CRunDec3

Generated by Doxygen 1.8.11

Contents

Chapter 1

Class Index

1.1 Class List

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

Chapter 2

Class Documentation

2.1 AsmMS Struct Reference

Structure containing: $\alpha_s^{(n_f)}$ (Asexact) and $m_{MS}^{(n_f)}$ (mMSexact)

```
#include <CRunDec.h>
```

Public Attributes

- double **Asexact**
- double **mMSexact**

2.1.1 Detailed Description

Structure containing: $\alpha_s^{(n_f)}$ (Asexact) and $m_{MS}^{(n_f)}$ (mMSexact)

This structure is used to return the results of AsmMsexact.

The documentation for this struct was generated from the following file:

- CRunDec.h

2.2 CRunDec Class Reference

Main class, contains all functions.

```
#include <CRunDec.h>
```

Public Member Functions

- **CRunDec** (int)
- int GetNf ()

  GetNf returns the number of light flavours currently in use.
- void SetNf (int nf)

  SetNf sets the number of light flavours.
- double LamExpl (double asmu, double mu, int nf, int nloops)

  LamExpl calculates $\Lambda^{(n_f)}$ from $\alpha_s^{(n_f)}$ explicitly solving for $\Lambda^{(n_f)}$.
- double LamImpl (double asmu, double mu, int nf, int nloops)

  LamImpl calculates $\Lambda^{(n_f)}$ from $\alpha_s^{(n_f)}$ implicitly solving for $\Lambda^{(n_f)}$.
- double AlphasLam (double Lambda, double mu, int nf, int nloops)

  AlphasLam calculates $\alpha_s^{(n_f)}$ from $\Lambda^{(n_f)}$.
- double AlphasExact (double asmu0, double mu0, double mu1, int nf, int nloops)

  AlphasExact calculates $\alpha_s^{(n_f)}(\mu_1)$ from $\alpha_s^{(n_f)}(\mu_0)$.
- double mMS2mMS (double mu0, double asmu1, double asmu0, int nf, int nloops)

  mMS2mMS calculates $m_{MS}^{(n_f)}(\mu_1)$ from $m_{MS}^{(n_f)}(\mu_0)$
- AsmMS AsmMSrunexact (double mmu, double asmu0, double mu0, double mu1, int nf, int nloops)

  AsmMSrunexact solves simultaneously the differential equations for $\alpha_s^{(n_f)}$ and $m_{MS}^{(n_f)}$.
- double DecLambdaUp (double lam, double massth, int nl, int nloops)

  DecLambdaUp calculates $\Lambda^{(n_l+1)}$ from $\Lambda^{(n_l)}$.
- double DecLambdaDown (double lam, double massth, int nl, int nloops)

  DecLambdaDown calculates $\Lambda^{(n_l)}$ from $\Lambda^{(n_l+1)}$.
- double DecAsDownOS (double asmu, double massth, double muth, int nl, int nloops)

  DecAsDownOS calculates $\alpha_s^{(n_l)}$ from $\alpha_s^{(n_l+1)}$.
- double DecAsUpOS (double asmu, double massth, double muth, int nl, int nloops)

  DecAsUpOS calculates $\alpha_s^{(n_l+1)}$ from $\alpha_s^{(n_l)}$.
- double DecAsDownMS (double asmu, double massth, double muth, int nl, int nloops)

  DecAsDownMS calculates $\alpha_s^{(n_l)}$ from $\alpha_s^{(n_l+1)}$.
- double DecAsUpMS (double asmu, double massth, double muth, int nl, int nloops)

  DecAsUpMS calculates $\alpha_s^{(n_l+1)}$ from $\alpha_s^{(n_l)}$.
- double DecAsDownSI (double asmu, double massth, double muth, int nl, int nloops)

  DecAsDownSI calculates $\alpha_s^{(n_l)}$ from $\alpha_s^{(n_l+1)}$.
- double DecAsUpSI (double asmu, double massth, double muth, int nl, int nloops)

  DecAsUpSI calculates $\alpha_s^{(n_l+1)}$ from $\alpha_s^{(n_l)}$.
- double DecMqUpOS (double mq, double asmu, double massth, double muth, int nl, int nloops)

  DecMqUpOS calculates $m_{MS}^{(n_l+1)}(\mu)$ from $m_{MS}^{(n_l)}(\mu)$.
- double DecMqDownOS (double mq, double asmu, double massth, double muth, int nl, int nloops)

  DecMqDownOS calculates $m_{MS}^{(n_l)}(\mu)$ from $m_{MS}^{(n_l+1)}(\mu)$.
- double DecMqUpMS (double mq, double asmu, double massth, double muth, int nl, int nloops)

  DecMqUpMS calculates $m_{MS}^{(n_l+1)}(\mu)$ from $m_{MS}^{(n_l)}(\mu)$.
- double DecMqDownMS (double mq, double asmu, double massth, double muth, int nl, int nloops)

  DecMqDownMS calculates $m_{MS}^{(n_l)}(\mu)$ from $m_{MS}^{(n_l+1)}(\mu)$.
- double DecMqUpSI (double mq, double asmu, double massth, double muth, int nl, int nloops)

  DecMqUpSI calculates $m_{MS}^{(n_l+1)}(\mu)$ from $m_{MS}^{(n_l)}(\mu)$.
- double DecMqDownSI (double mq, double asmu, double massth, double muth, int nl, int nloops)

  DecMqDownSI calculates $m_{MS}^{(n_l)}(\mu)$ from $m_{MS}^{(n_l+1)}(\mu)$.
- double AlL2AlH (double asl, double mu1, TriplenfMmu decpar[ ], double mu2, int nloops)

  AlL2AlH calculates $\alpha_s(\mu_2)$ from $\alpha_s(\mu_1)$ decoupling at intermediate scales, running from low to high.
- double AlH2AlL (double ash, double mu1, TriplenfMmu decpar[ ], double mu2, int nloops)

AlH2AlL calculates $\alpha_s(\mu_2)$ from $\alpha_s(\mu_1)$ decoupling at intermediate scales, running from high to low.

