/vname.h File Reference

This graph shows which files directly or indirectly include this file:



Macros

- #define VNAME(x) #x;
- #define VDUMP(x) std::cout << #x << ":\t" << x << std::endl

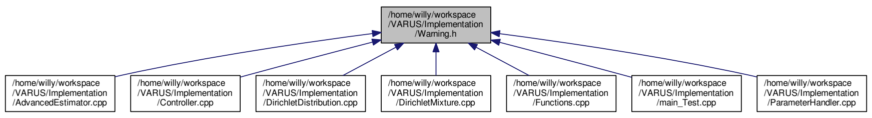
5.41.1 Macro Definition Documentation

5.41.1.1 #define VDUMP(*x*) std::cout << #x << ":\t" << x << std::endl

5.41.1.2 #define VNAME(*x*) #x;

5.42 /home/willy/workspace/VARUS/Implementation/Warning.h File Reference

This graph shows which files directly or indirectly include this file:



Macros

- #define WARN(msg) std::cerr << "WARNING in "<<__FUNCTION__<<"(): "<<msg; std::cerr<<std::endl;

5.42.1 Macro Definition Documentation

5.42.1.1 #define WARN(*msg*) std::cerr << "WARNING in "<<__FUNCTION__<<"(): "<<msg; std::cerr<<std::endl;