Generating Feynman Graphs and Amplitudes with *FeynArts*

Hagen Eck
Sepp Küblbeck
Physikalisches Institut
Am Hubland
D-8700 Würzburg

January 1991

Contents

0 Introduction

In high energy physics the interaction of matter and its mediating forces is successfully described in consistent field theoretical models. For the evaluation of these theories perturbative methods have become well-established. A systematic and very beautiful approach to perturbation theory was provided by R. Feynman, who introduced the concept of Feynman graphs and Feynman rules. Physical observables like cross sections or decay rates are expressed in terms of Feynman graphs in a very concise way. These graphs can be classified with one look and allow the organization of complex calculations such as the determination of higher order corrections with a renormalization procedure. However, in a non-trivial model a calculation beyond tree level quickly becomes tedious and very susceptible to errors. This is valid for finding all Feynman graphs and amplitudes, and even more for the evaluation of the amplitudes. Therefore many efforts have been made to automatize this procedure with computer algebra programs.

FeynArts is a package for users in high energy physics who want to calculate differential cross sections and decay rates for processes in field theories using the approach with Feynman graphs. The package creates Feynman graphs and analytical expressions (amplitudes) for S-matrix elements and truncated Green functions in renormalizable quantum field theories. It consists of several sub-programs each of which is written as general as possible. Programmed in *Mathematica* [1] *FeynArts* can generate multi-loop topologies, insert fields into them and is not restricted to a certain model (it only has to be renormalizable). *FeynArts* is based on algorithms that can handle complex models and it has sophisticated graphics facilities.

FeynArts produces graphical and analytical output. The graphics output are usual *Mathematica* graphics objects that can be printed on screen or printer or saved in a graphics file. The analytical output is a direct translation of generated Feynman graphs according to the given Feynman rules of a model. Since — in order to produce numerical results — further evaluation of these expressions is necessary, the analytical output is given in a form suitable for direct evalutation with the program package *FeynCalc* [5].

Anyone who knows the basic principles of *Mathematica*, e.g. how to assign an expression to a variable, will be able to produce graphs and amplitudes. Those who are familiar with *Mathematica* can manipulate the generated output of each function with standard methods (selecting subsets of lists, join results of different calculations etc.). Entering a new model already requires some routine in using *Mathematica*. In particular the use of the 'blank' (written `_`) in patterns should be known.

In this manual we use the same conventions as in the *Mathematica* manual:

> Text in `Typewriter` style is literal input which has to be typed as printed (e.g. function names).
>
> Text in *Italics* are expressions, the user has to fill in (e.g. function arguments).

Earlier versions of *FeynArts* and its algorithms have already been described in [2, 3]. Part of this manual extended by the description of the insertion algorithm, tests and CPU-times for the program as well as further references is contained in [4] and will also be submitted to Computer Physics Communications.

■ 0.1 The Structure of *FeynArts*

Programming *FeynArts* we tried to follow as closely as possible the procedure high energy physicists are familiar with. In this procedure one usually deals with a process a) with given external particles b) in a certain order of perturbation theory c) within an appropriate model. With these premises all Feynman graphs with non-vanishing couplings have to be found.

The creation of the Feynman graphs and amplitudes is performed by *FeynArts* if the Feynman rules of the model(s) underlying the calculations are provided in an appropriate model file. For the full standard model (electroweak and QCD) and for a few submodels the Feynman rules are supplied with the package, other models can be entered by the user. To obtain contributions with three or more loops the corresponding start topologies have to be entered by the user following the instructions in section 1.2.

After loading the package in a *Mathematica* session with `<<FeynArts.m` or within the `init.m` file[1] (see installation) the following functions perform the necessary operations to obtain Feynman graphs and amplitudes:

1. First the desired set of topologies must be generated with `CreateTopologies`.
2. With `InsertFields` the lines of the topologies are occupied with particles thus creating the Feynman graphs in list form.
3. The Feynman graphs can be drawn on screen or paper by applying `Paint` to the output of `InsertFields`.
4. `CreateFeynAmp` applied on these graphs yields the corresponding analytic expressions according to the given set of Feynman rules.

For each of the four *FeynArts*-functions — like for many *Mathematica* functions — `Options` can be given. The default options of a function are displayed with `Options[`*function*`]` and for the current session set with `SetOptions[`*function*`, ` *option* `->` *value*`]`. Usually they are set with each call of a function by appending them to the other arguments: *function*`[`*args*`, ` *option1*, *option2*, … `]`. If in a special case an option is of no physical relevance then this option must not be set.

As evaluations for graphs beyond tree level normally take several minutes screen messages are supplied when running the *FeynArts* functions (except `CreateTopologies`). These messages can be suppressed with the option `ScreenMessages`. In this case the suppressed output is pumped into the *FeynArts* null device `FeynArts.log` in the work directory.

We have tried to restrict the possibilities to run *FeynArts* as little as possible. It should do its job on any computer system where *Mathematica* 2.0 is available. Furthermore the user is free to copy *FeynArts* into the directory of his choice as long as he or she takes care that this directory is on the *Mathematica* search path (see installation).

Since *FeynArts* is a public domain package, we cannot guarantee that it will work correctly under any circumstances. If any problems occur we recommend that the user starts a new *Mathematica*-session and retries the same calculation. If this doesn't help, we are grateful for problem reports sent by mail or e-mail to:

Hagen Eck, Sepp Küblbeck
Physikalisches Institut
Universität Würzburg
Am Hubland
D-8700 Würzburg
e-mail: eck@vax.rz.uni-wuerzburg.dbp.de
kue@vax.rz.uni-wuerzburg.dbp.de

We also welcome any suggestions about how to improve the package in general.

You may make and distribute copies of *FeynArts* provided the code of the distributed copies is identical to the original code of *FeynArts*. Please make sure that the users you give *FeynArts* to, have access to a *FeynArts* manual.

■ 0.2 Acknowledgements

We thank Manfred Böhm and Ansgar Denner for decisive ideas when we developed the algorithms. Our patient β-testers Arnd Kraft and Dirk Michelsen, and especially Rolf Mertig and Georg Weiglein have deserved a big 'well-done'. Thanks to Markus Wendel who made up the name '*FeynArts*'. We greatly appreciate the courtesy of the whole team of Wolfram Research and we thank Stephen Wolfram very much for

[1] Throughout this manual — except in the sample run — it is assumed that *FeynArts* was loaded with the `init.m` file.

the pleasant stay. Expert hints from Tom Wickham-Jones and Joe Kaiping helped us improve the graphics of *FeynArts* and the layout of this manual.

■ 0.3 Installation

The most important point for *FeynArts* to work properly is that *Mathematica* finds the *FeynArts* functions. We suggest to open a directory *feynartsdir* for *FeynArts* and its subdirectories and to set a path from the directory where one wants to work to *feynartsdir* before loading *FeynArts*. In order to do this the following line must be typed in at the beginning of each *FeynArts* session or into the *Mathematica* init file `init.m` (or one has to open one containing this command) in the work directory:

`$Path = Append[$Path,`*feynartsdir*`]`

(where *feynartsdir* has to be a string, i.e. in quotation marks). This assures that no files with batch jobs or results in the work directory will interfere with *FeynArts* files.

For graphics output on screen the appropriate driver which is delivered with *Mathematica* must be loaded. This is done e.g. with `<<Tek.m` for a terminal with *tektronics* facilities and with `<<X11.m` for a X-window terminal. Usually *Mathematica* does this automatically.

A little initialization concerning the particle labels in *FeynArts* graphics might be adequate in order to make full use of the user's graphics resources. To this end the main file `FeynArts.m` must be edited. It contains eight font specifications for the particle labels in the Feynman graphs which are self-explaining. A short description is given in appendix A-1. Users who enter their own models have to adapt the model variables in `FeynArts.m` (see appendix A-2). For the generation of diagrams with infrared singularities we have entered a photon mass `MLA` in the model files. We have set `MLA` to zero in `FeynArts.m` and we have kept this throughout this manual. Users who want to do calculations with finite photon mass have to omit the corresponding line. Finally `FeynArts.m` contains the loading commands for the *FeynArts* subprograms.

Instead of loading *FeynArts* every time one calls *Mathematica* from the work directory one can add the command `<<FeynArts.m` in the *Mathematica* init file `init.m` of the work directory. Then the package is loaded automatically when starting a *Mathematica* session.

1 Creating the Topologies

We start this manual with the description of the function that creates the basic topological objects consisting of vertices and lines. We call these objects topologies and classify them according to the number of external lines ('legs') and the number of loops they contain, which corresponds to the order of perturbation theory in which the computation is set up.

The number of topologies increases sharply with the number of legs and (even more!) with the number of loops. This imposes constraints — depending on the available memory of the computer [2] — on the topologies *FeynArts* can create.

1.1 Using `CreateTopologies`

Since the topologies are classified by two numbers (legs and loops), the syntax of the topology creation function reads:

`CreateTopologies[` l, e `]`	creates topologies with l loops and e external legs

The creation of topologies is done iteratively from topologies with less external legs $e' < e$. (For a detailed description of the algorithm see [2, 3]). Therefore the complete set of start topologies with l loops and the minimum of external legs has to be entered before, following some conventions described below. This was already done for tree graphs (fig.1a), one-loop graphs (fig.1b) and two-loop graphs (fig.1c-e).

Here the topologies with two legs and one loop are generated (self-energies).

```
In[1]:= CreateTopologies[1,2]
Out[1]= TopologyList[Topology[2][Propagator[l1][v3[1], v3[1]],
   Propagator[ex][e[1], v3[2]], Propagator[in][v3[2], v3[1]],
   Propagator[ex][e[2], v3[2]]],
  Topology[2][Propagator[l1][v3[1], v3[2]],
   Propagator[l1][v3[2], v3[1]], Propagator[ex][e[2], v3[2]],
   Propagator[ex][e[1], v3[1]]],
  Topology[2][Propagator[l1][v4[2], v4[2]],
   Propagator[ex][e[1], v4[2]], Propagator[ex][e[2], v4[2]]]]
```

The two-loop topologies with five legs are already rather numerous.

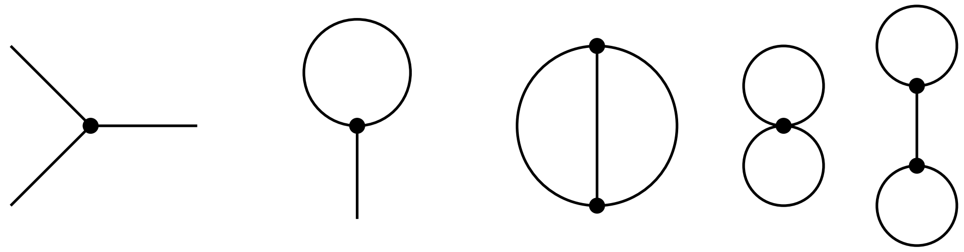
```
In[2]:= Length[ tt = CreateTopologies[ 2, 5 ] ]
Out[2]= 9225
```

`CreateTopologies` always returns an object with head `TopologyList`.

```
In[3]:= Head[ tt ]
Out[3]= TopologyList
```

From these examples it is easy to see the internal representation of a topology: a set of `Propagators` is collected inside an object with head `Topology[`s`]`. The index s in the head of a topology denotes the inverse of the combinatorial factor of this topology. How this factor is constructed is discussed in [3]. The combinatorial factor becomes important when constructing the Feynman amplitudes. As their arguments the `Propagators` carry the two points they connect and an index that distinguishes their types. Possible types of propagators are `ex` (for external propagators), `in` (for internal propagators) and `l1,l2,l3,`... (for loop propagators). External points are denoted by `e`, three-vertices by `v3` and four-vertices by `v4`.

[2]On a DEC5000 we have done up to 6 external legs for one-loop and up to 5 external legs for two-loops.