- double mL2mH (double mql, double asl, double mu1, TriplenfMmu decpar[ ], double mu2, int nloops)

  mL2mH calculates $m_{MS}(\mu_2)$ from $m_{MS}(\mu_1)$ decoupling at intermediate scales, running from low to high
- double mH2mL (double mqh, double ash, double mu1, TriplenfMmu decpar[ ], double mu2, int nloops)

  mH2mL calculates $m_{MS}(\mu_2)$ from $m_{MS}(\mu_1)$ decoupling at intermediate scales, running from high to low
- double mMS2mOS (double mMS, std::pair$<$ double, double $>$ $*$mq, double asmu, double mu, int nf, int nloops, double fdelm=1.0)

  mMS2mOS calculates M_{OS} from $m_{MS}^{(n_f)}(\mu)$
- double mOS2mMS (double mOS, std::pair$<$ double, double $>$ $*$mq, double asmu, double mu, int nf, int nloops, double fdelm=1.0)

  mOS2mMS calculates $m_{MS}^{(n_f)}(\mu)$ from M_{OS}
- double mMS2mSI (double mMS, double asmu, double mu, int nf, int nloops)

  mMS2mSI calculates $m_{MS}^{(n_f)}(m_{MS})$ from $m_{MS}^{(n_f)}(\mu)$
- double mRI2mMS (double mRI, double asmu, int nf, int nloops)

  mRI2mMS calculates $m_{MS}^{(n_f)}(\mu)$ from $m^{RI}(\mu)$
- double mMS2mRGI (double mMS, double asmu, int nf, int nloops)

  mMS2mRGI calculates m^{RGI} from $m_{MS}^{(n_f)}(\mu)$
- double mRGI2mMS (double mRGI, double asmu, int nf, int nloops)

  mRGI2mMS calculates $m_{MS}^{(n_f)}(\mu)$ from m^{RGI}
- double mOS2mSI (double mOS, std::pair$<$ double, double $>$ $*$mq, double asM, int nf, int nloops, double fdelm=1.0)

  mOS2mSI calculates $m_{MS}^{(n_f)}(m_{MS})$ from M_{OS}
- double mOS2mMSrun (double mOS, std::pair$<$ double, double $>$ $*$mq, double asmu, double mu, int nf, int nloops)

  mOS2mMSrun calculates $m_{MS}^{(n_f)}(\mu)$ from M_{OS}
- double mMS2mOSrun (double mMS, std::pair$<$ double, double $>$ $*$mq, double asmu, double mu, int nf, int nloops)

  mMS2mOSrun calculates M_{OS} from $m_{MS}^{(n_f)}(\mu)$
- double mMS2mRI (double mMS, double asmu, int nf, int nloops)

  mMS2mRI calculates $m^{RI}(\mu)$ from $m_{MS}^{(n_f)}(\mu)$
- double mOS2mMSit (double mOS, std::pair$<$ double, double $>$ $*$mq, double asmu, double mu, int nf, int nloops)

  mOS2mMSit calculates $m_{MS}^{(n_f)}(\mu)$ from M_{OS}
- double mMS2mRGImod (double mMS, double asmu, int nf, int nloops)

  mMS2mRGImod calculates m^{RGI} from $m_{MS}^{(n_f)}(\mu)$
- double mOS2mPS (double mOS, std::pair$<$ double, double $>$ $*$mq, double asmu, double mu, double muf, int nl, int nloops)

  mOS2mPS calculates $m^{PS}(\mu_f)$ from M_{OS}
- double mMS2mPS (double mMS, std::pair$<$ double, double $>$ $*$mq, double asmu, double mu, double muf, int nl, int nloops, double fdelm=1.0)

  mMS2mPS calculates $m^{PS}(\mu_f)$ from $m_{MS}^{(n_l+1)}(\mu)$
- double mPS2mMS (double mPS, std::pair$<$ double, double $>$ $*$mq, double asmu, double mu, double muf, int nl, int nloops, double fdelm=1.0)

  mPS2mMS calculates $m_{MS}^{(n_l+1)}(\mu)$ from $m^{PS}(\mu_f)$
- double mPS2mSI (double mPS, std::pair$<$ double, double $>$ $*$mq, double($*$as)(double), double muf, int nl, int nloops, double fdelm=1.0)

  mPS2mSI calculates $m_{MS}^{(n_l+1)}(m_{MS})$ from $m^{PS}(\mu_f)$
- double mOS2m1S (double mOS, std::pair$<$ double, double $>$ $*$mq, double asmu, double mu, int nl, int nloops)

  mOS2m1S calculates m^{1S} from M_{OS}
- double mMS2m1S (double mMS, std::pair$<$ double, double $>$ $*$mq, double asmu, double mu, int nl, int nloops, double fdelm=1.0)

mMS2m1S calculates m^{1S} *from* $m_{MS}^{(n_l+1)}(\mu)$

- double m1S2mMS (double m1S, std::pair$<$ double, double $>$ $*$mq, double asmu, double mu, int nl, int nloops, double fdelm=1.0)

  m1S2mMS calculates $m_{MS}^{(n_l+1)}(\mu)$ *from* m^{1S}
- double m1S2mSI (double m1S, std::pair$<$ double, double $>$ $*$mq, double($*$as)(double), int nl, int nloops, double fdelm=1.0)

  m1S2mSI calculates $m_{MS}^{(n_l+1)}(m_{MS})$ *from* m^{1S}
- double mOS2mRS (double mOS, std::pair$<$ double, double $>$ $*$mq, double asmu, double mu, double nuf, int nl, int nloops, bool prime)

  mOS2mRS calculates $m^{RS}(\nu_f)$ *or* $m^{RS'}(\nu_f)$ *from* M_{OS}
- double mMS2mRS (double mMS, std::pair$<$ double, double $>$ $*$mq, double asmu, double mu, double nuf, int nl, int nloops, double fdelm, bool prime)

  mMS2mRS calculates $m^{RS}(\nu_f)$ *or* $m^{RS'}(\nu_f)$ *from* $m_{MS}^{(n_l+1)}(\mu)$
- double mRS2mMS (double mRS, std::pair$<$ double, double $>$ $*$mq, double asmu, double mu, double nuf, int nl, int nloops, double fdelm, bool prime)

  mRS2mMS calculates $m_{MS}^{(n_l+1)}(\mu)$ *from* $m^{RS}(\nu_f)$ *or* $m^{RS'}(\nu_f)$
- double mRS2mSI (double mRS, std::pair$<$ double, double $>$ $*$mq, double($*$as)(double), double nuf, int nl, int nloops, double fdelm, bool prime)

  mRS2mSI calculates $m_{MS}^{(n_l+1)}(m_{MS})$ *from* $m^{RS}(\nu_f)$ *or* $m^{RS'}(\nu_f)$
- double mOS2mRS (double mOS, std::pair$<$ double, double $>$ $*$mq, double asmu, double mu, double nuf, int nl, int nloops)

  mOS2mRS calculates $m^{RS}(\nu_f)$ *from* M_{OS}
- double mMS2mRS (double mMS, std::pair$<$ double, double $>$ $*$mq, double asmu, double mu, double nuf, int nl, int nloops, double fdelm=1.0)

  mMS2mRS calculates $m^{RS}(\nu_f)$ *from* $m_{MS}^{(n_l+1)}(\mu)$
- double mRS2mMS (double mRS, std::pair$<$ double, double $>$ $*$mq, double asmu, double mu, double nuf, int nl, int nloops, double fdelm=1.0)

  mRS2mMS calculates $m_{MS}^{(n_l+1)}(\mu)$ *from* $m^{RS}(\nu_f)$
- double mRS2mSI (double mRS, std::pair$<$ double, double $>$ $*$mq, double($*$as)(double), double nuf, int nl, int nloops, double fdelm=1.0)

  mRS2mSI calculates $m_{MS}^{(n_l+1)}(m_{MS})$ *from* $m^{RS}(\nu_f)$
- double mOS2mRSp (double mOS, std::pair$<$ double, double $>$ $*$mq, double asmu, double mu, double nuf, int nl, int nloops)

