

CRunDec3

Generated by Doxygen 1.8.11

Contents

1	Class Index	1
1.1	Class List	1
2	Class Documentation	3
2.1	AsmMS Struct Reference	3
2.1.1	Detailed Description	3
2.2	CRunDec Class Reference	3
2.2.1	Detailed Description	7
2.2.2	Member Function Documentation	8
2.2.2.1	AIH2AIL(double ash, double mu1, TriplenfMmu decpar[], double mu2, int nloops)	8
2.2.2.2	AIL2AIH(double asl, double mu1, TriplenfMmu decpar[], double mu2, int nloops)	9
2.2.2.3	AlphasExact(double asmu0, double mu0, double mu1, int nf, int nloops)	9
2.2.2.4	AlphasLam(double Lambda, double mu, int nf, int nloops)	10
2.2.2.5	AsmMSrunexact(double mmu, double asmu0, double mu0, double mu1, int nf, int nloops)	10
2.2.2.6	DecAsDownMS(double asmu, double massth, double muth, int nl, int nloops)	10
2.2.2.7	DecAsDownOS(double asmu, double massth, double muth, int nl, int nloops)	11
2.2.2.8	DecAsDownSI(double asmu, double massth, double muth, int nl, int nloops)	11
2.2.2.9	DecAsUpMS(double asmu, double massth, double muth, int nl, int nloops)	11
2.2.2.10	DecAsUpOS(double asmu, double massth, double muth, int nl, int nloops)	12
2.2.2.11	DecAsUpSI(double asmu, double massth, double muth, int nl, int nloops)	12
2.2.2.12	DecLambdaDown(double lam, double massth, int nl, int nloops)	12
2.2.2.13	DecLambdaUp(double lam, double massth, int nl, int nloops)	13
2.2.2.14	DecMqDownMS(double mq, double asmu, double massth, double muth, int nl, int nloops)	13

2.2.2.15	DecMqDownOS(double mq, double asmu, double massth, double muth, int nl, int nloops)	13
2.2.2.16	DecMqDownSI(double mq, double asmu, double massth, double muth, int nl, int nloops)	14
2.2.2.17	DecMqUpMS(double mq, double asmu, double massth, double muth, int nl, int nloops)	14
2.2.2.18	DecMqUpOS(double mq, double asmu, double massth, double muth, int nl, int nloops)	15
2.2.2.19	DecMqUpSI(double mq, double asmu, double massth, double muth, int nl, int nloops)	15
2.2.2.20	GetNf()	15
2.2.2.21	LamExpl(double asmu, double mu, int nf, int nloops)	16
2.2.2.22	LamImpl(double asmu, double mu, int nf, int nloops)	16
2.2.2.23	m1S2mMS(double m1S, std::pair< double, double > *mq, double asmu, double mu, int nl, int nloops, double fdelm=1.0)	16
2.2.2.24	m1S2mSI(double m1S, std::pair< double, double > *mq, double(*as)(double), int nl, int nloops, double fdelm=1.0)	17
2.2.2.25	mH2mL(double mqh, double ash, double mu1, TriplenfMmu decpar[], double mu2, int nloops)	17
2.2.2.26	mL2mH(double mql, double asl, double mu1, TriplenfMmu decpar[], double mu2, int nloops)	17
2.2.2.27	mMS2m1S(double mMS, std::pair< double, double > *mq, double asmu, double mu, int nl, int nloops, double fdelm=1.0)	18
2.2.2.28	mMS2mMS(double mu0, double asmu1, double asmu0, int nf, int nloops)	18
2.2.2.29	mMS2mOS(double mMS, std::pair< double, double > *mq, double asmu, double mu, int nf, int nloops, double fdelm=1.0)	19
2.2.2.30	mMS2mOSrun(double mMS, std::pair< double, double > *mq, double asmu, double mu, int nf, int nloops)	19
2.2.2.31	mMS2mPS(double mMS, std::pair< double, double > *mq, double asmu, double mu, double muf, int nl, int nloops, double fdelm=1.0)	20
2.2.2.32	mMS2mRGI(double mMS, double asmu, int nf, int nloops)	20
2.2.2.33	mMS2mRGImod(double mMS, double asmu, int nf, int nloops)	20
2.2.2.34	mMS2mRI(double mMS, double asmu, int nf, int nloops)	21
2.2.2.35	mMS2mRS(double mMS, std::pair< double, double > *mq, double asmu, double mu, double nuf, int nl, int nloops, double fdelm, bool prime)	21
2.2.2.36	mMS2mRS(double mMS, std::pair< double, double > *mq, double asmu, double mu, double nuf, int nl, int nloops, double fdelm=1.0)	22