(a) (b) (c) (d) (e)

Fig.1 start topologies for tree (a), one-loop (b) and two-loop (c,d,e) topologies

For calculations beyond tree level even within a class of graphs with fixed numbers of loops and external legs one is usually interested in subsets which form building blocks for the whole evaluation together with the corresponding countergraphs. To allow this several options can be chosen.

option name	*default value*	
`Tadpoles`	`True`	whether to include tadpole graphs
`SelfEnergies`	`True`	whether to include graphs with two-point functions
`Triangles`	`True`	whether to include graphs with three-point functions
`Boxes`	`True`	whether to include graphs with four- and more-point functions
`WFCorrections`	`True`	whether to include graphs with tadpoles or self-energies on external lines (only works for graphs with irreducible start topologies)
`Theta`	`True`	whether to include descendents from fig.1c
`Eight`	`True`	whether to include descendents from fig.1d
`Bicycle`	`False`	whether to include descendents from fig.1e

Options of `CreateTopologies`

E.g. `CreateTopologies[ 1, 4, Tadpoles->False, Triangles->False]` produces all graphs with four external legs containing no tadpoles and and no three-point functions (there remain self-energy and box graphs). We call an irreducible graph with n legs an n-point function. n-point functions are the constituting parts of one-particle reducible Feynman graphs and multiplicative in momentum space.

The first four options apply to all loop graphs, the fifth only to graphs with irreducible start topologies (otherwise it gives wrong results) and the last three only to two-loop graphs. The latter denote whether descendents of the start topologies in figs.1 c, d and e, respectively are included.

This is the number of all topologies with one loop and four external legs.

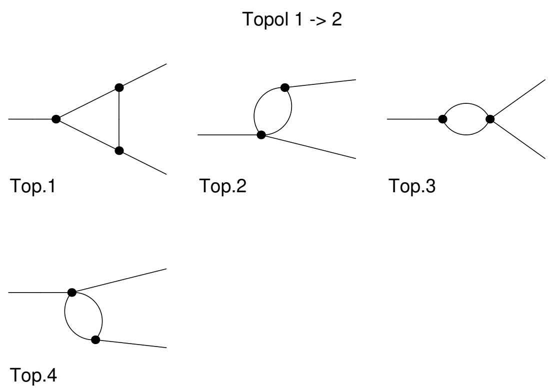
```
In[1]:= Length[ CreateTopologies[ 1, 4] ]
Out[1]= 99
```

Using options reduces the number of topologies and often leaves only the more 'interesting' topologies (only boxes in this case).

```
In[2]:= Length[ CreateTopologies[ 1, 4, Tadpoles->False,
        SelfEnergies->False, Triangles->False ] ]
Out[2]= 12
```

The topologies can be painted with `Paint` as explained in section 3.

```
In[3]:= Paint[ CreateTopologies[1,3,WFCorrections->False], 1,
        2, ScreenMessages->False ] ;
```



We think that for most computations in the electroweak standard model, the model for which *FeynArts* was originally designed, two loops are sufficient. How `CreateTopologies` can be applied to three and more loops is discussed in the following.

■ 1.2 Creating Topologies with three or more Loops

In this section we want to give a glance of the things that happen inside *FeynArts* for users that have already some experience with *Mathematica*.

FeynArts uses a recursive algorithm for the generation of the topologies. Like every recursion, this algorithm needs a condition to prevent infinite program execution. In the case of topology generation this condition is fulfilled when `CreateTopologies` meets a topology which is already defined and therefore can be used to create its successors.

`CreateTopologies` contains 5 pre-defined start topologies (see fig.1). They are entered in the file `CreateTopologies.m`. (For brevity we omit the two-loop topologies.):

```
(* -------------------------------------------------------- *)
BeginPackage["CreateTopologies`","FeynArts`"];
Needs["Utilities`Compare`"];
CreateTopologies::usage = "
...
( usages are ommited here )
```

```
Begin["`Private`"]
                                     (** start-Topologies **)
      tt = Topology[1][ Propagator[ex][e[1],v3[1]],
                        Propagator[ex][e[2],v3[1]],
                        Propagator[ex][e[3],v3[1]] ];
      o1 = Topology[2][ Propagator[l1][ v3[1] , v3[1] ],
                        Propagator[ex][ e[1] , v3[1] ] ];

                         (** recursive construction of topologies **)
      ConstructTopologies[0,3] = TopologyList[tt];
      ConstructTopologies[1,1] = TopologyList[o1];
      ConstructTopologies[loops_, ext_Integer] :=
          TopologyList[ Map[ AddOne[#1,ext]& ,
                        ConstructTopologies[loops,ext-1 ] ] ] ;
      (** main function, checking options **)
      CreateTopologies[l_, n_Integer, options___ ] :=
            CreateTopologies[l, n, options ] =
            Module[.....
(etc.)
(* ------------------------------------------------------- *)
```

This shows some parts of the beginning of this file which are essential. There are two main functions: the function `CreateTopologies` that is called by the user and an internal function `ConstructTopologies` that does all the iterative work.

It is easy to see how higher loop topologies can be added to this:

– Define start topology (or topologies) in the form:
  `Topology[` *combfactor* `][` *prop1*, *prop2*, … `]`
– The propagators (*prop1, prop2, …*) have to be of the form:
  `Propagator[` *type* `][` *point1, point2* `]`
  with *type* out of `l1, l2, l3,` … for a loop propagator or `ex` for an external propagator.
– The points are of the form `v3[`*n*`]` or `v4[`*m*`]` for 3- and 4-vertices respectively.
  Important: the indices of vertices that are not connected to external lines have to be **negative** in the start topologies to avoid multiple production of identical topologies!!
– The start topologies are used as entries to the function `ConstructTopologies`:
  `ConstructTopologies[` *x, y* `] = TopologyList[` *starttop1, …* `]`
  where x stands in *FeynArts* for the number of loops and y for the number of external legs.

When all things are set up correctly then `CreateTopologies` should be able to generate all topologies with more than y external legs out of the set of start topologies.

The x in the definition of the start topologies, though used in *FeynArts* to denote the number of loops in a topology, is nothing but a label to distinguish the start topologies. It can also be used to generate subsets of topologies, e.g. via

```
ConstructTopologies[ Global`a, 0 ]  =  ...
ConstructTopologies[ Global`b, 0 ]  =  ...
```

We invite all users to make their own experiments with the program as long as they re-distribute it only in the form they obtained it.

2 Inserting Fields into Topologies

The next step towards the creation of Feynman graphs and amplitudes is to insert particles into the topologies. Performing this with list manipulation routines on a computer results in a set of insertions in list form for each topology the routine is applied to. Once all insertions of particles which are in accordance with the allowed couplings of the considered model are found a well-known strict set of rules directly leads to Feynman graphs and amplitudes.

2.1 The usage of `InsertFields`

One usually considers a certain process

a) with given external particles
b) in a certain order of perturbation theory
c) within a certain model.

If one takes a list of topologies with the appropriate number of loops and legs produced by `CreateTopologies` and fixes incoming and outgoing particles of the process then a) and b) are satisfied. To fulfill c) the inner lines must be occupied with fields in accordance with the considered model.

If the model exists in form of one or more files `model.`*mod* in the subdirectory `Models`, the insertions for a given list of external particles *extlist* into a given list of topologies *toplist* are performed by `InsertFields[`*top*`,` *ext*`]`.

`InsertFields[` *top*`,` *ext*`, Model->{`*mod1, mod2, …* `} ]`	inserts fields from models *mod1,mod2,...* into *top* with external particles *ext*.

top can be a single topology or a list of topologies. However, its form must closely follow the conventions described in the previous section, so it is safe to produce it with `CreateTopologies`.

`Topology[` *prop, …* `]`	one single topology
`TopologyList[` *top, …* `]`	a list of topologies produced by `CreateTopologies`

possible values of *top* in `InsertFields`

ext must be a rule of the form `{`*field_1*`,… ,` *field_i*`} -> {`*field_i+1*`,… ,` *field_n*`}`, where the first list contains the incoming and the second list the outgoing fields. These fields must have the same names as in the file `model.`*mod* otherwise *FeynArts* will not accept them as valid external particles. The models entered by us comprise all particles of the full standard model (electroweak and QCD). Their field codes which have to be used in `InsertFields` are given in appendix A-1.

■ 2.2 The result: Feynman graphs in list form

All one-loop topologies with three external particles and no wave function corrections are created, so in this case all irreducible 3-point functions are returned. (Output is suppressed with ';'.)

```
In[1]:= tops = CreateTopologies[ 1, 3, WFCorrections->False ];
```

From these topologies we generate the graphs contributing to the electroweak one-loop corrections of the Z-decay into two electrons, in our code `V[2]->F[1]+F[-1]`. We neglect the couplings between the Higgs bosons and light fermions.

The message tells us that the `SM` with the set options is initialized. The binary code shows — in the same order as in the table for the options — which model options (starting with `Generation1`) were set to `True` (1) and which to `False` (0).

Four graphs are found all being inequivalent. The result already contains all necessary information to draw the Feynman graphs.

The overall `Head` of the list of topologies contains the options for the insertion. In addition `Process` contains the entered information which particles of the process are incoming and which are outgoing. According to this the external propagators are divided into incoming and outgoing ones with indices `inc` and `out`, respectively. Obviously the insertions are performed by appending a variable `fi[i]` to every propagator and a list of `Insertions` to every toplogy. Every entry in `Insertions` is a `Graph` which assigns fields to the variables. Different insertions can exhibit different symmetries, resulting in a different combinatorial factor, which is therefore written to each `Graph`.

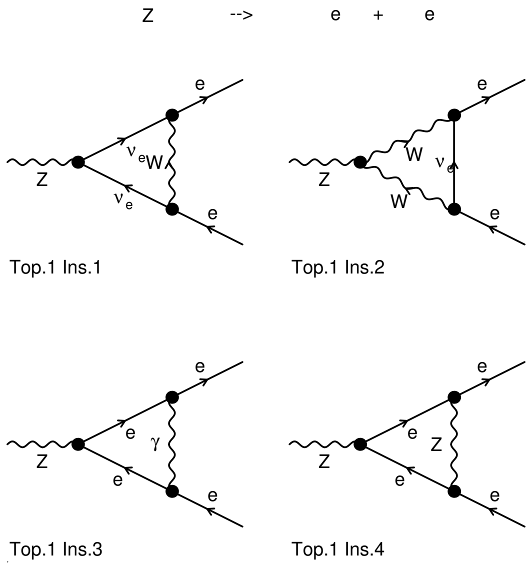
```
In[2]:= ins = InsertFields[ tops , {V[2]}->{F[1],F[-1]} ,
        Model->{SM} , LightFHCoupling -> False ]
 ... initializing model "SM1111000"
 ... LightFermionHiggsCouplings deleted
 |.    |
 |..   |
 |...  |
 |.... |
 |.....|
 ... (done)
1. topology (out of 4) being inserted
4 insertions before Compare
4 insertions after  Compare
2. topology (out of 4) being inserted
0 insertions before Compare
0 insertions after  Compare
3. topology (out of 4) being inserted
0 insertions before Compare
0 insertions after  Compare
4. topology (out of 4) being inserted
0 insertions before Compare
0 insertions after  Compare

Out of 4 tops. 3 have no inss. and are discarded
total number of insertions: 4

Out[2]= TopologyList[Model -> {SM}, Generation1 -> True,
   Generation2 -> True, Generation3 -> True,
   ElectronHCoupling -> True, LightFHCoupling -> False,
   QuarkMixing -> False, UnitaryGauge -> False,
   RemoveEmptyTops -> True, ProcessName -> V2F1aF1,
   Process -> {V[2]} -> {F[1], F[-1]}][Topology[Propagator[inc]
     [e[1], v3[1], fi[1]], Propagator[out][e[2], v3[2], fi[2]],
   Propagator[out][e[3], v3[3], fi[3]],
   Propagator[l1][v3[1], v3[2], fi[4]],
   Propagator[l1][v3[1], v3[3], fi[5]],
   Propagator[l1][v3[2], v3[3], fi[6]],
   Insertions[Graph[1][fi[1] -> V[2], fi[2] -> F[-1],
     fi[3] -> F[1], fi[4] -> F[2], fi[5] -> F[-2],
     fi[6] -> V[-3]], Graph[1][fi[1] -> V[2], fi[2] -> F[-1],
     fi[3] -> F[1], fi[4] -> V[3], fi[5] -> V[-3],
     fi[6] -> F[-2]], Graph[1][fi[1] -> V[2], fi[2] -> F[-1],
     fi[3] -> F[1], fi[4] -> F[1], fi[5] -> F[-1],
     fi[6] -> V[1]], Graph[1][fi[1] -> V[2], fi[2] -> F[-1],
     fi[3] -> F[1], fi[4] -> F[1], fi[5] -> F[-1],
     fi[6] -> V[2]]]]]
```

