

Update of the Binoth Les Houches Accord for a standard interface between Monte Carlo tools and One-Loop Programs

S. Alioli^a, S. Badger^b, B. Biedermann^c, F. Boudjema^d,
G. Cullen^e, A. Denner^f, S. Dittmaier^g, R. Frederix^h,
S. Frixione^h, M.V.Garzelliⁱ, E.W.N. Gloverⁿ, N. Greiner^k,
G. Heinrich^k, V. Hirschi^j, S. Höche^l, J. Huston^m, N. Kauer^o,
F. Kraussⁿ, G. Luisoni^k, D. Maîtreⁿ, F. Maltoni^p, P. Nason^r,
C. Oleari^r, G. Ossola^s, G. Passarino^t, R. Pittau^q, S. Plätzer^u,
S. Pozzorini^v, L. Reina^w, T. Robens^x, M. Schönherrⁿ,
F. Siegert^g, J.F.von Soden-Fraunhofen^k, F. Tackmann^u,
F. Tramontano^y, P. Uwer^c, G. Salam^h, P. Skands^h, J. Winter^k,
V. Yundin^b, G. Zanderighi^z

^a*Lawrence Berkeley National Laboratory and University of California, Berkeley,
CA 94720, USA*

^b*The Niels Bohr Institute, University of Copenhagen, DK-2100 Copenhagen,
Denmark*

^c*Humboldt-Universität zu Berlin, Institut für Physik, D-12489 Berlin, Germany*

^d*LAPTH, Université de Savoie and CNRS, F-74941 Annecy-le-Vieux, France*

^e*DESY, Zeuthen, Germany*

^f*Universität Würzburg, Institut für Theoretische Physik und Astrophysik, D-97074
Würzburg, Germany*

^g*Albert-Ludwigs-Universität Freiburg, Physikalisches Institut, D-79104 Freiburg,
Germany*

^h*CERN/PH, CH-1211 Geneva 23, Switzerland*

ⁱ*University of Nova Gorica, SI 5000 Nova Gorica, Slovenia*

^j*EPFL Lausanne, Switzerland*

^k*Max Planck Institute for Physics, 80805 Munich, Germany*

^l*SLAC, Stanford University, Stanford, CA 94309, USA*

^m*Michigan State University, East Lansing, MI 48840, USA*

ⁿ*Institute for Particle Physics Phenomenology, University of Durham, Durham,
DH1 3LE, UK*

^o*Department of Physics, Royal Holloway, University of London, Egham TW20
0EX, UK*

^p*CP3, Université Catholique de Louvain, 1348 Louvain-la-Neuve, Belgium*

^q*Departamento de Fisica Teorica y del Cosmos CAFPE, Universidad de Granada,
E-18071 Granada, Spain*

^r*Università di Milano-Bicocca, 20126 Milano, Italy*

^s*New York City College of Technology, City University of New York, Brooklyn NY
11201, USA*

^t*NFN, Sez. di Torino and Dip. Fisica Teorica, Università di Torino, Torino, Italy*

^u*DESY Hamburg, Germany*

^v*University of Zurich, Institute for Theoretical Physics, CH-8057 Zurich,
Switzerland*

^w*Florida State University, Tallahassee, FL 32306-4350, USA*

^x*University of Dresden, Germany*

^y*Dipartimento di Scienze Fisiche, Università degli studi di Napoli “Federico II”
and INFN, Sezione di Napoli, I-80125 Napoli, Italy*

^z*Rudolf Peierls Centre for Theoretical Physics, 1 Keble Road, OX13PN, Oxford,
UK*

draft version 1 July 2013

Abstract

We present an update of the Binnoth Les Houches Accord (BLHA) to standardise the interface between Monte Carlo programs and codes providing one-loop matrix elements.

Key words: Monte Carlo tools, one-loop computations, Les Houches Accord, automation

1 Introduction

The past years have seen an enormous progress in the development of programs providing next-to-leading order corrections for multi-particle final states. This is due to new developments concerning the calculation of one-loop amplitudes as well as important progress on the Monte Carlo side to account for real radiation at NLO. The modular structure of NLO calculations allows to share the tasks between a “One-Loop Provider (OLP)”, providing the virtual corrections, and a Monte Carlo program (MC) taking care of all the parts which do not involve loops. To facilitate the cross-talk between those two engines, a standard interface has been worked out during the workshop on Physics at

TeV Colliders at Les Houches in June 2009, called the “Binoth Les Houches Accord (BLHA)” [1].