2.2.2.37	mMS2mRSp(double mMS, std::pair< double, double > *mq, double asmu, double mu, double nuf, int nl, int nloops, double fdelm=1.0)	22
2.2.2.38	mMS2mSI(double mMS, double asmu, double mu, int nf, int nloops)	23
2.2.2.39	mOS2m1S(double mOS, std::pair< double, double > *mq, double asmu, double mu, int nl, int nloops)	23
2.2.2.40	mOS2mMS(double mOS, std::pair< double, double > *mq, double asmu, double mu, int nf, int nloops, double fdelm=1.0)	23
2.2.2.41	mOS2mMSit(double mOS, std::pair< double, double > *mq, double asmu, double mu, int nf, int nloops)	24
2.2.2.42	mOS2mMSrun(double mOS, std::pair< double, double > *mq, double asmu, double mu, int nf, int nloops)	24
2.2.2.43	mOS2mPS(double mOS, std::pair< double, double > *mq, double asmu, double mu, double muf, int nl, int nloops)	25
2.2.2.44	mOS2mRS(double mOS, std::pair< double, double > *mq, double asmu, double mu, double nuf, int nl, int nloops, bool prime)	25
2.2.2.45	mOS2mRS(double mOS, std::pair< double, double > *mq, double asmu, double mu, double nuf, int nl, int nloops)	25
2.2.2.46	mOS2mRSp(double mOS, std::pair< double, double > *mq, double asmu, double mu, double nuf, int nl, int nloops)	26
2.2.2.47	mOS2mSI(double mOS, std::pair< double, double > *mq, double asM, int nf, int nloops, double fdelm=1.0)	26
2.2.2.48	mPS2mMS(double mPS, std::pair< double, double > *mq, double asmu, double mu, double muf, int nl, int nloops, double fdelm=1.0)	27
2.2.2.49	mPS2mSI(double mPS, std::pair< double, double > *mq, double(*as)(double), double muf, int nl, int nloops, double fdelm=1.0)	27
2.2.2.50	mRGI2mMS(double mRGI, double asmu, int nf, int nloops)	27
2.2.2.51	mRI2mMS(double mRI, double asmu, int nf, int nloops)	28
2.2.2.52	mRS2mMS(double mRS, std::pair< double, double > *mq, double asmu, double mu, double nuf, int nl, int nloops, double fdelm, bool prime)	28
2.2.2.53	mRS2mMS(double mRS, std::pair< double, double > *mq, double asmu, double mu, double nuf, int nl, int nloops, double fdelm=1.0)	29
2.2.2.54	mRS2mSI(double mRS, std::pair< double, double > *mq, double(*as)(double), double nuf, int nl, int nloops, double fdelm, bool prime)	29
2.2.2.55	mRS2mSI(double mRS, std::pair< double, double > *mq, double(*as)(double), double nuf, int nl, int nloops, double fdelm=1.0)	30
2.2.2.56	mRSp2mMS(double mRS, std::pair< double, double > *mq, double asmu, double mu, double nuf, int nl, int nloops, double fdelm=1.0)	30
2.2.2.57	mRSp2mSI(double mRS, std::pair< double, double > *mq, double(*as)(double), double nuf, int nl, int nloops, double fdelm=1.0)	30
2.2.2.58	SetNf(int nf)	31
2.3	TriplenfMmu Struct Reference	31
2.3.1	Detailed Description	31

Chapter 1

Class Index

1.1 Class List

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

AsmMS	Structure containing: $\alpha_s^{(n_f)}$ (Asexact) and $m_{MS}^{(n_f)}$ (mMSexact)	3
CRunDec	Main class, contains all functions	3
TriplenMmu	Structure containing: the number of light flavours (nf), the mass of the heavy quark (Mth) and the decoupling scale (muth)	31

