```
1   /*****************************************************************************
2    * Package: GSM                                                              *
3    * Class  : WMass                                                            *
4    *                                                                           *
5    * Description:                                                              *
6    *      Prediction of W mass                                                 *
7    *                                                                           *
8    * Papers:                                                                   *
9    *      M. Awramik, M. Czakon, A. Freitas, G. Weiglein,                      *
10   *                  Phys. Rev. D69:053006, 2004, hep-ph/0311148.             *
11   *                                                                           *
12   *****************************************************************************
13  #include "TMath.h"
14  #include "TString.h"
15
16  #include "Gfitter/GMath.h"
17  #include "Gfitter/GTheory.h"
18  #include "Gfitter/GTheoryRef.h"
19  #include "Gfitter/GParameterRef.h"
20  #include "Gfitter/GReference.h"
21  #include "Gfitter/GVariable.h"
22  #include "Gfitter/GStore.h"
23  #include "Gfitter/GConstants.h"
24
25  #include "GSM/WMass.h"
26  #include "GSM/DAlphaQED.h"
27  #include "GSM/AlphaQCDAtQ.h"
28  #include "GSM/MH.h"
29
30  using namespace Gfitter;
31
32  ClassImp(GSM::WMass)
33
34  GSM::WMass::WMass(): Gfitter::GAuxTheory()
35  {
```

Hinweis:

Kommentare mit Hinweisen auf ZFitter sind grün markiert

Übereinstimmungen sind gelb markiert.

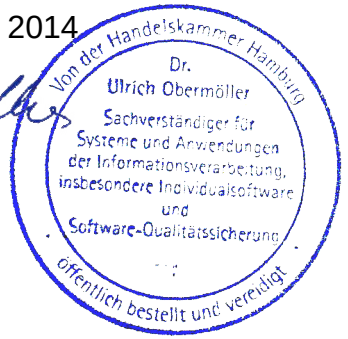
Auffällige Stellen bzw. Anmerkungen sind violett markiert

Anhang 11

zum Gutachten DESY ZFitter_GFitter vom 17.03.2014

Lübeck, den 17. März 2014

Dr. Ulrich Obermöller
-Sachverständiger-



```
36      SetTheoryName( GetName() );
37      SetExistDerivative( kFALSE );
38
39      const TString& logMH = gStore()->GetVariable( "GSMFlags::logMH" )->GetStringValue();
40      m_logger << kINFO << "Using logMH: \"" << logMH << "\"" << GEndl;
41
42      if      (logMH == "Yes" ) m_logMH  = kTRUE;
43      else if (logMH == "No" )  m_logMH  = kFALSE;
44      else {
45        m_logger << kFATAL << "unknown value for \"GSMFlags::logMH\": \"" << logMH << "\""
46                 << ". Possible are: \"Yes\" and \"No\"\""
47                 << GEndl;
48      }
49
50      BookParameter( "MZ",                &p_MZ );
51      BookParameter( "mt",                &p_mt );
52      BookParameter( "DeltaMW_Scale",     &p_DeltaMW_Scale );
53      BookParameter( "GF",                &p_GF );
54      BookTheory   ( "GSM::RunningAlphaQCD", &t_AlphasRun );
55      BookTheory   ( "GSM::DAlphaQED",    &t_DAlphaQED );
56      BookTheory   ( "GSM::MH",           &t_MH );
57    }
58
59    // init coefficients
60    void GSM::WMass::Initialise()
61    {
62      m_MW0 = 80.3799;
63      m_c1  = 0.05429;
64      m_c2  = 0.008939;
65      m_c3  = 0.0000890;
66      m_c4  = 0.000161;
67      m_c5  = 1.070;
68      m_c6  = 0.5256;
69      m_c7  = 0.0678;
70      m_c8  = 0.00179;
```

Match 1

```
71    m_c9  = 0.0000659;
72    m_c10 = 0.0737;
73    m_c11 = 114.9;
74  }
75
76  // eq. (6) of hep-ph/0311148
77  Double_t GSM::WMass::GetValue()
78  {
79    Initialise();
80
81    Double_t MH      = GetMH().GetValue(); //p_MH;
82    if( m_logMH ) MH = TMath::Exp( GetMH().GetValue() ); //p_MH );
83
84    Double_t deAlpha     = (GetDAlphaQED().DAlphaQEDMZ()/0.05907) - 1.0;
85    Double_t deAlphaS    = (GetAlphasRun().EvolveAlphas(p_MZ)/0.119) - 1.0;
86
87    Double_t LH          = TMath::Log(MH/100);
88    Double_t dH          = Gfitter::GMath::IPow( MH/100.0 , 2);
89
90    Double_t dt          = ((p_mt/174.3)*(p_mt/174.3) - 1.0);
91    Double_t dZ          = (p_MZ/91.1875) - 1.0;
92
93    Double_t mw = ( m_MW0 - m_c1*LH - m_c2*LH*LH + m_c3*Gfitter::GMath::IPow(LH,4)
94                  + m_c4*(dH-1) - m_c5*deAlpha
95                  + m_c6*dt - m_c7*dt*dt - m_c8*LH*dt
96                  + m_c9*dH*dt - m_c10*deAlphaS + m_c11*dZ );
97
98    // theoretical uncertainty in MW
99    mw = mw + p_DeltaMW_Scale*0.004;
100
101   // test GF dependency of GF
102   if( false ){
103     Double_t MZ2 = p_MZ*p_MZ;
104
105     Double_t deltaMW2 = ( MZ2/2.*( TMath::Sqrt(1-TMath::Sqrt(8.)*TMath::Pi()*GConstants::alphaQED()/(MZ2*GConstants::GF())))
```

4.2.3.3

dizet6_42.f

```
1687  *- M_w block:
1688  *--
1689        DAL5HH=CDAL5H
1690        DALPHA=AL4PI*DREAL(XFOTF3(IALEM,IALE2,IHVP,1,0,DAL5HH,-AMZ2))
1691        DELALP=DALPHA/0.05907D0-1D0
1692        DELALS=CALSZ/0.119D0-1D0
1693        DELHIG=LOG(AMH/100D0)
1694      DELHSQ=(AMH/100D0)**2
1695        DELTOP=(AMT/174.3D0)**2-1D0
1696        DELAMZ=(AMZ/91.1875D0)-1D0
1697  *
1698  * Improved expansion including MZ, MT**4 and MH**2 dependence
1699  * result for complete 2-loop corrections as of Nov. 2003:
1700  *
1701        AMW0=80.3799D0
1702        C1   = 0.05429D0
1703        C2   = 0.008939D0
1704      C3   = 0.0000890D0
1705      C4   = 0.000161D0
1706      C5   = 1.070D0
1707      C6   = 0.5256D0
1708      C7   = 0.0678D0
1709      C8   = 0.00179D0
1710      C9   = 0.0000659D0
1711      C10  = 0.0737D0
1712      C11  = 114.9D0
1713  *
1714        AMWNEW=AMW0-C1*DELHIG-C2*DELHIG**2+C3*DELHIG**4+C4*(DELHSQ-1D0)
1715       &             +C6*DELTOP-C7*DELTOP**2-C8*DELHIG*DELTOP+C9*DELHSQ*DELTOP
1716       &             -C5*DELALP-C10*DELALS+C11*DELAMZ
```

```
106                                     - TMath::Sqrt(1-TMath::Sqrt(8.)*TMath::Pi()*GConstants::alphaQED()/(MZ2*p_GF)) ) );
107          mw = TMath::Sqrt(mw*mw + deltaMW2);
108
109        }
110
111        return mw;
112      }
```