  mOS2mRSp calculates $m^{RS'}(\nu_f)$ *from* M_{OS}
- double mMS2mRSp (double mMS, std::pair$<$ double, double $>$ $*$mq, double asmu, double mu, double nuf, int nl, int nloops, double fdelm=1.0)

  mMS2mRSp calculates $m^{RS'}(\nu_f)$ *from* $m_{MS}^{(n_l+1)}(\mu)$
- double mRSp2mMS (double mRS, std::pair$<$ double, double $>$ $*$mq, double asmu, double mu, double nuf, int nl, int nloops, double fdelm=1.0)

  mRSp2mMS calculates $m_{MS}^{(n_l+1)}(\mu)$ *from* $m^{RS'}(\nu_f)$
- double mRSp2mSI (double mRS, std::pair$<$ double, double $>$ $*$mq, double($*$as)(double), double nuf, int nl, int nloops, double fdelm=1.0)

  mRSp2mSI calculates $m_{MS}^{(n_l+1)}(m_{MS})$ *from* $m^{RS'}(\nu_f)$
- double **LamExpl** (double asmu, double mu, int nloops)
- double **LamImpl** (double asmu, double mu, int nloops)
- double **AlphasLam** (double Lambda, double mu, int nloops)
- double **AlphasExact** (double asmu0, double mu0, double mu1, int nloops)
- double **mMS2mMS** (double mmu0, double asmu0, double asmu1, int nloops)
- AsmMS **AsmMSrunexact** (double mmu, double asmu0, double mu0, double mu1, int nloops)
- double **DecAsDownOS** (double asmu, double massth, double muth, int nloops)
- double **DecAsUpOS** (double asmu, double massth, double muth, int nloops)
- double **DecAsDownMS** (double asmu, double massth, double muth, int nloops)
- double **DecAsUpMS** (double asmu, double massth, double muth, int nloops)

- double **DecAsDownSI** (double asmu, double massth, double muth, int nloops)
- double **DecAsUpSI** (double asmu, double massth, double muth, int nloops)
- double **DecMqUpOS** (double mq, double asmu, double massth, double muth, int nloops)
- double **DecMqDownOS** (double mq, double asmu, double massth, double muth, int nloops)
- double **DecMqUpMS** (double mq, double asmu, double massth, double muth, int nloops)
- double **DecMqDownMS** (double mq, double asmu, double massth, double muth, int nloops)
- double **DecMqUpSI** (double mq, double asmu, double massth, double muth, int nloops)
- double **DecMqDownSI** (double mq, double asmu, double massth, double muth, int nloops)
- double **mMS2mOS** (double mMS, std::pair$<$ double, double $>$ $*$mq, double asmu, double mu, int nloops, double fdelm=1.0)
- double **mOS2mMS** (double mOS, std::pair$<$ double, double $>$ $*$mq, double asmu, double mu, int nloops, double fdelm=1.0)
- double **mMS2mSI** (double mMS, double asmu, double mu, int nloops)
- double **mRI2mMS** (double mRI, double asmu, int nloops)
- double **mMS2mRGI** (double mMS, double asmu, int nloops)
- double **mRGI2mMS** (double mRGI, double asmu, int nloops)
- double **mOS2mSI** (double mOS, std::pair$<$ double, double $>$ $*$mq, double asM, int nloops, double fdelm=1.0)
- double **mOS2mMSrun** (double mOS, std::pair$<$ double, double $>$ $*$mq, double asmu, double mu, int nloops)
- double **mMS2mOSrun** (double mMS, std::pair$<$ double, double $>$ $*$mq, double asmu, double mu, int nloops)
- double **mMS2mRI** (double mMS, double asmu, int nloops)
- double **mOS2mMSit** (double mOS, std::pair$<$ double, double $>$ $*$mq, double asmu, double mu, int nloops)
- double **mMS2mRGImod** (double mMS, double asmu, int nloops)

Public Attributes

- std::pair$<$ double, double $>$ **mq** [4]
- TriplenfMmu **nfMmu** [4]
- AsmMS **AM**

Friends

- double **fSetdydx** (CRunDec S, double A, int nl)
- double **fSetdydxa1** (CRunDec S, double x, double A)
- double **fSetdydxM1** (CRunDec S, double A, double M)
- double **fSetdydxa2** (CRunDec S, double x, double A)
- double **fSetdydxM2** (CRunDec S, double A, double M)
- double **fSetdydxa3** (CRunDec S, double x, double A)
- double **fSetdydxM3** (CRunDec S, double A, double M)
- double **fSetdydxa4** (CRunDec S, double x, double A)
- double **fSetdydxM4** (CRunDec S, double A, double M)
- double **fSetdydxa5** (CRunDec S, double x, double A)
- double **fSetdydxM5** (CRunDec S, double A, double M)

2.2.1 Detailed Description

Main class, contains all functions.

All RunDec functions are accessed via this class. For many of the functions versions with and without the parameters nl/nf exist. The ones with the parameters nl/nf call the ones without after calling SetNf(nl)/SetNf(nf). It is recommended not to use the ones without nl/nf since some of them change the values of nl/nf during their execution.

Detailed descriptions are only provided for the the functions with parameters nl/nf.

2.2.2 Member Function Documentation

2.2.2.1 double CRunDec::AlH2AlL (double *ash,* double *mu1,* TriplenfMmu *decpar[ ],* double *mu2,* int *nloops*)

AlH2AlL calculates $\alpha_s(\mu_2)$ from $\alpha_s(\mu_1)$ decoupling at intermediate scales, running from high to low.

Parameters

ash	$\alpha_s(\mu_1)$
mu1	μ_1
decpar	arrays of triples indicating the number of flavours, the OS-mass and the scale at which decoupling is performed
mu2	μ_2
nloops	number of loops

Returns

$\alpha_s(\mu_2)$

2.2.2.2 double CRunDec::AlL2AlH (double *asl,* double *mu1,* TriplenfMmu *decpar[ ],* double *mu2,* int *nloops*)

AlL2AlH calculates $\alpha_s(\mu_2)$ from $\alpha_s(\mu_1)$ decoupling at intermediate scales, running from low to high.

Parameters

asl	$\alpha_s(\mu_1)$
mu1	μ_1
decpar	arrays of triples indicating the number of flavours, the OS-mass and the scale at which decoupling is performed
mu2	μ_2
nloops	number of loops

Returns

$\alpha_s(\mu_2)$

2.2.2.3 double CRunDec::AlphasExact (double *asmu0,* double *mu0,* double *mu1,* int *nf,* int *nloops*)

AlphasExact calculates $\alpha_s^{(n_f)}(\mu_1)$ from $\alpha_s^{(n_f)}(\mu_0)$.

Parameters

asmu0	$\alpha_s^{(n_f)}(\mu_0)$
mu0	μ_0
mu1	μ_1
nf	n_f
nloops	number of loops

Returns

$\alpha_s^{(n_f)}(\mu_1)$

2.2.2.4 double CRunDec::AlphasLam (double *Lambda*, double *mu*, int *nf*, int *nloops*)

AlphasLam calculates $\alpha_s^{(n_f)}$ from $\Lambda^{(n_f)}$.