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 $\Lambda^{(nf)}$ from $\alpha_s^{(nf)}$ explicitly solving for $\Lambda^{(nf)}$.
- double **LamImpl** (double asmu, double mu, int nf, int nloops)
LamImpl calculates $\Lambda^{(nf)}$ from $\alpha_s^{(nf)}$ implicitly solving for $\Lambda^{(nf)}$.
- double **AlphasLam** (double Lambda, double mu, int nf, int nloops)
AlphasLam calculates $\alpha_s^{(nf)}$ from $\Lambda^{(nf)}$.
- double **AlphasExact** (double asmu0, double mu0, double mu1, int nf, int nloops)
AlphasExact calculates $\alpha_s^{(nf)}(\mu_1)$ from $\alpha_s^{(nf)}(\mu_0)$.
- double **mMS2mMS** (double mu0, double asmu1, double asmu0, int nf, int nloops)
mMS2mMS calculates $m_{MS}^{(nf)}(\mu_1)$ from $m_{MS}^{(nf)}(\mu_0)$.
- **AsmMS AsmMSrunexact** (double mmu, double asmu0, double mu0, double mu1, int nf, int nloops)
AsmMSrunexact solves simultaneously the differential equations for $\alpha_s^{(nf)}$ and $m_{MS}^{(nf)}$.
- 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 **AIL2AIH** (double asl, double mu1, **TriplenfMmu** decpar[], double mu2, int nloops)
AIL2AIH calculates $\alpha_s(\mu_2)$ from $\alpha_s(\mu_1)$ decoupling at intermediate scales, running from low to high.
- double **AIH2AIL** (double ash, double mu1, **TriplenfMmu** decpar[], double mu2, int nloops)

- AIH2AIL 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)

- 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::AIH2AIL (double ash, double mu1, TriplenfMmu decpa[], double mu2, int nloops)`

AIH2AIL calculates $\alpha_s(\mu_2)$ from $\alpha_s(\mu_1)$ decoupling at intermediate scales, running from high to low.

Parameters

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

Returns

$$\alpha_s(\mu_2)$$

2.2.2.2 `double CRunDec::AIL2AIH (double asl, double mu1, TriplenfMmu decpa[], double mu2, int nloops)`

AIL2AIH calculates $\alpha_s(\mu_2)$ from $\alpha_s(\mu_1)$ decoupling at intermediate scales, running from low to high.

Parameters

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

<i>asmu0</i>	$\alpha_s^{(n_f)}(\mu_0)$
<i>mu0</i>	μ_0
<i>mu1</i>	μ_1
<i>nf</i>	n_f
<i>nloops</i>	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

<i>Lambda</i>	$\Lambda^{(n_f)}$
<i>mu</i>	μ
<i>nf</i>	n_f
<i>nloops</i>	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

<i>mmu</i>	$m_{MS}^{(n_f)}(\mu_0)$
<i>asmu0</i>	$\alpha_s^{(n_f)}(\mu_0)$
<i>mu0</i>	μ_0
<i>mu1</i>	μ_1
<i>nf</i>	n_f
<i>nloops</i>	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

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

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

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

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

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

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

<i>lam</i>	$\Lambda^{(n_l+1)}$
<i>massth</i>	quark mass at which the matching is performed ($m_{MS}^{(n_l+1)}(m_{MS})$)
<i>nl</i>	n_l
<i>nloops</i>	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

<i>lam</i>	$\Lambda^{(n_l)}$
<i>massth</i>	quark mass at which the matching is performed ($m_{MS}^{(n_l+1)}(m_{MS})$)
<i>nl</i>	n_l
<i>nloops</i>	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

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

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

Parameters

<i>massth</i>	mass of the heavy quark (M_{OS})
<i>muth</i>	scale μ at which the matching is performed
<i>nl</i>	n_l
<i>nloops</i>	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

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

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

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

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

<i>asmu</i>	$\alpha_s^{(n_f)}(\mu)$
<i>mu</i>	μ
<i>nf</i>	n_f
<i>nloops</i>	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