These are the corresponding Feynman graphs. (The usage of `Paint` is described in section 3.)

```
In[3]:= Paint[ ins, GraphsPerRow->2, RowsPerSheet->2
        ScreenMessages->False]
```




```
Out[3]= {-FeynArtsGraphics-}
```

■ 2.3 The options for `InsertFields`

The most important option is `Model`, which allows one to choose the relevant model within which the process is to happen. `Model` has to be set to a `List`. It can contain one or more of the models contained in the `Models`-directory. Up to now we have entered the electroweak standard model `SM`, quantum chromodynamics for three generations of quarks `QCD` and quantum electrodynamics for quarks and leptons `QED` as well as only for leptons `QE`. In addition a toy model `QC` which contains QCD with only one quark flavor is available. Of course the models should be physically meaningful in the chosen combinations. So `QE` and `QC` cannot be combined with one of the other models. In our conventions the complete standard model including QCD would be chosen setting `Model -> {SM,QCD}`.

option name	*default value*	
`Model`	`{SM}`	chooses the models (in brackets) from which particles are inserted
`Generation1`	`True`	whether to include the 1st fermion generation
`Generation2`	`True`	whether to include the 2nd fermion generation
`Generation3`	`True`	whether to include the 3rd fermion generation
`ElectronHCoupling`	`True`	`False` turns the electron-Higgs couplings off
`LightFHCoupling`	`True`	`False` turns all fermion-Higgs couplings off except to the top quark (proportionality to fermion mass over mass of the W-boson)
`QuarkMixing`	`False`	`False` sets the quark mixing matrix equal to the identity matrix, `True` introduces the matrix elements `KMM`*ij*
`UnitaryGauge`	`False`	whether to use the unitary gauge
`RemoveEmptyTops`	`True`	whether to remove topologies for which no insertions are found
`ScreenMessages`	`True`	whether to print the insertion messages on screen.
`ProcessName`	`Automatic`	The name you want to give to the process. It will mark all created Feynman graphs and amplitudes.

Options of `InsertFields`

Most of the options of `InsertFields` are for imposing constraints on the available models, like omitting certain groups of particles or neglecting couplings.

There are three options `Generation1`, `Generation2` and `Generation3` selecting the desired set of fermion generations. They apply to `SM`, `QCD`, `QED` and `QE` but not to `QC` of course.

With three other options subsets of the couplings of the SM, which are usually negligible in the high energy domain, can be put to zero: There is an option QuarkMixing which sets the quark mixing matrix equal to Identity by default. Set to True it introduces a mixing matrix with entries QMM*ij* between quark *i* and quark *j*. ElectronHCoupling and LightFHCoupling neglect the small couplings ($\sim m_f/m_W$) of the Higgs fields to electrons and to all light fermions, respectively. (Please note that LightFHCoupling->False overrules ElectronHCoupling->True!) Of course this leads in general to a smaller amount of graphs.

The same is valid if one sets UnitaryGauge to True in the SM, the only model where this option applies. Then all unphysical fields (unphysical Higgs and Faddeev-Popov ghosts) will not appear and the Feynman rules are changed in the appropriate way. If set to False the Feynman rules are retained in 't Hooft gauge. In this case one can set an option called FeynmanGauge when applying CreateFeynAmp to create the Feynman amplitudes (see section 4.1) in 't Hooft-Feynman gauge.

The other options serve to design the output. At the end of the insertion procedure RemoveEmptyTops discards by default all topologies for which no Graphs were found. If the user has produced and maybe Painted a list of topologies (see section 3.1) and wants to keep track of the corresponding numbers in the TopologyList he can set RemoveEmptyTops to False.

ProcessName should be a short name for the process. (It appears — with appropriate indices — with the Feynman graphs and amplitudes when applying Paint and CreateFeynAmp.) If no name is specified InsertFields automatically extracts a name from the particle codes of the external particles (see appendix A-1) by omitting the square brackets and inserting a "a" (for antiparticle) for each field with negative index. So the decay of W-boson, in our code:

```
V[3] -> F[1] + F[-2]
```

would yield the name: V3F1aF2.

Before the insertion can start the models called in the option Model must be initialized with their actual options. *FeynArts* initializes the models automatically whenever it meets a model that hasn't been initialized yet. If the option ScreenMessages is set to True *FeynArts* gives a corresponding message on the screen (e.g. ...initializing model "SM1110100" and a sequence of rows of dots indicating how much of the initialization of the routine is completed). The binary code shows the values of the set model options in the same order as in the table for the options. Although not all options are meaningfull for all models they are also included in the code for book-keeping reasons. Furthermore (if ScreenMessages is set to True) the program tells during the run on which topology it works and how many insertions are found for it before and after the Compare routine. This routine collects identical insertions, keeps only one representative and adjusts the combinatorial factor. Finally the total amount of found Feynman graphs is displayed.

3 Drawing Feynman Graphs

One of the advantages of Feynman's approach to field theory is that single contributions to cross sections and decay rates can be visualized in a very intuitive way. A program generating Feynman graphs would not be complete without some sort of graphics routine to show the fruits of its work.

Graphs can only be drawn by a machine if the positions of the vertex points on the sheet are determined before. Their coordinates can be entered by hand or they can be found with the help of a routine. The problem to make up an algorithm which automatically finds a nice representation especially for graphs with more than one loop is anything but trivial. Unfortunately our private location finder routine is not yet in a state to be distributed with *FeynArts*. However we have written the routine `FeynEdit` which allows to enter graph points in a comfortable way. This routine is described in section 3.2.

3.1 The *FeynArts* Graphics Routine

The *FeynArts* graphics routine is `Paint[` *expr* `]`. Its syntax is:

`Paint[` *expr* `]`	draw Feynman diagrams
`Paint[` *expr, n, m* `]`	draw topologies

It expects input generated by *FeynArts* i.e. *expr* has to be of the form:

`Topology[` *prop, …* `]`	one single topology
`TopologyList[` *top, …* `]`	a list of topologies produced by `CreateTopologies`
`Topology[` *prop, …* `,  Insertions[` *ins, …* `] ]`	Feynman graphs for one …
`TopologyList[` *descr* `][` *insertedtop, …* `]`	… or more topologies as produced by `InsertFields`

Possible arguments of `Paint`.

Since `Paint` has to know how many incoming and outgoing lines the drawn topology should have, one has to specify those when drawing 'bare' topologies, i.e. for topologies produced by `CreateTopologies` the syntax is `Paint[` *expr, n, m* `]`, where *n* and *m* are the numbers of incoming and outgoing lines, respectively.

For drawing Feynman graphs produced by `InsertFields` no further specification is needed since the inserted topologies contain all information about the process.

option name	*default value*	
`GraphsPerRow`	`3`	how many graphs per row
`RowsPerSheet`	`3`	how many rows of graphs on one sheet
`ScreenOutput`	`True`	whether to show the graphs on screen
`PSFileOutput`	`True`	whether to write the graphics output to a postscript file
`ScreenMessages`	`True`	whether to print the messages which graph is being painted.
`AutoEdit`	`False`	set to `True` the `FeynEdit` modus is automatically entered
`FullNumbering`	`False`	additional graph numbering
`GrayBackground`	`False`	whether to use grey background
`ShowGrid`	`False`	whether to show a grid
`ShowPointNames`	`False`	whether to show the vertex names
`PropagatorThickness`	`Automatic`	how thick to draw the propagator
`VertexSize`	`Automatic`	which size to use for the vertices

Options of `Paint`

The graphs are drawn in `RowsPerSheet` rows of `GraphsPerRow` graphs. Each graph is captioned with two numbers for the current topology and insertion, respectively. The option `FullNumbering` allows to add a third number which tells the position of the current graph in the total of all graphs. When applied to a single inserted topology (without head `TopologyList`), `Paint` will omit the header for each printed sheet and will label the graphs only as `Ins.`*m*.

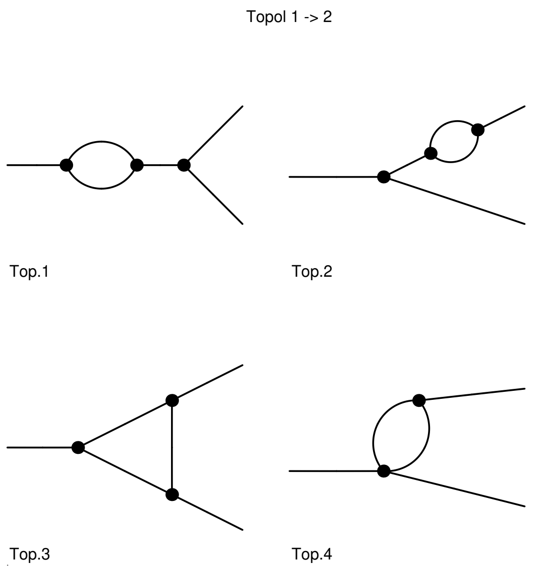
If option `PSFileOutput` is set to `True`, a pre-postscript file with the name *ProcessName*`.gra` is created in the current work directory. The name of this file can be changed using the option `ProcessName` in `InsertFields`. For instructions how to print the graphics output (directly or via graphics file) we refer to the *Mathematica User's Guide* for the various machines.

The output of the graphs to screen can be suppressed by setting `ScreenOutput` to `False`. Again with `ScreenMessages` the messages on screen can be avoided. If a topology comes up for which no graph points are found these can most easily be entered with `FeynEdit` as described in the next section. This edit mode is automatically entered when the option `AutoEdit -> True` is added. Otherwise the corresponding Feynman graphs in the output are replaced by a few question marks.

A few other less important options are implemented and listed in the table for the options of `Paint`.

Here is an example for some one-loop topologies.

```
In[1]:= Paint[ Take[ CreateTopologies[1,3],{7,10} ], 1, 2,
        RowsPerSheet->2, GraphsPerRow->2]
```



`ins` contains one of the two-loop contributions to a quark selfenergy

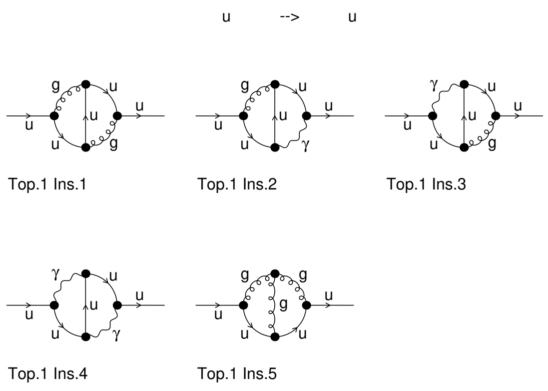
```
In[2]:= ins = InsertFields[ Take[ CreateTopologies[2,2],{3} ],
        {F[3]}->{F[3]}, Model->{QCD,QED} ];

 ... initializing model "QCD1111100"
 |.    |
 |..   |
 |...  |
 |.... |
 |.....|
 ... (done)
 ... initializing model "QED1111100"
 |.    |
 |..   |
 |...  |
 |.... |
 |.....|
 ... (done)
1. topology (out of 1) being inserted
10 insertions before Compare
5 insertions after  Compare

total number of insertions: 5
```

and here are the corresponding Feynman graphs.

```
In[3]:= Paint[ ins , RowsPerSheet->3, GraphsPerRow->3 ]
```



3.2 Entering Graph Points

As mentioned above the drawing program has to know from somewhere, at which position every point should be located (at least this is better for the result of the drawing). The `Paint` routine reads this information from the file *GraphInfo.m* in the subdirectory *GraphInfo*.