Parameters

Lambda	$\Lambda^{(n_f)}$
mu	μ
nf	n_f
nloops	number of loops

Returns

$\alpha_s^{(n_f)}(\mu)$

2.2.2.5 AsmMS CRunDec::AsmMSrunexact (double *mmu*, double *asmu0*, double *mu0*, double *mu1*, int *nf*, int *nloops*)

AsmMSrunexact solves simultaneously the differential equations for $\alpha_s^{(n_f)}$ and $m_{MS}^{(n_f)}$.

Returns a structure of type AsmMS.

Parameters

mmu	$m_{MS}^{(n_f)}(\mu_0)$
asmu0	$\alpha_s^{(n_f)}(\mu_0)$
mu0	μ_0
mu1	μ_1
nf	n_f
nloops	number of loops

Returns

$(\alpha_s^{(n_f)}(\mu_1), m_{MS}^{(n_f)}(\mu_1))$

2.2.2.6 double CRunDec::DecAsDownMS (double *asmu*, double *massth*, double *muth*, int *nl*, int *nloops*)

DecAsDownMS calculates $\alpha_s^{(n_l)}$ from $\alpha_s^{(n_l+1)}$.

Parameters

asmu	$\alpha_s^{(n_l+1)}(\mu)$
massth	mass of the heavy quark ($m_{MS}^{(n_l+1)}(\mu)$)
muth	scale μ at which the matching is performed
nl	n_l
nloops	number of loops

Returns

$\alpha_s^{(n_l)}(\mu)$

2.2.2.7 double CRunDec::DecAsDownOS (double *asmu,* double *massth,* double *muth,* int *nl,* int *nloops*)

DecAsDownOS calculates $\alpha_s^{(n_l)}$ from $\alpha_s^{(n_l+1)}$.

Parameters

asmu	$\alpha_s^{(n_l+1)}(\mu)$
massth	mass of the heavy quark (M_{OS})
muth	scale μ at which the matching is performed
nl	n_l
nloops	number of loops

Returns

$\alpha_s^{(n_l)}(\mu)$

2.2.2.8 double CRunDec::DecAsDownSI (double *asmu,* double *massth,* double *muth,* int *nl,* int *nloops*)

DecAsDownSI calculates $\alpha_s^{(n_l)}$ from $\alpha_s^{(n_l+1)}$.

Parameters

asmu	$\alpha_s^{(n_l+1)}(\mu)$
massth	mass of the heavy quark ($m_{MS}^{(n_l+1)}(m_{MS})$)
muth	scale μ at which the matching is performed
nl	n_l
nloops	number of loops

Returns

$\alpha_s^{(n_l)}(\mu)$

2.2.2.9 double CRunDec::DecAsUpMS (double *asmu,* double *massth,* double *muth,* int *nl,* int *nloops*)

DecAsUpMS calculates $\alpha_s^{(n_l+1)}$ from $\alpha_s^{(n_l)}$.

Parameters

asmu	$\alpha_s^{(n_l)}(\mu)$
massth	mass of the heavy quark ($m_{MS}^{(n_l+1)}(\mu)$)
muth	scale μ at which the matching is performed
nl	n_l
nloops	number of loops

Returns

$\alpha_s^{(n_l+1)}(\mu)$

2.2.2.10 double CRunDec::DecAsUpOS (double *asmu,* double *massth,* double *muth,* int *nl,* int *nloops*)

DecAsUpOS calculates $\alpha_s^{(n_l+1)}$ from $\alpha_s^{(n_l)}$.

Parameters

asmu	$\alpha_s^{(n_l)}(\mu)$
massth	mass of the heavy quark (M_{OS})
muth	scale μ at which the matching is performed
nl	n_l
nloops	number of loops

Returns

$\alpha_s^{(n_l+1)}(\mu)$

2.2.2.11 double CRunDec::DecAsUpSI (double *asmu,* double *massth,* double *muth,* int *nl,* int *nloops*)

DecAsUpSI calculates $\alpha_s^{(n_l+1)}$ from $\alpha_s^{(n_l)}$.

Parameters

asmu	$\alpha_s^{(n_l)}(\mu)$
massth	mass of the heavy quark ($m_{MS}^{(n_l+1)}(m_{MS})$)
muth	scale μ at which the matching is performed
nl	n_l
nloops	number of loops

Returns

$\alpha_s^{(n_l+1)}(\mu)$

2.2.2.12 double CRunDec::DecLambdaDown (double *lam,* double *massth,* int *nl,* int *nloops*)

DecLambdaDown calculates $\Lambda^{(n_l)}$ from $\Lambda^{(n_l+1)}$.

Parameters

lam	$\Lambda^{(n_l+1)}$
massth	quark mass at which the matching is performed ($m_{MS}^{(n_l+1)}(m_{MS})$)
nl	n_l
nloops	number of loops

Returns

$\Lambda^{(n_l)}$

2.2.2.13 double CRunDec::DecLambdaUp (double *lam*, double *massth*, int *nl*, int *nloops*)

DecLambdaUp calculates $\Lambda^{(n_l+1)}$ from $\Lambda^{(n_l)}$.

Parameters

lam	$\Lambda^{(n_l)}$
massth	quark mass at which the matching is performed ($m_{MS}^{(n_l+1)}(m_{MS})$)
nl	n_l
nloops	number of loops

Returns

$\Lambda^{(n_l+1)}$

2.2.2.14 double CRunDec::DecMqDownMS (double *mq*, double *asmu*, double *massth*, double *muth*, int *nl*, int *nloops*)

DecMqDownMS calculates $m_{MS}^{(n_l)}(\mu)$ from $m_{MS}^{(n_l+1)}(\mu)$.

Parameters

mq	$m_{MS}^{(n_l+1)}(\mu)$
asmu	$\alpha_s^{(n_l+1)}(\mu)$
massth	mass of the heavy quark ($m_{MS}^{(n_l+1)}(\mu)$)
muth	scale μ at which the matching is performed
nl	n_l
nloops	number of loops

Returns

$m_{MS}^{(n_l)}(\mu)$

2.2.2.15 double CRunDec::DecMqDownOS (double *mq*, double *asmu*, double *massth*, double *muth*, int *nl*, int *nloops*)

DecMqDownOS calculates $m_{MS}^{(n_l)}(\mu)$ from $m_{MS}^{(n_l+1)}(\mu)$.

Parameters

mq	$m_{MS}^{(n_l+1)}(\mu)$
asmu	$\alpha_s^{(n_l+1)}(\mu)$

Parameters

massth	mass of the heavy quark (M_{OS})
muth	scale μ at which the matching is performed
nl	n_l
nloops	number of loops

Returns

$m_{MS}^{(n_l)}(\mu)$

2.2.2.16 double CRunDec::DecMqDownSI (double *mq*, double *asmu*, double *massth*, double *muth*, int *nl*, int *nloops*)

DecMqDownSI calculates $m_{MS}^{(n_l)}(\mu)$ from $m_{MS}^{(n_l+1)}(\mu)$.