<i>asmu</i>	$\alpha_s^{(n_f)}(\mu)$
<i>mu</i>	μ
<i>nf</i>	n_f
<i>nloops</i>	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

<i>m1S</i>	m^{PS}
<i>mq</i>	pointer to pairs of light quark masses, not active
<i>asmu</i>	$\alpha_s^{(n_l)}(\mu)$
<i>mu</i>	μ
<i>nl</i>	n_l
<i>nloops</i>	number of loops
<i>fdelm</i>	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

<i>m1S</i>	m^{1S}
<i>mq</i>	pointer to pairs of light quark masses, not active
<i>as</i>	pointer to a function which calculates $\alpha_s^{(n_l)}(\mu)$ and takes μ as argument
<i>nl</i>	n_l
<i>nloops</i>	number of loops
<i>fdelm</i>	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

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

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

<i>mMS</i>	$m_{MS}^{(n_l+1)}(\mu)$
<i>mq</i>	pointer to pairs of light quark masses, not active
<i>asmu</i>	$\alpha_s^{(n_l)}(\mu)$
<i>mu</i>	μ
<i>nl</i>	n_l
<i>nloops</i>	number of loops
<i>fdelm</i>	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

<i>mu0</i>	$m_{MS}^{(n_f)}(\mu_0)$
<i>asmu0</i>	$\alpha_s^{(n_f)}(\mu_0)$
<i>asmu1</i>	$\alpha_s^{(n_f)}(\mu_1)$
<i>nf</i>	n_f
<i>nloops</i>	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

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

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

<i>mMS</i>	$m_{MS}^{(n_l+1)}(\mu)$
<i>mq</i>	pointer to pairs of light quark masses, not active
<i>asmu</i>	$\alpha_s^{(n_l)}(\mu)$
<i>mu</i>	μ
<i>muf</i>	μ_f
<i>nl</i>	n_l
<i>nloops</i>	number of loops
<i>fdelm</i>	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

<i>mMS</i>	$m_{MS}^{(n_f)}(\mu)$
<i>asmu</i>	$\alpha_s^{(n_f)}(\mu)$
<i>nf</i>	n_f
<i>nloops</i>	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

<i>mMS</i>	$m_{MS}^{(n_f)}(\mu)$
<i>asmu</i>	$\alpha_s^{(n_f)}(\mu)$
<i>nf</i>	n_f
<i>nloops</i>	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

<i>mMS</i>	$m_{MS}^{(n_f)}(\mu)$
<i>asmu</i>	$\alpha_s^{(n_f)}(\mu)$
<i>nf</i>	n_f
<i>nloops</i>	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 mMS2mRS

Parameters

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

Returns

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

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

<i>mMS</i>	$m_{MS}^{(n_l+1)}(\mu)$
<i>mq</i>	pointer to pairs of light quark masses, not active
<i>asmu</i>	$\alpha_s^{(n_l)}(\mu)$
<i>mu</i>	μ
<i>nuf</i>	ν_f
<i>nl</i>	n_l
<i>nloops</i>	number of loops
<i>fdelm</i>	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

<i>mMS</i>	$m_{MS}^{(n_l+1)}(\mu)$
<i>mq</i>	pointer to pairs of light quark masses, not active
<i>asmu</i>	$\alpha_s^{(n_l)}(\mu)$
<i>mu</i>	μ
<i>nuf</i>	ν_f
<i>nl</i>	n_l
<i>nloops</i>	number of loops
<i>fdelm</i>	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

<i>mMS</i>	$m_{MS}^{(n_f)}(\mu)$
<i>asmu</i>	$\alpha_s^{(n_f)}(\mu)$
<i>mu</i>	μ
<i>nf</i>	n_f
<i>nloops</i>	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

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

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

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

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

<i>mOS</i>	M_{OS}
<i>mq</i>	pointer to pairs of light quark masses, not active
<i>asmu</i>	$\alpha_s^{(n_l)}(\mu)$
<i>mu</i>	μ
<i>muf</i>	μ_f
<i>nl</i>	n_l
<i>nloops</i>	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 mOS2mRS