The drawing informations are stored for the following sets of topologies:

- tree level: $1 \to 2, 2 \to 2, 2 \to 3$
- one-loop level: $1 \to 1, 1 \to 2, 2 \to 2$
- two-loop level: $1 \to 1,$

So for most of the other topologies to be drawn, the point information must be added. (Among other things the authors of *FeynArts* want to give its users the opportunity to be a little bit creative.)

`FeynEdit` is a simple interactive procedure that provides the user with editing functions for shaping topologies. The syntax of `FeynEdit` ressembles the syntax of Paint:

`FeynEdit[` *expr* `]`	edit Feynman diagram *expr*
`FeynEdit[` *expr, n, m* `]`	edit topology *expr*

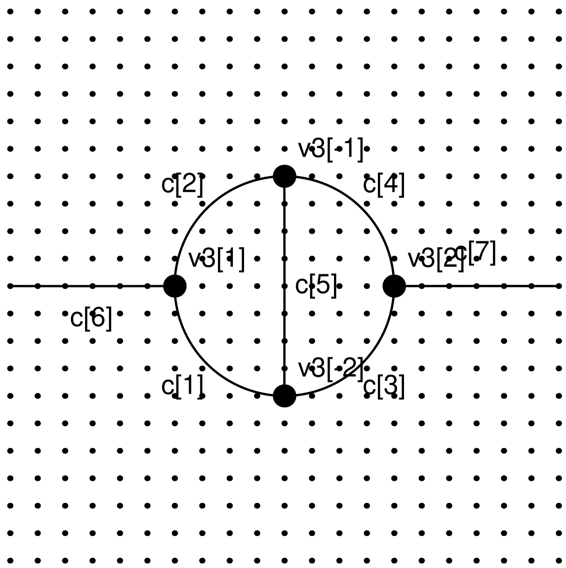
The major difference between the arguments of `Paint` and `FeynEdit` is that only single topologies (inserted or not) fit as arguments of `FeynEdit`. If the user wishes to edit a complete list of topologies, he has to map `FeynEdit` on this list, i.e. use it in the form `FeynEdit/@` *expr* when *expr* is a list of inserted topologies or `FeynEdit[#,`*n,m*`]& /@` *expr* when *expr* is a list of bare topologies (*n,m* numbers of incoming and outgoing particles respectively).

A much more comfortable way using `FeynEdit` is to call `Paint` with the option `AutoEdit -> True`. As soon as an unknown topology occurs, `Paint` stops the program execution and invokes `FeynEdit`. (Of course this only makes sense when using `Paint` interactively.)

When `FeynEdit` is started, it draws the topology placing the vertices on a grid of dimensions 20×20. The grid is also shown in the picture as well as the internal names of the vertices and centerpoints of the propagators. For a topology that is already known to *FeynArts* (i.e. that has already been treated with `FeynEdit` before) this would look as follows:

The grid for fixing the vertexpoints is shown as well as the names of the vertex points and the numbers of the centers of the lines.

```
In[1]:= FeynEdit[ CreateTopologies[ 2, 2 ][[3]], 1, 1 ]
```



For an unknown topology, `FeynEdit` puts the vertices on random sites on the grid. The location and the shape of the propagator lines are specified by the coordinates of the end points and the location of its center (midpoint of the line). This center can be fixed by its coordinates or by its displacement from the midpoint of the straight line connecting the two vertices. All coordinates must be entered in units of the grid with size 20×20. Like in the graphics the vertices are denoted by `v[`*n*`]` and the center points by `c[`*n*`]` with integer numbers *n*.

The user is asked to do something with the current topology. This is symbolized by the prompt "`FeynEdit>`". Any action consists of at least one small letter followed by the RETURN-Key. Some of them (like vertex-shift) prompt for additional information. The possible actions are:

- **c** (center): choose coordinates of center of line. `FeynEdit` first prompts for the name of the center you wish to fix, then for the values of x- and y-coordinates.
- **d** (displacement): determine curvature of the line by choosing the displacement. (This is the distance of the midpoint of the arc to the midpoint of the straight line connecting the two vertices.) After specifying the center of the line, `FeynEdit` prompts for the value of the displacement.

  Remark: FeynEdit computes the location of the center point from a given displacement. It is not possible to do this, when the propagator line ends at the same point where it started (tadpoles) and '**d**' will not work in this case.
- **f** (file): save the complete graphics information in a file. On some computer systems this doesn't work. In this case you have to save the graphics information function 'by hand'. To do this type

Save[*graphinfofile*, GraphPoints`GraphPoints ] *after* leaving FeynEdit[3]. The file *graphinfofile* is in the directory GraphInfo and is named GraphInfo.m. So under UNIX the command would read: Save[ "GraphInfo/GraphInfo.m", GraphPoints`GraphPoints].

- **h** (help): print out a list of commands.
- **p** (print): print the topology as a list of propagators together with coordinates of vertices and center points.
- **r** (roll the dice): the internal points are distributed randomly on the grid. This can be used in the beginning of a FeynEdit-session, when the first randomization did cause a completely disordered distribution of points.
- **s** (show): display the topology.
- **u** (up): shift the complete graph in y-direction. FeynEdit prompts for the number of units you want to shift. Of course negative inputs are allowed.
- **v** (vertex-shift): this is probably the most often used function. You are prompted for the name of the vertex to shift (again make sure to type this correct) and for the value of shift in x- and y-direction.
- **x** (exit): leave FeynEdit. Do not forget to save your graphics information otherwise your work will be lost when you leave *Mathematica* !!!

All input numbers can be negative (e.g. for shifting a vertex in negative x-direction). Especially for the displacement of the center points often 'negative' displacements are needed. For self-energy loops the displacements of the two loop-propagators usually are of the same magnitude but of different signs.

Furthermore it should be mentioned, that FeynEdit tries to avoid 'ugly' random distributions: no two points are drawn on the same location and all internal points are three units off the edge of the grid. FeynEdit recognizes selfenergies and tadpoles and uses an automatic displacement and center choice routine.

[3]If one stores the points by hand (with this method) under a different name, one can of course have different versions of the same topology available.

4 Creating the Analytic Expressions

■ 4.1 Using `CreateFeynAmp`

From the Feynman graphs in list form, *FeynArts* creates the corresponding analytical expressions with the function `CreateFeynAmp`.

`CreateFeynAmp[` *set* `]`	produces the analytic expressions to a *set* of graphs generated with `InsertFields`

set is an expression produced by `InsertFields` with the form

`TopologyList[` *rule1, rule2, …* `][` *topology1, topology2, …* `]`

Arguments of `CreateFeynAmp`.

The rules in the head of the topologylist (*rule1, rule2,…*) contain the complete information about the model in which the graphs were generated as well as the information about the external particles and the process name. So no further arguments and specifications are needed. Therefore `CreateFeynAmp` cannot be applied to single topologies which do not carry the required head `TopologyList[...]`.

The output is in a form that is easy to work on with *FeynCalc*. If the Feynman rules of the used model are entered like for the standard model (see appendix B) it consists of three parts: The non-commuting part containing Dirac and Gell-Mann matrices, the denominator of the scalar integral and the 'rest' consisting of the coupling constants, symmetry factors, and Lorentz tensors.

option name	*default value*	
`ScreenMessages`	`True`	whether to print the messages about the translation on screen.
`Truncated`	`False`	whether to produce the amplitudes without wave functions.
`FeynmanGauge`	`True`	choice of the parameter `GaugeXi`
`$DDimension`	`False`	choice of prefactor

Options of `CreateFeynAmp`.

The `ScreenMessages` option allows to suppress the coded messages that are displayed during program execution: A row of *j* points is shown if the Feynman graphs of the *j*th topology are translated. A bar '`|`' marks the place where the row of points for the last topology will end. Behind it appears a number indicating how many insertions the topology has which is currently worked on. It is a measure for the time the translation takes.

If the graphs were produced by `InsertFields` in the (non-unitary) 't Hooft gauge one can set the gauge parameters $\xi(i),\ i = \gamma, Z, W$ equal to 1 with the option `FeynmanGauge`. Of course this means that one chooses the 't Hooft-Feynman gauge.

The option `$DDimensions` sets the value of the prefactor of every Feynman graph. In $\$D$ dimensions it is $(\frac{\mu^{4-\$D}}{(2\pi)^{\$D}})^{(loops)}$ else it is $(\frac{1}{2\pi})^{4\cdot(loops)}$ (in 4 dimensions).

Let us create some simple topologies, those contributing to the one-loop self energy.

```
In[1]:= Length[ tt = CreateTopologies[1,2] ]

Out[1]= 3
```

This are the Feynman graphs for the Gluon selfenergy in quantum chromo dynamics.

```
In[2]:= ins = InsertFields[ tt, {G[1]}->{G[1]}, Model->{QCD} ];

 ... initializing model "QCD1111100"
 |.    |
 |..   |
 |...  |
 |.... |
 |.....|
 ... (done)
1. topology (out of 3) being inserted
15 insertions before Compare
8 insertions after  Compare
2. topology (out of 3) being inserted
15 insertions before Compare
8 insertions after  Compare
3. topology (out of 3) being inserted
1 insertions before Compare
1 insertions after  Compare

total number of insertions: 17
```

`InsertFields` keeps the structure of seperate topologies.

```
In[3]:= Length[ ins ]

Out[3]= 3
```

In contrast `amp` is just al list of analytical expressions.

```
In[4]:= Length[ amp = CreateFeynAmp[ ins, Truncated->True ] ]

.  |(8)
.. |(8)
...|(1)

Out[4]= 17
```

here is one of them. The head `FeynAmp` collects the name of the graph, the momenta on the loops and the wanted analytical expression.

```
In[5]:= amp[[10]]

                                           2
Out[5]= FeynAmp[G1G1, T2, I2, N10][-(GS  Integral[q1]

      tr[la[gi4] . la[gi2]]

      tr[(MT + gs[-p1 + q1]) . ga[li4] . (MT + gs[q1]) .

                          4      2            2      2     2
        ga[li2]]) / (64 Pi  (-MT  + (p1 - q1) ) (-MT  + q1 ))]
```

If a graph has a propagator with mass and momentum 0, `CreateFeynAmp` sets it to zero (which is the correct result in consistent field theories). So tadpoles with mass 0 in their tails yield 0.

```
In[6]:= amp[[2]]

Out[6]= FeynAmp[G1G1, T1, I2, N2][0]
```

The momenta of the i incoming particles are called `p1`,...,pi, the momenta of the j outgoing particles are called `k1`,... ,`k`j. For conventions and physical content of the used symbols see appendix C. For each loop a momentum `q`n is introduced ($n = 1, \ldots, loops$).

The output of `Out[5]` in the example above is not the internal representation of a Feynman amplitude. What *Mathematica* printed on the screen is just a user friendly decryption of a somewhat lengthy but "*FeynCalc*-friendly" internal expression (see next section).

■ 4.2 Interpreting Results

To show how *FeynArts* habdles things internally, we create the amplitudes for the decay of a Z-boson into to neutrinos in the electroweak SM.

This creates the vertex topologies without wave function corrections. (Output is suppressed with ';'.)