Parameters

mq	$m_{MS}^{(n_l+1)}(\mu)$
asmu	$\alpha_s^{(n_l+1)}(\mu)$
massth	mass of the heavy quark ($m_{MS}^{(n_l+1)}(m_{MS})$)
muth	scale μ at which the matching is performed
nl	n_l
nloops	number of loops

Returns

$m_{MS}^{(n_l)}(\mu)$

2.2.2.17 double CRunDec::DecMqUpMS (double *mq*, double *asmu*, double *massth*, double *muth*, int *nl*, int *nloops*)

DecMqUpMS calculates $m_{MS}^{(n_l+1)}(\mu)$ from $m_{MS}^{(n_l)}(\mu)$.

Parameters

mq	$m_{MS}^{(n_l)}(\mu)$
asmu	$\alpha_s^{(n_l)}(\mu)$
massth	mass of the heavy quark ($m_{MS}^{(n_l+1)}(\mu)$)
muth	scale μ at which the matching is performed
nl	n_l
nloops	number of loops

Returns

$m_{MS}^{(n_l+1)}(\mu)$

2.2.2.18 double CRunDec::DecMqUpOS (double *mq,* double *asmu,* double *massth,* double *muth,* int *nl,* int *nloops*)

DecMqUpOS calculates $m_{MS}^{(n_l+1)}(\mu)$ from $m_{MS}^{(n_l)}(\mu)$.

Parameters

mq	$m_{MS}^{(n_l)}(\mu)$
asmu	$\alpha_s^{(n_l)}(\mu)$
massth	mass of the heavy quark (M_{OS})
muth	scale μ at which the matching is performed
nl	n_l
nloops	number of loops

Returns

$m_{MS}^{(n_l+1)}(\mu)$

2.2.2.19 double CRunDec::DecMqUpSI (double *mq,* double *asmu,* double *massth,* double *muth,* int *nl,* int *nloops*)

DecMqUpSI calculates $m_{MS}^{(n_l+1)}(\mu)$ from $m_{MS}^{(n_l)}(\mu)$.

Parameters

mq	$m_{MS}^{(n_l)}(\mu)$
asmu	$\alpha_s^{(n_l)}(\mu)$
massth	mass of the heavy quark ($m_{MS}^{(n_l+1)}(m_{MS})$)
muth	scale μ at which the matching is performed
nl	n_l
nloops	number of loops

Returns

$m_{MS}^{(n_l+1)}(\mu)$

2.2.2.20 int CRunDec::GetNf ( )

GetNf returns the number of light flavours currently in use.

Returns

n_f

2.2.2.21 double CRunDec::LamExpl (double *asmu*, double *mu*, int *nf*, int *nloops*)

LamExpl calculates $\Lambda^{(n_f)}$ from $\alpha_s^{(n_f)}$ explicitly solving for $\Lambda^{(n_f)}$.

Parameters

asmu	$\alpha_s^{(n_f)}(\mu)$
mu	μ
nf	n_f
nloops	number of loops

Returns

$\Lambda^{(n_f)}$

2.2.2.22 double CRunDec::LamImpl (double *asmu*, double *mu*, int *nf*, int *nloops*)

LamImpl calculates $\Lambda^{(n_f)}$ from $\alpha_s^{(n_f)}$ implicitly solving for $\Lambda^{(n_f)}$.

Parameters

asmu	$\alpha_s^{(n_f)}(\mu)$
mu	μ
nf	n_f
nloops	number of loops

Returns

$\Lambda^{(n_f)}$

2.2.2.23 double CRunDec::m1S2mMS (double *m1S*, std::pair< double, double > * *mq*, double *asmu*, double *mu*, int *nl*, int *nloops*, double *fdelm* = `1.0`)

m1S2mMS calculates $m_{MS}^{(n_l+1)}(\mu)$ from m^{1S}

Parameters

m1S	m^{PS}
mq	pointer to pairs of light quark masses, not active
asmu	$\alpha_s^{(n_l)}(\mu)$
mu	μ
nl	n_l
nloops	number of loops
fdelm	factor multiplying the non-logarithmic part of the 4-loop term

Returns

$m_{MS}^{(n_l+1)}(\mu)$

2.2.2.24 double CRunDec::m1S2mSI (double *m1S*, std::pair$<$ double, double $>$ $*$ *mq*, double($*$)(double) *as*, int *nl*, int *nloops*, double *fdelm* = `1.0`)

m1S2mSI calculates $m_{MS}^{(n_l+1)}(m_{MS})$ from m^{1S}

Parameters

m1S	m^{1S}
mq	pointer to pairs of light quark masses, not active
as	pointer to a function which calculates $\alpha_s^{(n_l)}(\mu)$ and takes μ as argument
nl	n_l
nloops	number of loops
fdelm	factor multiplying the non-logarithmic part of the 4-loop term

Returns

$m_{MS}^{(n_l+1)}(m_{MS})$

2.2.2.25 double CRunDec::mH2mL (double *mqh*, double *ash*, double *mu1*, TriplenfMmu *decpar[ ]*, double *mu2*, int *nloops*)

mH2mL calculates $m_{MS}(\mu_2)$ from $m_{MS}(\mu_1)$ decoupling at intermediate scales, running from high to low

Parameters

mqh	$m_{MS}(\mu_1)$
ash	$\alpha_s(\mu_1)$
mu1	μ_1
decpar	arrays of triples indicating the number of flavours, the OS-mass and the scale at which decoupling is performed
mu2	μ_2
nloops	number of loops

Returns

$m_{MS}(\mu_2)$

2.2.2.26 double CRunDec::mL2mH (double *mql*, double *asl*, double *mu1*, TriplenfMmu *decpar[ ]*, double *mu2*, int *nloops*)

mL2mH calculates $m_{MS}(\mu_2)$ from $m_{MS}(\mu_1)$ decoupling at intermediate scales, running from low to high

Parameters

mql	$m_{MS}(\mu_1)$
asl	$\alpha_s(\mu_1)$
mu1	μ_1
decpar	arrays of triples indicating the number of flavours, the OS-mass and the scale at which decoupling is performed
mu2	μ_2
nloops	number of loops

Returns

$m_{MS}(\mu_2)$

2.2.2.27 double CRunDec::mMS2m1S (double *mMS,* std::pair< double, double > * *mq,* double *asmu,* double *mu,* int *nl,* int *nloops,* double *fdelm* = `1.0`)

mMS2m1S calculates m^{1S} from $m_{MS}^{(n_l+1)}(\mu)$

Parameters

mMS	$m_{MS}^{(n_l+1)}(\mu)$
mq	pointer to pairs of light quark masses, not active
asmu	$\alpha_s^{(n_l)}(\mu)$
mu	μ
nl	n_l
nloops	number of loops
fdelm	factor multiplying the non-logarithmic part of the 4-loop term