Parameters

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

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

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

<i>mOS</i>	M_{OS}
<i>mq</i>	pointer to pairs of light quark masses
<i>asM</i>	$\alpha_s^{(n_f)}(M_{OS})$
<i>nf</i>	n_f
<i>nloops</i>	number of loops
<i>fdelm</i>	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

<i>mPS</i>	$m^{PS}(\mu_f)$
<i>mq</i>	pointer to pairs of light quark masses, not active
<i>asmu</i>	$\alpha_s^{(n_l)}(\mu)$
<i>mu</i>	μ
<i>muf</i>	μ_f
<i>nl</i>	n_l
<i>nloops</i>	number of loops
<i>fdelm</i>	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

<i>mPS</i>	$m^{PS}(\mu_f)$
<i>mq</i>	pointer to pairs of light quark masses, not active
<i>as</i>	pointer to a function which calculates $\alpha_s^{(n_l)}(\mu)$ and takes μ as argument
<i>muf</i>	μ_f
<i>nl</i>	n_l
<i>nloops</i>	number of loops
<i>fdelm</i>	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

<i>mRGI</i>	m^{RGI}
<i>asmu</i>	$\alpha_s^{(n_f)}(\mu)$
<i>nf</i>	n_f
<i>nloops</i>	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

<i>mRI</i>	$m^{RI}(\mu)$
<i>asmu</i>	$\alpha_s^{(n_f)}(\mu)$
<i>nf</i>	n_f
<i>nloops</i>	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

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

<i>mRS</i>	$m^{RS}(\nu_f)$
<i>mq</i>	pointer to pairs of light quark masses, not active
<i>asmu</i>	$\alpha_s^{(n_l)}(\mu)$
<i>mu</i>	μ
<i>nuf</i>	ν_f
<i>nl</i>	n_l
<i>nloops</i>	number of loops
<i>fdelm</i>	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

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

<i>mRS</i>	$m^{RS}(\nu_f)$
<i>mq</i>	pointer to pairs of light quark masses, not active
<i>as</i>	pointer to a function which calculates $\alpha_s^{(n_l)}(\mu)$ and takes μ as argument
<i>nuf</i>	ν_f
<i>nl</i>	n_l
<i>nloops</i>	number of loops
<i>fdelm</i>	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

<i>mRS</i>	$m^{RS'}(\nu_f)$
<i>mq</i>	pointer to pairs of light quark masses, not active
<i>asmu</i>	$\alpha_s^{(n_l)}(\mu)$
<i>mu</i>	μ
<i>nuf</i>	ν_f
<i>nl</i>	n_l
<i>nloops</i>	number of loops
<i>fdelm</i>	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

<i>mRS</i>	$m^{RS'}(\nu_f)$
<i>mq</i>	pointer to pairs of light quark masses, not active
<i>as</i>	pointer to a function which calculates $\alpha_s^{(n_l)}(\mu)$ and takes μ as argument
<i>nuf</i>	ν_f
<i>nl</i>	n_l
<i>nloops</i>	number of loops
<i>fdelm</i>	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

<i>nf</i>	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 AIH2AIL, AIL2AIH, mH2mL and mL2mH.

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

- CRunDec.h

Index

AIH2AIL
 CRunDec, [8](#)
AIL2AIH
 CRunDec, [9](#)
AlphasExact
 CRunDec, [9](#)
AlphasLam
 CRunDec, [9](#)
AsmMSrunexact
 CRunDec, [10](#)
AsmMS, [3](#)

CRunDec, [3](#)
 AIH2AIL, [8](#)
 AIL2AIH, [9](#)
 AlphasExact, [9](#)
 AlphasLam, [9](#)
 AsmMSrunexact, [10](#)
 DecAsDownMS, [10](#)
 DecAsDownOS, [11](#)
 DecAsDownSI, [11](#)
 DecAsUpMS, [11](#)
 DecAsUpOS, [12](#)
 DecAsUpSI, [12](#)
 DecLambdaDown, [12](#)
 DecLambdaUp, [13](#)
 DecMqDownMS, [13](#)
 DecMqDownOS, [13](#)
 DecMqDownSI, [14](#)
 DecMqUpMS, [14](#)
 DecMqUpOS, [15](#)
 DecMqUpSI, [15](#)
 GetNf, [15](#)
 LamExpl, [15](#)
 LamImpl, [16](#)
 m1S2mMS, [16](#)
 m1S2mSI, [17](#)
 mH2mL, [17](#)
 mL2mH, [17](#)
 mMS2m1S, [18](#)
 mMS2mMS, [18](#)
 mMS2mOSrun, [19](#)
 mMS2mOS, [19](#)
 mMS2mPS, [19](#)
 mMS2mRGImod, [20](#)
 mMS2mRGI, [20](#)
 mMS2mRSp, [22](#)
 mMS2mRI, [21](#)
 mMS2mRS, [21](#), [22](#)
 mMS2mSI, [22](#)