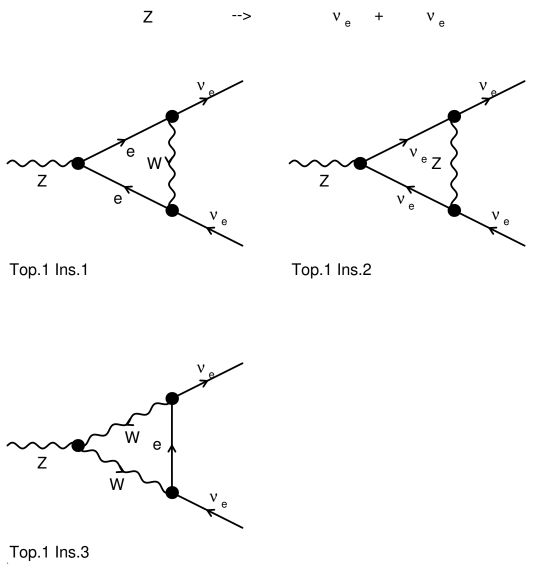
```
In[1]:= tt = CreateTopologies[1,3,WFCorrections->False];
```

`ins` contains all irreducible vertex corrections to the decay $Z \to \nu\overline{\nu}$ in the `SM` with neglected couplings between Higgs particles and light fermions. The run time messages are suppressed this time.

```
In[2]:= ins = InsertFields[ tt, {V[2]}->{F[2],F[-2]},
        Model->{SM}, LightFHCoupling->False,
        ScreenMessages->False ]
Out[2]= TopologyList[Model -> {SM}, Generation1 -> True,
   Generation2 -> True, Generation3 -> True,
   ElectronHCoupling -> True, LightFHCoupling -> False,
   QuarkMixing -> False, UnitaryGauge -> False,
   RemoveEmptyTops -> True, ProcessName -> V2F2aF2,
   Process -> {V[2]} -> {F[2], F[-2]}][Topology[Propagator[inc]
     [e[1], v3[1], fi[1]], Propagator[out][e[2], v3[2], fi[2]],
   Propagator[out][e[3], v3[3], fi[3]],
   Propagator[l1][v3[1], v3[2], fi[4]],
   Propagator[l1][v3[1], v3[3], fi[5]],
   Propagator[l1][v3[2], v3[3], fi[6]],
   Insertions[Graph[1][fi[1] -> V[2], fi[2] -> F[-2],
     fi[3] -> F[2], fi[4] -> F[1], fi[5] -> F[-1],
     fi[6] -> V[3]], Graph[1][fi[1] -> V[2], fi[2] -> F[-2],
     fi[3] -> F[2], fi[4] -> F[2], fi[5] -> F[-2],
     fi[6] -> V[2]], Graph[1][fi[1] -> V[2], fi[2] -> F[-2],
     fi[3] -> F[2], fi[4] -> V[-3], fi[5] -> V[3],
     fi[6] -> F[-1]]]]]
```

These are the graphs.

```
In[3]:= Paint[ ins, GraphsPerRow->2, RowsPerSheet->2,
        ScreenMessages->False ]
```



This is the output form of a Feynman amplitude to a certain process. In addition to the model option which were used to create the amplitude its head contains the `Process` containing the involved particles, their momenta and masses.

Underneath we give the internal representation (*FeynCalc* form) of the second Feynman amplitude. On the left side we give the usual mathematical notation which is obtained using the translation rules given in appendix A-3 and substituting in addition `li2`, `li3` and `li4` by μ, ν and ρ, respectively.

```
In[4]:= amp = CreateFeynAmp[ ins ]
.|(3)
Out[4]= FeynAmpList[Model -> {SM}, Generation1 -> True,
   Generation2 -> True, Generation3 -> True,
   ElectronHCoupling -> True, LightFHCoupling -> False,
   QuarkMixing -> False, UnitaryGauge -> False,
   RemoveEmptyTops -> True, ProcessName -> V2F2aF2,
   Process ->
    {{V[2], p1, MZ}} -> {{F[2], k1, 0}, {F[-2], k2, 0}}][FeynA\

                                    -I    3
     mp[V2F2aF2, T1, I1, N1][(-- EL  ep[p1, li2] g[li4, li3]
                                    32

                                                 1 - ga[5]
      Integral[q1] u[k1, 0] . ga[li3] . (---------) .
                                                     2

                                              2   1 - ga[5]
       (ME + gs[k1 + q1]) . ga[li2] . (SW  - ---------) .
                                                       4

                                                  1 - ga[5]
       (ME + gs[k1 - p1 + q1]) . ga[li4] . (---------) .
                                                      2

       v[k2, 0]) /

            4        2      2       2            2
    (CW Pi  (-MW  + q1 ) (-ME  + (k1 + q1) )

         2                  2    3
      (-ME  + (k1 - p1 + q1) ) SW )],

                                     -I    3
  FeynAmp[V2F2aF2, T1, I2, N2][(-- EL  ep[p1, li2] g[li3, li4]
                                     16

                                                 1 - ga[5]
      Integral[q1] u[k1, 0] . ga[li3] . (---------) .
                                                     4

                                    1 - ga[5]
       gs[k1 + q1] . ga[li2] . (---------) . gs[k1 - p1 + q1] .
                                        4

                          1 - ga[5]
       ga[li4] . (---------) . v[k2, 0]) /
                              4

        3    4           2                2      2     2    3
    (CW  Pi  (k1 + q1)  (k1 - p1 + q1)  (-MZ  + q1 ) SW )],

                                     I          3
  FeynAmp[V2F2aF2, T1, I3, N3][(-- CW EL  ep[p1, li2]
                                    32

      g[li3, li4] g[li6, li5] Integral[q1]

                                    1 - ga[5]
      u[k1, 0] . ga[li4] . (---------) . (ME + gs[-q1]) .
                                        2

                          1 - ga[5]
       ga[li6] . (---------) . v[k2, 0]
                              2

      (g[li5, li3] (-2 k1 + p1 - 2 q1)[li2] +
        g[li2, li5] (k1 - 2 p1 + q1)[li3] +
        g[li3, li2] (k1 + p1 + q1)[li5])) /

       4      2      2       2            2
    (Pi  (-ME  + q1 ) (-MW  + (k1 + q1) )

         2                  2    3
      (-MW  + (k1 - p1 + q1) ) SW )]]
```

$$\frac{ie^3}{32\pi^4 c_w s_w^3}\epsilon^\mu(p_1)g^{\nu\rho}\int d^4q_1 \frac{\overline{u}(k_1)\gamma_\nu\frac{\omega_-}{2}}{(q_1^2-m_Z^2)}\times$$

$$\times\frac{(\not{k}_1+\not{q}_1)\gamma_\mu\frac{\omega_-}{2}(\not{k}_1-\not{p}_1+\not{q}_1)\gamma_\rho\frac{\omega_-}{2}v(k_2)}{[(q_1+k_1)^2-m_e^2][(q_1+k_1-p_1)^2-m_e^2]}$$

```
In[5]:= InputForm[ amp[[2]] ]

Out[5]//InputForm=
  FeynAmp[GraphName[V2F2aF2, T1, I2, N2], q1,
   (-I/16*EL^3*LeptonSpinor[k1, 0] . DiracMatrix[li3] .
       (ChiralityProjector[-1]/2) . DiracSlash[k1 + q1] .
       DiracMatrix[li2] . (ChiralityProjector[-1]/2) .
       DiracSlash[k1 - p1 + q1] . DiracMatrix[li4] .
       (ChiralityProjector[-1]/2) . LeptonSpinor[-k2, 0]*
      FeynAmpDenominator[PropagatorDenominator[q1, MZ],
       PropagatorDenominator[k1 + q1, 0],
       PropagatorDenominator[k1 - p1 + q1, 0]]*
      MetricTensor[li3, li4]*PolarizationVector[p1, li2])/
    (CW^3*Pi^4*SW^3)]
```

The list of amplitudes contains the variable `Process` which specifies the external particles and their associated variables. It contains the momenta and masses of the external particles. Overall momentum conservation is obeyed in the form:

$\sum$ *incoming momenta* $=$ $\sum$ *outgoing momenta.*

This additional information is necessary for the further evaluation of the matrix elements, e.g. by *FeynCalc*. *FeynCalc* is a package that is already adapted to the internal representation of *FeynArts*. We give the complete list of translation rules for the built-in objects in appendix A-3.

5 A Sample Run

In this section we show the run for creating all triangle and box one-loop graphs with the corresponding amplitudes for $e^+e^- \to ZH$ in the electroweak standard model without quark mixing. It was this run we used to calculate the radiative corrections to the cross section for ZH-production [4]. As the result is very lengthy, we suppress the output with the *Mathematica* standard supplying ';' at the end of most commands.

Loading the package.

```
In[1]:= <<FeynArts.m

FeynArts 1.0 for Mathematica 2.0 (October 1991)
by Hagen Eck and Sepp Kueblbeck
[additional packages being loaded]
```

We create all triangle and box one-loop graphs with four external legs.

```
In[2]:= topols = CreateTopologies[ 1, 4,
             Tadpoles->False,
             SelfEnergies->False ];
```

`topols` contains 36 box and triangle topologies

```
In[3]:= Length[ topols ]

Out[3]= 36
```

The external particles are specified and the desired model is automatically initialized. After this, the insertion starts. It can be seen, that numerous insertions were found for the first topology and none for the last.

```
In[4]:= ZHgraphs = InsertFields[ topols,
           {F[1],F[-1]} -> {V[2],S[1]},
            ElectronHCoupling -> False ];

 ... initializing model "SM1110100"
 ... ElectronHiggsCouplings deleted
 |.    |
 |..   |
 |...  |
 |.... |
 |.....|
 ... (done)
1. topology (out of 36) being inserted
78 insertions before Compare
78 insertions after  Compare
2. topology (out of 36) being inserted
10 insertions before Compare
5 insertions after  Compare
.
...   numbers 3 to 34 are omitted
.
36. topology (out of 36) being inserted
0 insertions before Compare
0 insertions after  Compare

Out of 36 tops. 26 have no inss. and are discarded
total number of insertions: 108
```

The amplitudes are created and assigned to `ampl`. The first topology with 78 insertions takes longest to be translated.

```
In[5]:= ampl = CreateFeynAmp[ ZHgraphs ];

.         |(78)
..        |(5)
...       |(5)
....      |(6)
.....     |(2)
......    |(2)
.......   |(2)
........  |(2)
......... |(4)
..........|(2)
```

This saves the result of the computation in the file `eeZH.amp`.

```
In[6]:= ampl >> eeZH.amp
```

We use `Paint` to draw the graphs. Twelve graphics objects are created, each being a sheet with an array of Feynman graphs.

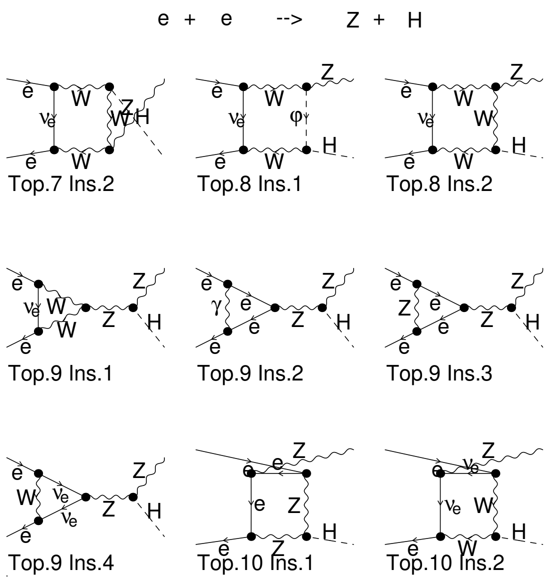
```
In[7]:= Paint[ ZHgraphs, ScreenOutput -> False ]

overwrite file "F1aF1V2S1.gra"
topology: 1  insertion: 1  nr: 1
topology: 1  insertion: 2  nr: 2
topology: 1  insertion: 3  nr: 3
topology: 1  insertion: 4  nr: 4
.
...   numbers 5 to 105 are omitted
.
topology: 9  insertion: 4  nr: 106
topology: 10  insertion: 1  nr: 107
topology: 10  insertion: 2  nr: 108

Out[7]= {-FeynArtsGraphics-, -FeynArtsGraphics-,
  -FeynArtsGraphics-, -FeynArtsGraphics-, -FeynArtsGraphics-,
  -FeynArtsGraphics-, -FeynArtsGraphics-, -FeynArtsGraphics-,
  -FeynArtsGraphics-, -FeynArtsGraphics-, -FeynArtsGraphics-,
  -FeynArtsGraphics-}
```

One can draw single elements of the list of `FeynArtsGraphics` objects.

```
In[8]:= Show[ %[[12]] ]
```



The whole run takes less than ten minutes on a DEC 5000 or a NeXTcube. The saved graphs can be printed on a laser printer and the amplitudes can be evaluated with *FeynCalc*.