Returns

m^{1S}

2.2.2.28 double CRunDec::mMS2mMS (double *mu0,* double *asmu1,* double *asmu0,* int *nf,* int *nloops*)

mMS2mMS calculates $m_{MS}^{(n_f)}(\mu_1)$ from $m_{MS}^{(n_f)}(\mu_0)$

Parameters

mu0	$m_{MS}^{(n_f)}(\mu_0)$
asmu0	$\alpha_s^{(n_f)}(\mu_0)$
asmu1	$\alpha_s^{(n_f)}(\mu_1)$
nf	n_f
nloops	number of loops

Returns

$m_{MS}^{(n_f)}(\mu_1)$

2.2.2.29 double CRunDec::mMS2mOS (double *mMS,* std::pair$<$ double, double $> *$ *mq,* double *asmu,* double *mu,* int *nf,* int *nloops,* double *fdelm* = `1.0`)

mMS2mOS calculates M_{OS} from $m_{MS}^{(n_f)}(\mu)$

Parameters

mMS	$m_{MS}^{(n_f)}(\mu)$
mq	pointer to pairs of light quark masses
asmu	$\alpha_s^{(n_f)}(\mu)$
mu	μ
nf	n_f
nloops	number of loops
fdelm	factor multiplying the non-logarithmic part of the 4-loop term

Returns

M_{OS}

2.2.2.30 double CRunDec::mMS2mOSrun (double *mMS,* std::pair$<$ double, double $> *$ *mq,* double *asmu,* double *mu,* int *nf,* int *nloops*)

mMS2mOSrun calculates M_{OS} from $m_{MS}^{(n_f)}(\mu)$

Calculates $m_{MS}^{(n_f)}(m_{MS})$ in an intermediate step

Parameters

mMS	$m_{MS}^{(n_f)}(\mu)$
mq	pointer to pairs of light quark masses
asmu	$\alpha_s^{(n_f)}(\mu)$
mu	μ
nf	n_f
nloops	number of loops

Returns

M_{OS}

2.2.2.31 double CRunDec::mMS2mPS (double *mMS*, std::pair$<$ double, double $> *$ *mq*, double *asmu*, double *mu*, double *muf*, int *nl*, int *nloops*, double *fdelm* = `1.0`)

mMS2mPS calculates $m^{PS}(\mu_f)$ from $m_{MS}^{(n_l+1)}(\mu)$

Parameters

mMS	$m_{MS}^{(n_l+1)}(\mu)$
mq	pointer to pairs of light quark masses, not active
asmu	$\alpha_s^{(n_l)}(\mu)$
mu	μ
muf	μ_f
nl	n_l
nloops	number of loops
fdelm	factor multiplying the non-logarithmic part of the 4-loop term

Returns

$m^{PS}(\mu_f)$

2.2.2.32 double CRunDec::mMS2mRGI (double *mMS*, double *asmu*, int *nf*, int *nloops*)

mMS2mRGI calculates m^{RGI} from $m_{MS}^{(n_f)}(\mu)$

for the difference between mMS2mRGImod and mMS2mRGI see arXiv:1201:6149

Parameters

mMS	$m_{MS}^{(n_f)}(\mu)$
asmu	$\alpha_s^{(n_f)}(\mu)$
nf	n_f
nloops	number of loops

Returns

m^{RGI}

2.2.2.33 double CRunDec::mMS2mRGImod (double *mMS*, double *asmu*, int *nf*, int *nloops*)

mMS2mRGImod calculates m^{RGI} from $m_{MS}^{(n_f)}(\mu)$

for the difference between mMS2mRGImod and mMS2mRGI see arXiv:1201:6149

Parameters

mMS	$m_{MS}^{(n_f)}(\mu)$
asmu	$\alpha_s^{(n_f)}(\mu)$
nf	n_f
nloops	number of loops

Returns

m^{RGI}

2.2.2.34 double CRunDec::mMS2mRI (double *mMS,* double *asmu,* int *nf,* int *nloops*)

mMS2mRI calculates $m^{RI}(\mu)$ from $m_{MS}^{(n_f)}(\mu)$

Parameters

mMS	$m_{MS}^{(n_f)}(\mu)$
asmu	$\alpha_s^{(n_f)}(\mu)$
nf	n_f
nloops	number of loops

Returns

$m^{RI}(\mu)$

2.2.2.35 double CRunDec::mMS2mRS (double *mMS,* std::pair$<$ double, double $> *$ *mq,* double *asmu,* double *mu,* double *nuf,* int *nl,* int *nloops,* double *fdelm,* bool *prime*)

mMS2mRS calculates $m^{RS}(\nu_f)$ or $m^{RS'}(\nu_f)$ from $m_{MS}^{(n_l+1)}(\mu)$

This is a helper function, use the version without the parameter prime or mMS2mRSp

Parameters

mMS	$m_{MS}^{(n_l+1)}(\mu)$
mq	pointer to pairs of light quark masses, not active
asmu	$\alpha_s^{(n_l)}(\mu)$
mu	μ
nuf	ν_f
nl	n_l
nloops	number of loops
fdelm	factor multiplying the non-logarithmic part of the 4-loop term
prime	selects if m^{RS} or $m^{RS'}$ should be calculated

Returns

$m^{RS}(\nu_f)$ or $m^{RS'}(\nu_f)$

2.2.2.36 double CRunDec::mMS2mRS (double *mMS*, std::pair< double, double > * *mq*, double *asmu*, double *mu*, double *nuf*, int *nl*, int *nloops*, double *fdelm* = `1.0`)

mMS2mRS calculates $m^{RS}(\nu_f)$ from $m_{MS}^{(n_l+1)}(\mu)$

Parameters

mMS	$m_{MS}^{(n_l+1)}(\mu)$
mq	pointer to pairs of light quark masses, not active
asmu	$\alpha_s^{(n_l)}(\mu)$
mu	μ
nuf	ν_f
nl	n_l
nloops	number of loops
fdelm	factor multiplying the non-logarithmic part of the 4-loop term

Returns

$m^{RS}(\nu_f)$

2.2.2.37 double CRunDec::mMS2mRSp (double *mMS*, std::pair< double, double > * *mq*, double *asmu*, double *mu*, double *nuf*, int *nl*, int *nloops*, double *fdelm* = `1.0`)

mMS2mRSp calculates $m^{RS'}(\nu_f)$ from $m_{MS}^{(n_l+1)}(\mu)$

Parameters

mMS	$m_{MS}^{(n_l+1)}(\mu)$
mq	pointer to pairs of light quark masses, not active
asmu	$\alpha_s^{(n_l)}(\mu)$
mu	μ
nuf	ν_f
nl	n_l
nloops	number of loops
fdelm	factor multiplying the non-logarithmic part of the 4-loop term

Returns

$m^{RS'}(\nu_f)$

2.2.2.38 double CRunDec::mMS2mSI (double *mMS*, double *asmu*, double *mu*, int *nf*, int *nloops*)

mMS2mSI calculates $m_{MS}^{(n_f)}(m_{MS})$ from $m_{MS}^{(n_f)}(\mu)$

Parameters

mMS	$m_{MS}^{(n_f)}(\mu)$
asmu	$\alpha_s^{(n_f)}(\mu)$
mu	μ
nf	n_f
nloops	number of loops