Meanwhile, the use of this interface [2–6] and further developments in OLP and MC codes have brought up the necessity to extend it with further options. The aim of this article is to provide a public document where an update of the BLHA is proposed and conventions are defined to pass parameters, calculational schemes etc., and to return less inclusive information, such as matrix elements which are not summed over all colours and helicities.

2 Existing features of the interface

We do not aim at an exhaustive description of the complete framework of the interface here, referring to [1] for more details. However we sketch the main features any extension will build upon.

A flowchart of the setup between Monte Carlo program and One Loop Provider (where the new functions defined in BLHA version 2 are included already) is shown in Fig. 1.

The interaction between an OLP and a MC proceeds in two phases: the *initialisation* phase and the *run-time* phase. In the initialisation phase, the MC creates a file containing information about the setup and the subprocesses it will need from the OLP to perform the computation. A subprocess can be either a partonic subprocess or a component thereof (e.g. a specific helicity amplitude or a colour partial amplitude). The particles are identified by specifying their particle data group (PDG) code. This file is called **order file**. An example ($Z + 1$ jet) for an order file using the original standards written by SHERPA, called `OLE_order.1h`, is shown in Fig. 2.

The OLP then reads the order file and checks availability for each item. Then it returns a **contract file** telling the MC what it can provide. A contract file generated by GoSAM as a response to SHERPA’s contract file, looks like the one in Fig. 3. From this file, one can already see where an upgrade of the interface is clearly needed: As the original standard did not contain a function which allows to pass parameters, the definition of masses and widths is marked as “Ignored by OLP” in the contract file. Certainly the parameters are passed in the actual calculation, but in a non-standardised way, as an individual agreement between the particular MC and OLP. How to define a standard for the passing of parameters is described in Section 3.1.

The first integer label after each subprocess specifies that this subprocess contains only one component (if it was composed e.g. of several helicity configu-

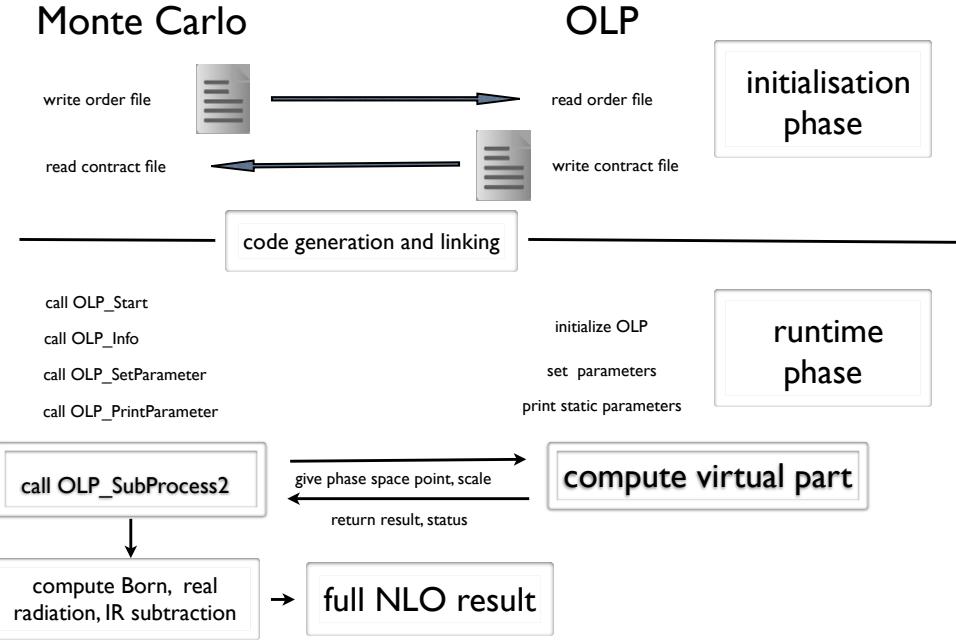


Fig. 1. Interplay between Monte Carlo program and One Loop Provider (OLP). In the initialisation phase of the interface, the OLP “signs” an order file placed by the MC. At runtime, the Monte Carlo program provides Born, real radiation part minus infrared subtraction terms and integrated subtraction terms. The OLP provides the virtual amplitude for each phase space point. The phase space integration is done by the Monte Carlo program.

rations to be evaluated separately, this first label would be an integer larger than one). The second integer acts as a label for each subprocess, used at runtime to call the individual subprocesses.