A-1 Particle Codes and Representations

vector bosons:					
`V[1]`	photon	γ	`V[2]`	Z-boson	Z
`V[3]`	W^--boson	W	`V[-3]`	W^+-boson	W
`G[1]`	gluon	g			
fermions:					
`F[1]`	electron (e)	e	`F[-1]`	positron	e
`F[2]`	e-neutrino	ν_e	`F[-2]`	e-antineutrino	ν_e
`F[3]`	u-quark	u	`F[-3]`	anti-u-quark	u
`F[4]`	d-quark	d	`F[-4]`	anti-d-quark	d
`F[5]`	muon (μ)	μ	`F[-5]`	anti-μ	μ
`F[6]`	μ-neutrino	ν_μ	`F[-6]`	μ-antineutrino	ν_μ
`F[7]`	c-quark	c	`F[-7]`	anti-c-quark	c
`F[8]`	s-quark	s	`F[-8]`	anti-s-quark	s
`F[9]`	tauon (τ)	τ	`F[-9]`	anti-τ	τ
`F[10]`	τ-neutrino	ν_τ	`F[-10]`	τ-antineutrino	ν_τ
`F[11]`	t-quark	t	`F[-11]`	anti-t-quark	t
`F[12]`	b-quark	b	`F[-12]`	anti-b-quark	b
`F[I]`	'toy'-quark	q	`F[-I]`	'toy'-antiquark	q
Faddeev Popov ghosts:					
`U[1]`	photon-ghost	u_γ	`U[-1]`	photon-antighost	u_γ
`U[2]`	Z-ghost	u_Z	`U[-2]`	Z-antighost	u_Z
`U[3]`	W^--ghost	u_-	`U[-3]`	W^--antighost	u_-
`U[4]`	W^+-ghost	u_+	`U[-4]`	W^+-antighost	u_+
`U[5]`	gluon-ghost	u_g	`U[-5]`	gluon-antighost	u_g
scalar bosons:					
`S[1]`	neutral physical Higgs				H
`S[2]`	neutral unphysical Higgs				χ
`S[3]`	negatively charged unphysical Higgs				φ
`S[-3]`	positively charged unphysical Higgs				φ

Table A.1: Particle codes and graphical representation. The direction of the particle propagation is from left to right.

Table A.1 gives the codes and graphical representations for all particles defined in the model files:

model	vector bosons	fermions	ghost fields	scalar fields
QC	G	q	u_G	–
QCD	G	u, d, c, s, t, b	u_G	–
QE	γ	e, μ, τ	–	–
QED	γ	$e, u, d, \mu, c, s, \tau, t, b$	–	–
SM	γ, Z, W	$e, \nu_e, u, d, \mu, \nu_\mu, c, s, \tau, \nu_\tau, t, b$	u_γ, u_Z, u_+, u_-	H, χ, ϕ

The first of the three (sub)columns in table A.1 gives the particle code as it is used in the internal *FeynArts* representation and as it has to be entered when specifying the external particles in the function `InsertFields`. We have chosen to give members of a particle-antiparticle pair opposite signs in the arguments. The user who enters his own model does not have to follow this convention. What he should stick to, however, are the heads of the particle code. So particles with heads `F` (fermions) have non-commuting Feynman rules, they and particles with heads `U` (Faddeev-Popov ghosts) get "–" signs for closed loops and permutations of external legs. Fields with heads `V` and `G` (vector bosons) receive Lorentz or Lorentz and gluon indices, respectively. The remaining (scalar) particles carry a head `S`.

The center column contains the standard model names of the fields. The right column gives their graphical representation within *FeynArts*, where the specified particle propagates from left to right. The propagators of the particles will appear in this way if the output device has the `Symbol`-font.

As many systems do not support symbol fonts on screen we have set the `$ScreenTextFont` and `$ScreenSymbolFont` to a 10 point `Courier` font by default. If you want to change the font defaults you have to edit the file `FeynArts.m`. In this file eight font variables can be specified right after the release notes. Those beginning with `$Small` are for subscripts and should be of a size of about 7 or 8 points whereas the normal size can be 9 or 10 points. `..TextFont` determines which font is used for latin letters, whereas `...SymbolFont` does the same for greek letters. If the available symbol font has different assignments for greek letters the values of the function `LineSpec` has to be changed accordingly in the model files.[4] `Screen` and `Printer` specify the output form for which the font is for.

The different forms of lines and the subscripted variables will appear in any case, however, if your output device does not have a font with Greek letters it will return mnemonic latin letters. The translation rules for this are also given (and can be changed) in `FeynArts.m`. We suggest the following replacements and we have entered them in `FeynArts.m`:

Particle	is denoted by
γ	A
ν	n
μ	m
τ	l
χ	x
φ	fi

[4] At this point we wish to remind you that if you change something in `FeynArts.m` or the model files or any other files, be sure that a redistributed version of *FeynArts* consists only of the original files!

A-2 Entering a New Model

FeynArts was originally designed to be a tool for Feynman graph generation in the standard model. The implemented algorithms are general enough to be able to create Feynman graphs and amplitudes in any renormalizable quantum field theory. However, the package is not especially designed for the safe entering of a new model so that this is a delicate business for users who are not familiar with the *FeynArts* code. So the model options described in section 2 which apply to the models entered by us will in general not work for other models. Apart from that if the the rules given in appendix A-1 and this section for entering a new model are followed, it is likely that *FeynArts* will work for the new model.[5]

The Feynman rules are the input which is most susceptible to errors. We have entered them for the full standard model `{QCD,SM}`. Covering about 1000 lines of code this part of *FeynArts* is the largest. It was extensively checked by several people and by producing output which could be easily checked or compared to well-known results.

Users who want to use a different model have to enter Feynman rules and graphics specifications into a file in the subdirectory `Models` called '`model.`*mod*'. The extension *mod* is an arbitrary name you can choose for the model. We used extensions with three or less letters in order not to prevent the use of *FeynArts* under MS-DOS. Furthermore *mod* has to be comprised in the list of all available models `Models` and in `SingleModels` if it is not to be combined with other models in `FeynArts.m`. When in addition to that the Feynman rules are entered, Feynman graphs and amplitudes can be created within this model by setting the option `Model->{`*mod*`}` in `InsertFields` (see section 2.2) and applying `Paint` and `CreateFeynAmp` to the result.

We recommend that in entering a new model one follows our conventions as closely as possible. Some rules *must* be obeyed in order to produce sensible output. To illustrate this we want to explain the model file for a simple example: QED for the first lepton generation. We refer users who want to see more sophisticated models with more kinds of fields to the `Models` subdirectory of *FeynArts*.

First a name for the model must be specified. We call it `eQED` like QED only for `e`(electron and positron).[6] All particles of the theory must be assigned variables, which have to be indiced. The conventions for the code are described in appendix A-1. Particles with Lorentz and/or gluon indices must carry those in addition to the field index in the Feynman rules. So vector particles (and gluon ghosts) must have a second index, the Lorentz index (gluon index), gluons even a third one, the gluon index. We use `V[1,li_]`, `F[1]` and `F[-1]` for the photon, the electron and the positron, respectively.

Vertices and propagators must be specified with heads `PV` in lists of equations called `CoupList[`*modelname*`]` and `PropList[`*modelname*`]`. The left hand side of each equation contains the particles including their indices and the momenta they carry, the right hand side is the analytic expression corresponding to the Feynman rule. These can contain non-commuting objects obeying Dirac or Gell-Mann algebra in order to suppress spinor and color indices. These objects must have suitable heads so that they can be treated appropriately in the further evaluation process beyond *FeynArts* (see appendix A-3) and they must be collected in a factor with head `NonCommutative`. Spinors must be of the form `LeptonSpinor[`p`,`m`]` for leptons and `QuarkSpinor[`p`,`m`]` for quarks with momentum p and and mass m.

Apart from that, the exact manner of entering the code for a model (e.g. how the variables are called) is arbitrary and can be changed for one's own purposes. The models provided with *FeynArts* were entered in a code so that an automatic evaluation of the created amplitudes with *FeynCalc* is possible and straightforward. This is also the reason why the denominator factors of the propagators are given in form of a function `PropagatorDenominator` with two variables, the momentum and the mass. (`PropagatorDenominator[p,m]` stands for $1/(p^2 - m^2)$.) The exact conventions for the Feynman rules

[5] You can contact us if you have problems entering a new model.

[6] This model is not entered in *FeynArts*, but it can be obtained from model `QE` by setting the options `Generation2` and `Generation3` to `False`.

we have entered are given in appendix A-3. So e.g. `me` and `el` stand for electron mass and elementary charge, respectively.

For `eQED` in 't Hooft-Feynman gauge we would enter: (direction of momentum)

```
PropList[eQED] =
{
 Prop[in][ F[-1],F[1],P_ ] ==
      PV[ NonCommutative[ DiracSlash[P] + me ]
            I PropagatorDenominator[P,me] ],
 Prop[ex][ F[ 1],P_ ] ==
      PV[ NonCommutative[ LeptonSpinor[P,me] ] ],

Prop[in][ V[1,li1_],V[1,li2_],P_ ] ==
      PV[ - MetricTensor[li1, li2]
            I PropagatorDenominator[P,0] ],
 Prop[ex][ V[1,li_],P_ ] ==
      PV[ PolarizationVector[ P,li ] ],
};
```

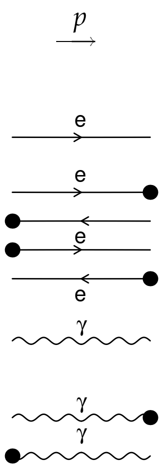
$$\frac{i(\not p + m_e)}{p^2 - m_e^2}$$

$u(p, m_e), v(-p, m_e)\Big\{$

$\overline{u}(p, m_e), \overline{v}(-p, m_e)\Big\{$

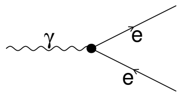
$$\frac{-ig_{\mu\nu}}{p^2}$$

$\epsilon_\mu(p), \epsilon_\mu(p)\Big\{$




```
CoupList[eQED] =
{
 Coup[{V[1,li_],P1_},{F[-1],P2_},{F[1],P3_}] ==
     PV[ NonCommutative[DiracMatrix[li]], I*el ]
};
```

$ie\gamma_\mu$



For the vertices the sequence of the particles is arbitrary. All momenta p*i* are counted incoming. For the propagators the rules are the following: In the internal propagators (`Prop[in]`) the right particle (second argument) is propagating in the direction of its momentum *p*, the left particle (first argument) must be the appropriate antiparticle. For all wave functions (`Prop[ex]`) of a particle-antiparticle pair only one Feynman rule has to be entered, the one for an incoming particle or antiparticle. In the case of bosons outgoing fields get the complex conjugate wave function. For fermions in general four different wave functions exist. Again only the Feynman rule for an incoming fermion or antifermion must be given (as shown for the electron in the example above). How the different wave functions for incoming/outgoing fermion/antifermion are distinguished is described in appendix A-3. internal form (and a `v` instead of a `u` in the output form).

If *FeynArts* is to draw different sorts of lines with particle labels, the corresponding specifications have to be made. Each particle gets three attributes: the type, the direction and the label of the line. Allowed line types are `straight`, `wavy`, `cycloid`, `dashed` and `dotted`, allowed directions are `forward`, `backward` and `none`. For `eQED` we enter:

```
LineSpec[ F[ 1] ] = { straight,forward ,"e";}
LineSpec[ F[-1] ] = { straight,backward,"e";}
LineSpec[ V[ 1] ] = { wavy,none,Greek["g"] }
```

In order to make *FeynArts* use Greek or subscripted labels in the Feynman graphs the functions `Greek` and `Sub` are provided. To get the desired Greek letter you have to give its equivalent in `Courier` font as done above for the "γ" for the `Symbol` font.

`Greek[` *string* `]`	makes `Paint` use the `$...SymbolFonts` as specified in `FeynArts.m`.
`Sub[` *string1*`,` *string2* `]`	makes `Paint` produce graph labels of the form $string1_{string2}$. *string1* and *string2* can both be of the form `Greek[`*string*`]`.