Returns

$m_{MS}^{(n_f)}(m_{MS})$

2.2.2.39 double CRunDec::mOS2m1S (double *mOS*, std::pair< double, double > * *mq*, double *asmu*, double *mu*, int *nl*, int *nloops*)

mOS2m1S calculates m^{1S} from M_{OS}

Parameters

mOS	M_{OS}
mq	pointer to pairs of light quark masses, not active
asmu	$\alpha_s^{(n_l)}(\mu)$
mu	μ
nl	n_l
nloops	number of loops

Returns

m^{1S}

2.2.2.40 double CRunDec::mOS2mMS (double *mOS*, std::pair< double, double > * *mq*, double *asmu*, double *mu*, int *nf*, int *nloops*, double *fdelm* = `1.0`)

mOS2mMS calculates $m_{MS}^{(n_f)}(\mu)$ from M_{OS}

Parameters

mOS	M_{OS}
mq	pointer to pairs of light quark masses
asmu	$\alpha_s^{(n_f)}(\mu)$
mu	μ
nf	n_f
nloops	number of loops
fdelm	factor multiplying the non-logarithmic part of the 4-loop term

Returns

$m_{MS}^{(n_f)}(\mu)$

2.2.2.41 double CRunDec::mOS2mMSit (double *mOS*, std::pair$<$ double, double $> *$ *mq*, double *asmu*, double *mu*, int *nf*, int *nloops*)

mOS2mMSit calculates $m_{MS}^{(n_f)}(\mu)$ from M_{OS}

DEPRECATED, will be removed in future versions

Parameters

mOS	M_{OS}
mq	pointer to pairs of light quark masses
asmu	$\alpha_s^{(n_f)}(\mu)$
mu	μ
nf	n_f
nloops	number of loops

Returns

$m_{MS}^{(n_f)}(\mu)$

2.2.2.42 double CRunDec::mOS2mMSrun (double *mOS*, std::pair$<$ double, double $> *$ *mq*, double *asmu*, double *mu*, int *nf*, int *nloops*)

mOS2mMSrun calculates $m_{MS}^{(n_f)}(\mu)$ from M_{OS}

Calculates $m_{MS}^{(n_f)}(m_{MS})$ in an intermediate step

Parameters

mOS	M_{OS}
mq	pointer to pairs of light quark masses
asmu	$\alpha_s^{(n_f)}(\mu)$
mu	μ
nf	n_f
nloops	number of loops

Returns

$m_{MS}^{(n_f)}(\mu)$

2.2.2.43 double CRunDec::mOS2mPS (double *mOS*, std::pair$<$ double, double $>$ $*$ *mq*, double *asmu*, double *mu*, double *muf*, int *nl*, int *nloops*)

mOS2mPS calculates $m^{PS}(\mu_f)$ from M_{OS}

Parameters

mOS	M_{OS}
mq	pointer to pairs of light quark masses, not active
asmu	$\alpha_s^{(n_l)}(\mu)$
mu	μ
muf	μ_f
nl	n_l
nloops	number of loops

Returns

$m^{PS}(\mu_f)$

2.2.2.44 double CRunDec::mOS2mRS (double *mOS*, std::pair$<$ double, double $>$ $*$ *mq*, double *asmu*, double *mu*, double *nuf*, int *nl*, int *nloops*, bool *prime*)

mOS2mRS calculates $m^{RS}(\nu_f)$ or $m^{RS'}(\nu_f)$ from M_{OS}

This is a helper function, use the version without the parameter prime or mOS2mRSp

Parameters

mOS	M_{OS}
mq	pointer to pairs of light quark masses, not active
asmu	$\alpha_s^{(n_l)}(\mu)$
mu	μ
nuf	ν_f
nl	n_l
nloops	number of loops
prime	selects if m^{RS} or $m^{RS'}$ should be calculated

Returns

$m^{RS}(\nu_f)$ or $m^{RS'}(\nu_f)$

2.2.2.45 double CRunDec::mOS2mRS (double *mOS*, std::pair$<$ double, double $>$ $*$ *mq*, double *asmu*, double *mu*, double *nuf*, int *nl*, int *nloops*)

mOS2mRS calculates $m^{RS}(\nu_f)$ from M_{OS}

Parameters

mOS	M_{OS}
mq	pointer to pairs of light quark masses, not active
asmu	$\alpha_s^{(n_l)}(\mu)$
mu	μ
nuf	ν_f
nl	n_l
nloops	number of loops

Returns

$m^{RS}(\nu_f)$

2.2.2.46 double CRunDec::mOS2mRSp (double *mOS,* std::pair< double, double > ∗ *mq,* double *asmu,* double *mu,* double *nuf,* int *nl,* int *nloops*)

mOS2mRSp calculates $m^{RS'}(\nu_f)$ from M_{OS}

Parameters

mOS	M_{OS}
mq	pointer to pairs of light quark masses, not active
asmu	$\alpha_s^{(n_l)}(\mu)$
mu	μ
nuf	ν_f
nl	n_l
nloops	number of loops

Returns

$m^{RS'}(\nu_f)$

2.2.2.47 double CRunDec::mOS2mSI (double *mOS,* std::pair< double, double > ∗ *mq,* double *asM,* int *nf,* int *nloops,* double *fdelm* = `1.0`)

mOS2mSI calculates $m_{MS}^{(n_f)}(m_{MS})$ from M_{OS}

Parameters

mOS	M_{OS}
mq	pointer to pairs of light quark masses
asM	$\alpha_s^{(n_f)}(M_{OS})$
nf	n_f
nloops	number of loops
fdelm	factor multiplying the non-logarithmic part of the 4-loop term

Returns

$m_{MS}^{(n_f)}(m_{MS})$

2.2.2.48 double CRunDec::mPS2mMS (double *mPS*, std::pair< double, double > ∗ *mq*, double *asmu*, double *mu*, double *muf*, int *nl*, int *nloops*, double *fdelm* = `1.0`)

mPS2mMS calculates $m_{MS}^{(n_l+1)}(\mu)$ from $m^{PS}(\mu_f)$

Parameters

mPS	$m^{PS}(\mu_f)$
mq	pointer to pairs of light quark masses, not active
asmu	$\alpha_s^{(n_l)}(\mu)$
mu	μ
muf	μ_f
nl	n_l
nloops	number of loops
fdelm	factor multiplying the non-logarithmic part of the 4-loop term

Returns

$m_{MS}^{(n_l+1)}(\mu)$

2.2.2.49 double CRunDec::mPS2mSI (double *mPS*, std::pair< double, double > ∗ *mq*, double(∗)(double) *as*, double *muf*, int *nl*, int *nloops*, double *fdelm* = `1.0`)

mPS2mSI calculates $m_{MS}^{(n_l+1)}(m_{MS})$ from $m^{PS}(\mu_f)$

Parameters

mPS	$m^{PS}(\mu_f)$
mq	pointer to pairs of light quark masses, not active
as	pointer to a function which calculates $\alpha_s^{(n_l)}(\mu)$ and takes μ as argument
muf	μ_f
nl	n_l
nloops	number of loops
fdelm	factor multiplying the non-logarithmic part of the 4-loop term