 mOS2m1S, [23](#)
 mOS2mMSit, [24](#)
 mOS2mMSrun, [24](#)
 mOS2mMS, [23](#)
 mOS2mPS, [24](#)
 mOS2mRSp, [26](#)
 mOS2mRS, [25](#)
 mOS2mSI, [26](#)
 mPS2mMS, [27](#)
 mPS2mSI, [27](#)
 mRGI2mMS, [27](#)
 mRI2mMS, [28](#)
 mRS2mMS, [28](#), [29](#)
 mRS2mSI, [29](#)
 mRSp2mMS, [30](#)
 mRSp2mSI, [30](#)
 SetNf, [31](#)

DecAsDownMS
 CRunDec, [10](#)
DecAsDownOS
 CRunDec, [11](#)
DecAsDownSI
 CRunDec, [11](#)
DecAsUpMS
 CRunDec, [11](#)
DecAsUpOS
 CRunDec, [12](#)
DecAsUpSI
 CRunDec, [12](#)
DecLambdaDown
 CRunDec, [12](#)
DecLambdaUp
 CRunDec, [13](#)
DecMqDownMS
 CRunDec, [13](#)
DecMqDownOS
 CRunDec, [13](#)
DecMqDownSI
 CRunDec, [14](#)
DecMqUpMS
 CRunDec, [14](#)
DecMqUpOS
 CRunDec, [15](#)
DecMqUpSI
 CRunDec, [15](#)

GetNf
 CRunDec, [15](#)

LamExpl
 CRunDec, [15](#)
 LamImpl
 CRunDec, [16](#)

 m1S2mMS
 CRunDec, [16](#)
 m1S2mSI
 CRunDec, [17](#)
 mH2mL
 CRunDec, [17](#)
 mL2mH
 CRunDec, [17](#)
 mMS2m1S
 CRunDec, [18](#)
 mMS2mMS
 CRunDec, [18](#)
 mMS2mOSrun
 CRunDec, [19](#)
 mMS2mOS
 CRunDec, [19](#)
 mMS2mPS
 CRunDec, [19](#)
 mMS2mRGImod
 CRunDec, [20](#)
 mMS2mRGI
 CRunDec, [20](#)
 mMS2mRSp
 CRunDec, [22](#)
 mMS2mRI
 CRunDec, [21](#)
 mMS2mRS
 CRunDec, [21](#), [22](#)
 mMS2mSI
 CRunDec, [22](#)
 mOS2m1S
 CRunDec, [23](#)
 mOS2mMSit
 CRunDec, [24](#)
 mOS2mMSrun
 CRunDec, [24](#)
 mOS2mMS
 CRunDec, [23](#)
 mOS2mPS
 CRunDec, [24](#)
 mOS2mRSp
 CRunDec, [26](#)
 mOS2mRS
 CRunDec, [25](#)
 mOS2mSI
 CRunDec, [26](#)
 mPS2mMS
 CRunDec, [27](#)
 mPS2mSI
 CRunDec, [27](#)
 mRGI2mMS
 CRunDec, [27](#)
 mRI2mMS
 CRunDec, [28](#)

 mRS2mMS
 CRunDec, [28](#), [29](#)
 mRS2mSI
 CRunDec, [29](#)
 mRSp2mMS
 CRunDec, [30](#)
 mRSp2mSI
 CRunDec, [30](#)

 SetNf
 CRunDec, [31](#)

 TriplenfMmu, [31](#)