The use of `Greek` and `Sub`

A-3 Translation of *FeynArts* Output

In order to be able to incorporate results of *FeynArts* and *FeynCalc* into one's calculation we give here our conventions to translate *FeynArts* output. `CreateFeynAmp` creates amplitudes with a very explicit internal representation which can be interpreted by *FeynCalc*. This form, as well as the output form which appears on screen for the *FeynArts* user and the corresponding mathematical and physical notations are given in the following tables. Expressions in typewriter style are literal output, whereas expressions in italics are variables (filled in by *FeynArts*).

■ A-3.1 Conventions

We have chosen the conventions such that the appearance of the factors $i/(2\pi)^4$ is minimized. Consequently all propagators, vertices and wavefunctions are free of these factors but every loop integral carries a factor $1/(2\pi)^4$ (or $\frac{\mu^{4-\$D}}{(2\pi)^{\$D}}$ in $\$D$ dimensions). This results in a global factor $1/(2\pi)^4N$ (or $\frac{\mu^{(4-\$D)N}}{(2\pi)^{\$DN}}$) for an N-loop graph. Every propagator and vertex contributes a factor i. In order to obtain real S-matrix elements (apart from absorptive parts in higher orders) we multiply each Feynman diagram by a factor $-i$. This factor $-i$ and the external wavefunctions are omitted for truncated amplitudes (option `Truncated` set to `True`). The overall momentum conservation delta-function is understood implicitely and not given explicitely.

Apart from extra minus signs for closed fermion and ghost loops Feynman diagrams which differ by permutations of the external fermion lines get relative minus signs. These are fixed as follows. The incoming particles are numbered from 1 to i, the outgoing ones from $i+1$ to n. Every outgoing spinor or incoming antispinor is connected via fermion lines to an incoming spinor or outgoing antispinor. This defines the spinor chains. Writing all the spinor chains of a given Feynman diagram in an arbitrary order fixes a sequence of particle numbers corresponding to the spinors. If this sequence differs by an odd permutation from the numerically ordered sequence of all particle numbers associated with fermions the graph gets an extra minus sign, otherwise not. For example consider a graph with four external fermions with particle numbers $\{1,2,3,4\}$. Let the spinor chains be $\overline{u}(\mathrm{particle3})\ldots v(\mathrm{particle2})\cdot\overline{v}(\mathrm{particle1})\ldots u(\mathrm{particle4})$. This gives the sequence $\{3,2,1,4\}$ which is an odd permutation of $\{1,2,3,4\}$. Consequently this graph is multiplied with a minus sign.

■ A-3.2 Tensors with Lorentz indices

internal form	output form	translation
`li`*n*	`li`*n*	Lorentz index: `li`*n* $= 1\ldots4 \quad n\in N$
`MetricTensor[li`*n*`,li`*m*`]`	`g[li`*n*`,li`*m*`]`	the metric tensor $g^{\mathrm{li}n,\mathrm{li}m}$
`PolarizationVector[p`*i*`,li`*j*`]`	`ep[p`*i*`,li`*j*`]`	wave function $\epsilon^{\mathrm{li}j}(\mathrm{p}i)$ of incoming vector bosons or the corresponding outgoing antiparticle (gluons get index `gi`*k* in addition)
`Conjugate[ PolarizationVector[p`*i*`,li`*j*`] ]`	`ep(*)[p`*i*`,li`*j*`]`	wave function $(\epsilon^{\mathrm{li}j})(\mathrm{p}i)$ of outgoing vector bosons or the corresponding outgoing antiparticle (gluons get index `gi`*k* in addition)
`FourVector[p`*i*`,li`*j*`]`	p*i*,li*j*	the four dimensional vector `p`*i* with Lorentz index `li`*j*: $p_i^{\mathrm{li}j}$

■ A-3.3 Elements of the Dirac algebra

internal form	output form	translation
`DiracMatrix[li`*n*`]`	`ga[li`*n*`]`	Dirac gamma matrix $\gamma_{\mathrm{li}n}$
`DiracMatrix[5]`	`ga[5]`	the Dirac matrix $\gamma_5 = i\gamma_0\gamma_1\gamma_2\gamma_3$
`ChiralityProjector[+1]`	`(1+ga[5])/2`	$\omega_+ = \frac{1}{2}(1+\gamma_5)$
`ChiralityProjector[-1]`	`(1-ga[5])/2`	$\omega_- = \frac{1}{2}(1-\gamma_5)$
`DiracSlash[`*expr*`]`	`gs[`*expr*`]`	contraction $\gamma_\mu (expr)^\mu$
`DiracTrace[`*expr*`]`	`tr[`*expr*`]`	trace over products of Dirac matrices contained in *expr*
`LeptonSpinor[p`*i*`,m`*j*`]`	`u(p`*i*`,m`*j*`)`	spinors of a lepton $u, \overline{u}$,
	`v(p`*i*`,m`*j*`)`	$v, \overline{v}$ with momentum `p`*i* and mass `m`*j*
`QuarkSpinor[p`*i*`,m`*j*`]`	`u(p`*i*`,m`*j*`)`	spinors of a quark $u, \overline{u}$
	`v(p`*i*`,m`*j*`)`	$v, \overline{v}$ with momentum `p`*i* and mass `m`*j*

A *spinor* (`LeptonSpinor` or `QuarkSpinor`) is translated considering the sign of its momentum `p`*i* and its position in a `Dot`-product (spinor chain). The sign of the momentum always indicates its direction with respect to the flow of fermion number:

$$\begin{array}{lcllcl} spinor\texttt{[p,m]} \cdot expr & = & \overline{u}(p) & expr \cdot spinor\texttt{[p,m]} & = & u(p) \\ spinor\texttt{[-p,m]} \cdot expr & = & \overline{v}(p) & expr \cdot spinor\texttt{[-p,m]} & = & v(p) \end{array}$$

■ A-3.4 Elements of the SU(3) algebra

internal form	output form	translation
`gi`*a*	`gi`*a*	gluon index: `gi`*a* $=$ 1… 8 for the SU(3) $a \in N$
`SU3Delta[gi`*a*`,gi`*b*`]`	`d[gi`*a*`,gi`*b*`]`	Kronecker symbol $\delta^{\mathrm{gi}a,\mathrm{gi}b} = \begin{cases} 1, & \mathtt{gi}a = \mathtt{gi}b \\ 0, & \text{else} \end{cases}$
`SU3F[gi`*a*`,gi`*b*`,gic]`	`f[gi`*a*`,gi`*b*`,gic]`	structure constants $f^{\mathrm{gi}a,\mathrm{gi}b,\mathrm{gic}}$ of the SU(3)
`GellMannMatrix[gi`*a*`]`	`la[gi`*a*`]`	the Gell-Mann matrix $\lambda^{\mathrm{gi}a}$
`GellMannTrace[`*expr*`]`	`tr[`*expr*`]`	trace over products of Gell-Mann matrices contained in *expr*

■ A-3.5 Miscellaneous

internal form	output form	translation
`PropagatorDenominator[p`i`,m`a`]`	$\frac{1}{\mathtt{p}i^2-\mathtt{m}a^2}$	$\frac{1}{\mathtt{p}i^2-\mathtt{m}a^2}$
`FeynAmpDenominator[`*expr*`]`	`Integral [q[1],...]` *expr*	$\int d\mathtt{q1}...expr$ (integral sign is supressed in the internal form)
`Mu`	`Mu`	renormalization parameter
`GaugeXi[`*f*`]`	`xi[`*f*`]`	gauge parameter of the 't Hooft gauge *f* stands for the boson and can take values `A, W, Z`
`$D`	`$D`	space-time-dimension
`QMM`*ij*	`QMM`*ij*	entries of the quark mixing matrix for fermions coupling to a $W^+(i,j=1..3)$
`Conjugate[QMM`*ij*`]`	`QMM(*)`*ij*	for fermions coupling to a $W^-(i,j=1..3)$
`p`*i*	`p`*i*	momenta of incoming particles
`k`*i*	`k`*i*	momenta of outgoing particles
`q`*i*	`q`*i*	loop momenta

■ A-3.6 Mathematical and physical constants

`I` : imaginary unit i `Pi` : $\pi = 3.1415\ldots$

In *FeynArts* all parameters are written with capital letters so that they can be directly fed into a standard fortran program (after having worked on the output with *FeynCalc*.

`EL` : elementary charge `GS` : strong coupling constant `SW,CW` : sine and cosine of the weak mixing angle

Masses:

`ME` : electron	`MM` : muon	`ML` : tauon
`MU` : up-quark	`MC` : charm-quark	`MT` : top-quark
`MD` : down-quark	`MS` : strange-quark	`MB` : bottom-quark
`MZ` : Z-boson	`MW` : W-boson	`MH` : physical Higgs
`MQ` : 'toy'-quark		

A-4 *FeynArts* Reference Guide

This reference guide briefly describes the functions of *FeynArts*. If a table of options is given the left column gives the option name, the second the default value and the third a short description.

All *FeynArts*-objects that are not contained in this section are mathematical objects and are described in appendix A-3.

■ `AntiParticle`

`AntiParticle[`*mod*`][`*part*`]` gives the anti-particle of *part* in the model *mod*.

mod is a descriptive symbol that includes the values of the model options in binary code. ■ `AntiParticle` doesn't work before the specified model is initialized. ■ The initialized models *mod* can be called with `InitializedModels`.

■ `AutoEdit`

`AutoEdit` is an option for `Paint`.

Calling `Paint` with the option `AutoEdit -> True` will cause `Paint` to stop program execution and invoke `FeynEdit` whenever an unknown topology occurs. ■ See page 20.

■ `Bicycle`

`Bicycle` is an option for `CreateTopologies`.

■ `Bicycle -> True` includes descendents of the start topology of fig. 1e. ■ The generated topologies are one-particle reducible 2-loop graphs. ■ See page 10.

■ `Boxes`

`Boxes` is an option for `CreateTopologies`.

`Boxes -> True` includes 4- and more-point functions in n-loop topologies. ■ See page 10.

■ `Coup`

`Coup[`*i1, i2, …* `]` is the representation of a vertex in the model-defining file.

The *i1, i2, …* are the incoming particles and have to be given in the form: {*field, momentum*}. ■ See page 36.

■ `CoupList`

`CoupList[`*mod*`]` is the list collecting all possible couplings of the model *mod*.

`CoupList[`*mod*`]` is defined in the file `model.`*mod*. ■ The defined couplings in `CoupList` are given in form of equations. ■ See page 35.

■ CreateFeynAmp

`CreateFeynAmp[` *graphlist* `]` translates *graphlist*, which is a list of Feynman graphs, into amplitudes.

graphlist must be exactly of the form as created by `InsertFields`. The head of *graphlist* tells `CreateFeynAmp` which Feynman rules to apply. ■ The following options can be given:

`ScreenMessages`	`True`	whether to print the messages about the translation on screen.
`Truncated`	`False`	whether to produce the amplitudes without wave functions.
`FeynmanGauge`	`True`	choice of the parameter `GaugeXi`
`$DDimension`	`False`	choice of prefactor

■ See page 25.

■ CreateTopologies

`CreateTopologies[`*l,e*`]` creates all topologies with *l* loops and *e* external legs.

The Topologies are created in list form with the appropriate combinatorial factor. ■ The following options can be given:

`Tadpoles`	`True`	whether to include tadpole graphs
`SelfEnergies`	`True`	whether to include graphs with two-point functions
`Triangles`	`True`	whether to include graphs with three-point functions
`Boxes`	`True`	whether to include one-loop graphs with four- and more-point functions
`WFCorrections`	`True`	whether to include graphs with tadpoles or self energies on external lines (only works for graphs with irreducible start topologies)
`Theta`	`True`	whether to include descendents from fig.1c
`Eight`	`True`	whether to include descendents from fig.1d
`Bicycle`	`False`	whether to include descendents from fig.1e

■ See page 9.

■ Eight

`Eight` is an option for `CreateTopologies`.

`Eight -> True` includes descendents from fig.1d in the topology creation of 2-loop topologies. ■ See page 10.