Returns

$m_{MS}^{(n_l+1)}(m_{MS})$

2.2.2.50 double CRunDec::mRGI2mMS (double *mRGI*, double *asmu*, int *nf*, int *nloops*)

mRGI2mMS calculates $m_{MS}^{(n_f)}(\mu)$ from m^{RGI}

Parameters

mRGI	m^{RGI}
asmu	$\alpha_s^{(n_f)}(\mu)$
nf	n_f
nloops	number of loops

Returns

$m_{MS}^{(n_f)}(\mu)$

2.2.2.51 double CRunDec::mRI2mMS (double *mRI,* double *asmu,* int *nf,* int *nloops*)

mRI2mMS calculates $m_{MS}^{(n_f)}(\mu)$ from $m^{RI}(\mu)$

Parameters

mRI	$m^{RI}(\mu)$
asmu	$\alpha_s^{(n_f)}(\mu)$
nf	n_f
nloops	number of loops

Returns

$m_{MS}^{(n_f)}(\mu)$

2.2.2.52 double CRunDec::mRS2mMS (double *mRS,* std::pair$<$ double, double $>$ $*$ *mq,* double *asmu,* double *mu,* double *nuf,* int *nl,* int *nloops,* double *fdelm,* bool *prime*)

mRS2mMS calculates $m_{MS}^{(n_l+1)}(\mu)$ from $m^{RS}(\nu_f)$ or $m^{RS'}(\nu_f)$

This is a helper function, use the version without the parameter prime or mRSp2mMS

Parameters

mRS	$m^{RS}(\nu_f)$ or $m^{RS'}(\nu_f)$
mq	pointer to pairs of light quark masses, not active
asmu	$\alpha_s^{(n_l)}(\mu)$
mu	μ
nuf	ν_f
nl	n_l
nloops	number of loops
fdelm	factor multiplying the non-logarithmic part of the 4-loop term
prime	selects if m^{RS} or $m^{RS'}$ is given

Returns

$m_{MS}^{(n_l+1)}(\mu)$

2.2.2.53 double CRunDec::mRS2mMS (double *mRS*, std::pair< double, double > * *mq*, double *asmu*, double *mu*, double *nuf*, int *nl*, int *nloops*, double *fdelm* = `1.0`)

mRS2mMS calculates $m_{MS}^{(n_l+1)}(\mu)$ from $m^{RS}(\nu_f)$

Parameters

mRS	$m^{RS}(\nu_f)$
mq	pointer to pairs of light quark masses, not active
asmu	$\alpha_s^{(n_l)}(\mu)$
mu	μ
nuf	ν_f
nl	n_l
nloops	number of loops
fdelm	factor multiplying the non-logarithmic part of the 4-loop term

Returns

$m_{MS}^{(n_l+1)}(\mu)$

2.2.2.54 double CRunDec::mRS2mSI (double *mRS*, std::pair< double, double > * *mq*, double(*)(double) *as*, double *nuf*, int *nl*, int *nloops*, double *fdelm*, bool *prime*)

mRS2mSI calculates $m_{MS}^{(n_l+1)}(m_{MS})$ from $m^{RS}(\nu_f)$ or $m^{RS'}(\nu_f)$

This is a helper function, use the version without the parameter prime or mRSp2mSI

Parameters

mRS	$m^{RS}(\nu_f)$ or $m^{RS'}(\nu_f)$
mq	pointer to pairs of light quark masses, not active
as	pointer to a function which calculates $\alpha_s^{(n_l)}(\mu)$ and takes μ as argument
nuf	ν_f
nl	n_l
nloops	number of loops
fdelm	factor multiplying the non-logarithmic part of the 4-loop term
prime	selects if m^{RS} or $m^{RS'}$ is given

Returns

$m_{MS}^{(n_l+1)}(m_{MS})$

2.2.2.55 double CRunDec::mRS2mSI (double *mRS*, std::pair$<$ double, double $>$ $*$ *mq*, double($*$)(double) *as*, double *nuf*, int *nl*, int *nloops*, double *fdelm* = `1.0`)

mRS2mSI calculates $m_{MS}^{(n_l+1)}(m_{MS})$ from $m^{RS}(\nu_f)$

Parameters

mRS	$m^{RS}(\nu_f)$
mq	pointer to pairs of light quark masses, not active
as	pointer to a function which calculates $\alpha_s^{(n_l)}(\mu)$ and takes μ as argument
nuf	ν_f
nl	n_l
nloops	number of loops
fdelm	factor multiplying the non-logarithmic part of the 4-loop term

Returns

$m_{MS}^{(n_l+1)}(m_{MS})$

2.2.2.56 double CRunDec::mRSp2mMS (double *mRS*, std::pair$<$ double, double $>$ $*$ *mq*, double *asmu*, double *mu*, double *nuf*, int *nl*, int *nloops*, double *fdelm* = `1.0`)

mRSp2mMS calculates $m_{MS}^{(n_l+1)}(\mu)$ from $m^{RS'}(\nu_f)$

Parameters

mRS	$m^{RS'}(\nu_f)$
mq	pointer to pairs of light quark masses, not active
asmu	$\alpha_s^{(n_l)}(\mu)$
mu	μ
nuf	ν_f
nl	n_l
nloops	number of loops
fdelm	factor multiplying the non-logarithmic part of the 4-loop term

Returns

$m_{MS}^{(n_l+1)}(\mu)$

2.2.2.57 double CRunDec::mRSp2mSI (double *mRS*, std::pair$<$ double, double $>$ $*$ *mq*, double($*$)(double) *as*, double *nuf*, int *nl*, int *nloops*, double *fdelm* = `1.0`)

mRSp2mSI calculates $m_{MS}^{(n_l+1)}(m_{MS})$ from $m^{RS'}(\nu_f)$

Parameters

mRS	$m^{RS'}(\nu_f)$
mq	pointer to pairs of light quark masses, not active
as	pointer to a function which calculates $\alpha_s^{(n_l)}(\mu)$ and takes μ as argument
nuf	ν_f
nl	n_l
nloops	number of loops
fdelm	factor multiplying the non-logarithmic part of the 4-loop term

Returns

$m_{MS}^{(n_l+1)}(m_{MS})$

2.2.2.58 void CRunDec::SetNf (int *nf*)

SetNf sets the number of light flavours.

Parameters

nf	n_f

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

- CRunDec.h
- CRunDec.cpp

2.3 TriplenfMmu Struct Reference

Structure containing: the number of light flavours (nf), the mass of the heavy quark (Mth) and the decoupling scale (muth)

```
#include <CRunDec.h>
```

Public Attributes

- int **nf**
- double **Mth**
- double **muth**

2.3.1 Detailed Description

Structure containing: the number of light flavours (nf), the mass of the heavy quark (Mth) and the decoupling scale (muth)

This structure is used to pass information on decoupling thresholds to AlH2AlL, AlL2AlH, mH2mL and mL2mH.

The documentation for this struct was generated from the following file:

- CRunDec.h

Index