■ ElectronHCoupling

`ElectronHCoupling` is an option for `InsertFields`.

`ElectronHCoupling -> False` sets all couplings of the Higgs fields to electrons to 0. ■ The electron-Higgs couplings are also included in the class of light-fermion-Higgs couplings, i.e. electron-Higgs couplings are 0 if `LightFHCoupling -> False` even if `ElectronHCoupling` is set to `True`. ■ See page 16.

■ FeynAmp

`FeynAmp` is the head of a single Feynman amplitude.

The internal form (`InputForm`) of `FeynAmp` contains two or more arguments: the name of the translated graph (head `GraphName`), the integration momenta on the loops of the loops of the graph (only present if loop graphs are translated) and the analytical expression for the graph. ■ The `OutputForm` of `FeynAmp` is of the form `FeynAmp[`*name,numbers*`][`*AmplitudesInOutputForm*`]`. ■ See page 26.

■ FeynAmpList

`FeynAmpList` is the head of the result of `CreateFeynAmp` applied to a list of Feynman graphs (head `TopologyList`).

See page 29.

■ FeynEdit

`FeynEdit[`*expr*`]` invokes an interactive procedure to fix or change the appearance of a topology when drawn with `Paint`.

`FeynEdit[`*graph*`]` starts editing a Feynman graph of a certain process (created by `InsertFields`). ■ `FeynEdit[`*top, n, m*`]` starts editing a topology for a process with *n* incoming and *m* outgoing particles. ■ See page 22.

■ FeynmanGauge

`FeynmanGauge` is an option for `CreateFeynAmp`.

`FeynmanGauge` sets the value of the parameter `GaugeXi[`*f*`]`, where *f* is one of the gauge fields `A, Z, W`. ■ See page 25.

■ FullNumbering

`FullNumbering` is an option for `Paint`.

`FullNumbering` will cause `Paint` to append a third number to the graph labels that counts the graphs drawn so far. ■ See page 20.

■ Generation1
■ Generation2
■ Generation3

`Generation`*N* are options for `InsertFields`.

`Generation`*N* `-> True` initializes a model with the *N*th fermion generation included. ■ Default is to include all three fermion generations. ■ See page 16.

■ Graph

`Graph` is the head for a list of rules that perform substitutions for dummy fields which appear in a topology with fields of a certain model.

The objects with head `Graph` are elements of a list of `Insertions`. ■ Every `Graph` corresponds to one single Feynman graph of a certain `Topology`. ■ See page 14.

■ GraphName

`GraphName` is the head of an object that labels a special Feynman amplitude. It contains the `ProcessName`, the position of the graph (in a `TopologyList`), the position of the graph in the list of `Insertions` for one `Topology` and the position of the graph in the complete list of amplitudes (head `FeynAmpList`).

See page 29.

■ GraphsPerRow

`GraphsPerRow` is an option for `Paint`.

See page 20.

■ Greek

`Greek`[*string*] is used in the `LineSpec` function for the specification of a particle label in a model file. It makes `FeynArts` label the propagator with the corresponding symbol of the specified symbol font.

See page 36.

■ GrayBackground

`GrayBackground` is an option for `Paint`.

The result depends very strongly on the resolution of the screen or the printer. ■ See page 20.

■ InitializedModels

`InitializedModels` is a list that contains the descriptive symbols for all initialized models.

■ InsertFields

`InsertFields`[*top,extli*] inserts particles into the topologies of *top* so that they are compatible with the list of external particles *extli* and a given model.

The model that is used for inserting is given by default or by additional (optional) arguments. The default model is the electroweak standard model. ■ The topologies have to be given in the conventions as created by `CreateTopologies`. ■ The external particles have to be given in the form {*field_1*,... , *field_i*} `->` {*field_i+1*,... , *field_n*}. where the first list contains the incoming and the second list the outgoing fields. ■ The following options can be given:

`Model`	`{SM}`	chooses the models (in brackets) from which particles are inserted
`Generation1`	`True`	whether to include the 1st fermion generation
`Generation2`	`True`	whether to include the 2nd fermion generation
`Generation3`	`True`	whether to include the 3rd fermion generation
`ElectronHCoupling`	`True`	`False` turns the electron-Higgs couplings off
`LightFHCoupling`	`True`	`False` turns all fermion-Higgs couplings off except to the top quark
`QuarkMixing`	`False`	`False` sets the quark mixing matrix equal to the identity matrix
`UnitaryGauge`	`False`	whether to use the unitary gauge
`RemoveEmptyTops`	`True`	whether to remove topologies for which no insertions are found
`ScreenMessages`	`True`	whether to print the insertion messages on screen.
`ProcessName`	`Automatic`	The name you want to give to the process.

■ See page 13.

■ LineSpec

`LineSpec`[*part*] is a function that describes the appearance of a propagator with particle *part* in the graphics.

`LineSpec`[*part*] usually is a list with three elements: line type (`straight, wavy, cycloid, dashed, dotted`), specification of arrow on line (`forward, backward, none`) and label to appear in the graphics for that particle (usually a string constant). ■ See page 36.

■ LightFHCoupling

`LightFHCoupling` is an option for `InsertFields`.

`LightFHCoupling -> False` sets all couplings of light fermions (these are all fermions except the top quark) equal to zero. ■ See page 16.

■ Model

Model is an option for InsertFields.

Model has to be set to a list that contains all the models from which particles should be inserted. ■ For each model specified, a file model.*name* must exist. ■ All possible models can be combined, no matter if the result makes any physical sense. ■ See page 16.

■ NonCommutative

NonCommutative is the head of non-commutative objects inside PV. The arguments of NonCommutative are multiplied using the Dot-product.

■ Paint

Paint[*graphlist*] draws the Feynman graphs or topologies contained in *graphlist*.

graphlist must be a list of insertions as produced by InsertFields or a list of topologies produced by CreateTopologies. ■ The following options can be given:

GraphsPerRow	3	how many graphs per row
RowsPerSheet	3	how many rows of graphs on one sheet
ScreenOutput	True	whether to show the graphs on screen
PSFileOutput	True	whether to write the graphics output to a postscript file
ScreenMessages	True	whether to print the messages which graph is being painted.
AutoEdit	False	set to True the FeynEdit modus is automatically entered
FullNumbering	False	additional graph numbering
GrayBackground	False	whether to use grey background
ShowGrid	False	whether to show a grid
ShowPointNames	False	whether to show the vertex names
PropagatorThickness	Automatic	how thick to draw the propagator
VertexSize	Automatic	which size to use for the vertices

■ See page 19.

■ Process

Process is one of the descriptive rules that appear in the head of a FeynAmpList or a TopologyList if insertions have been made.

In a TopologyList created by InsertFields, Process will be of the Form: Process -> {*f1*, *f2*, … }->{*g1*, *g2*, … }, where the *fi* are the incoming and the *gi* are the outgoing fields. ■ In a FeynAmpList created by CreateFeynAmp, the fields will be replaced by lists of three elements: { field, momentum, mass }. ■ See page 14.

■ ProcessName

ProcessName is an option for InsertFields.

ProcessName sets the name of the graphics file that is used for saving FeynArtsGraphics produced with Paint. The extension is always ".gra". ■ See page 16.

■ Prop

Prop[*type*][*f1*, *f2*, *mom*] is the representation of a propagator for the definition of analytical expressions.

type is in or ex for internal propagators or wave functions respectively. ■ *f1* and *f2* are antiparticle and particle. ■ The field (particle *f2*) propagates with momentum *mom*. ■ See page 36.

■ `Propagator`

`Propagator[`*type*`][`*descr*`]` is the representation of a propagator in a topology.

type can be `ex`, `inc`, `out` or `l`*i* where *i* is an integer. ■ `Propagator[`*type*`][`*p1*, *p2*`]` denotes a propagator running from point *p1* to point *p2*. ■ `Propagator[`*type*`][`*p1*, *p2*, *f*`]` denotes a propagator with field *f*. ■ `Propagator[`*type*`][`*p1*, *p2*, *f*, *q*`]` denotes a propagator with field *f* carrying a momentum *q*. ■ See page 9.

■ `PropagatorThickness`

`PropagatorThickness` is an option for `Paint`.

■ `PropList`

`PropList[`*mod*`]` is the list collecting all possible propagators of the model *mod*.

`PropList[`*mod*`]` is defined in the file `model.`*mod*. ■ The defined propagators in `PropList` are given in form of equations. ■ See page 35.

■ `PSFileOutput`

`PSFileOutput` is an option for `Paint`.

■ `PV`

`PV` is the head of an object that contains the analytic expression for a `Propagator` or a `Vertex`.

`PV` contains one single expression. ■ The arguments of all `PV` in a Feynman graph are multiplied to construct the amplitude. ■ For arguments of `PV` belonging to the Dirac algebra or the group SU3, non-commutative multiplication is used.

■ `QuarkMixing`

`QuarkMixing` is an option for `InsertFields`.

`QuarkMixing` determines the choice of the Kobayashi-Maskawa-Matrix. ■ `QuarkMixing -> False` sets the Kobayashi-Maskawa-Matrix to the identity matrix. ■ See page 16.

■ `RemoveEmptyTops`

`RemoveEmptyTops` is an option for `InsertFields`.

See page 16.

■ `RowsPerSheet`

`RowsPerSheet` is an option for `Paint`.

See page 20.

■ `ScreenMessages`

`ScreenMessages` is an option for `CreateFeynAmp`, `InsertFields` and `Paint`.

`ScreenMessages -> False` suppresses all run-time messages that usually appear on the screen.

■ `ScreenOutput`

`ScreenOutput` is an option for `Paint`.

See page 20.

■ `SelfEnergies`

`SelfEnergies` is an option for `CreateTopologies`.

`SelfEnergies -> True` includes graphs with 2-point functions when creating the topologies. ■ See page 10.

■ `ShowGrid`

`ShowGrid` is an option for `Paint`.

`ShowGrid -> True` is used when invoking `Paint` via `FeynEdit`. ■ See page 20.

■ `ShowPointNames`

`ShowPointNames` is an option for `Paint`.

`ShowPointNames -> True` displays the symbols that represent a vertex and the symbols for the centers of the propagators. ■ See page 20.

■ `Sub`

`Sub`[*s1*,*s2*] denotes a subscripted symbol for the `Paint` routine.

`Sub` is used in the `LineSpec` function to define the string that is printed as a propagator label for a certain field. ■ `Sub`[*s1*,*s2*] means that string *s2* appears as subscript of the string *s1*. ■ See page 36.

■ `Tadpoles`

`Tadpoles` is an option for `CreateTopologies`.

`Tadpoles -> True` includes graphs with 1-point functions when creating the topologies. ■ See page 10.

■ `Theta`

`Theta` is an option for `CreateTopologies`.

`Theta -> True` includes descendents from fig.1c when creating 2-loop topologies. ■ See page 10.

■ `Topology`

`Topology`[*comb*] [*Pr1, Pr2,* …] is the representation of a pure topology with combinatorial factor *comb*. The *Pri* are the propagators of the topology and have head `Propagator`.
`Topology`[*Pr1, Pr2,* … *, ins*] is the representation of a topology with insertions *ins*. *ins* is an object with head `Insertions` and represents a list of Feynman graphs originating from the topology.

See page 13.

■ References

[1] S. Wolfram, '*Mathematica* — a system for doing mathematics by computer', Addison-Wesley, New York, 1988.

[2] J. Küblbeck, M. Böhm and A. Denner, *Comp. Phys. Comm.* **60** (1990) 165.

[3] 'Computeralgebraische Erzeugung von Feynmangraphen allgemeiner renormierbarer Eichtheorien', H. Eck, diplom thesis, Universität Würzburg, 1990.

[4] 'The application of *FeynArts* — a computeralgebraic program for the generation of Feynman graphs — to the process $e^+e^- \to ZH$', J. Küblbeck, Ph. D. Thesis, Universität Würzburg, 1991.

[5] R. Mertig, M. Böhm and A. Denner, *Comp. Phys. Comm.* **64** (1991) 345.