After the contract has been *signed*, the communication between MC and OLP proceeds via function calls. At present, there are two functions which allow the transfer of information between the two programs. One is the function `OLP_Start(char* fname, int* ierr)` which should be called by the MC before starting the runtime phase. The character string in the first argument contains the name of the contract file. The integer in the second argument is set to 1 by the function call if the contract file is accepted. In case of failure, the second argument is different from one, and an error message of the type `Error: can not handle contract file` should be issued.

The second function which is already in place, but will have a different argument list with the new standards, is the function `OLP_EvalSubProcess`. The parameters to be passed to the `OLP_EvalSubProcess` function according to the original version of the interface are (in this order):

```

# OLE_order.lh
# Created by Sherpa-1.4.1

MatrixElementSquareType CHsummed
CorrectionType QCD
IRregularisation DRED
AlphasPower 1
AlphaPower 2
OperationMode CouplingsStrippedOff

Z_mass 91.118
Z_width 2.49
W_mass 80.419
W_width 2.0476
sin_th_2 0.221051079833

# process list
1 -1 -> 11 -11 21
21 1 -> 11 -11 1
21 -1 -> 11 -11 -1
2 -2 -> 11 -11 21
21 2 -> 11 -11 2
21 -2 -> 11 -11 -2

```

Fig. 2. Example of an order file for the process $Z + \text{jet}$, created by SHERPA.

- the integer label of the subprocess (as given in the contract file)
- an array containing the components of the momenta. The momenta are placed in a one dimensional array, where physical scattering kinematics is used, i.e. $k_1 + k_2 = k_3 + \dots + k_m$. For each particle, the kinematics is specified by a 5-tuple: $(E_j, k_j^x, k_j^y, k_j^z, M_j)$. Thus a full m -particle event is specified by an array of $5*m$ double precision numbers filled with the m 5-tuples, ordered by the particle labels.
- the renormalisation scale, μ_R , as a double precision number, or an array of scales, if different scales need to be passed.
- the strong coupling $\alpha_s(\mu_R)$, where $\alpha_s(\mu_R) = 1$ can be used to indicate that the MC multiplies the returned values with the adequate coupling constants.
- the array where the computed results are returned.

The returned array is expected to contain at least four real-valued double precision numbers

PoleCoeff2, PoleCoeff1, PoleCoeff0, BornSquare
--

which correspond to the colour- and helicity-summed (resp. averaged for the

```

# vim: syntax=olp
#@OLP GOSAM 1.0
#@IgnoreUnknown True
#@IgnoreCase False
IRregularisation DRED | OK
AlphaPower 2 | OK
sin_th_2 0.221051079833 | OK # Ignored by OLP
Z_width 2.49 | OK # Ignored by OLP
Z_mass 91.118 | OK # Ignored by OLP
W_mass 80.419 | OK # Ignored by OLP
CorrectionType QCD | OK
AlphasPower 1 | OK
W_width 2.0476 | OK # Ignored by OLP
OperationMode CouplingsStrippedOff | OK
MatrixElementSquareType CHsummed | OK
1 -1 -> 11 -11 21 | 1 3
21 1 -> 11 -11 1 | 1 4
21 -1 -> 11 -11 -1 | 1 5
2 -2 -> 11 -11 21 | 1 0
21 2 -> 11 -11 2 | 1 1
21 -2 -> 11 -11 -2 | 1 2

```

Fig. 3. Example of a contract file for the process $Z + \text{jet}$, created by GoSAM. As the original interface did not define a function which allows to pass parameters, the definition of masses and widths is marked as “Ignored by OLP” in the contract file.

initial state) terms A_2 , A_1 , A_0 , $|\text{Born}|^2$.

The conventions for the overall prefactor are as in the original proposal [1]: the general structure of the virtual correction is given by

$$\mathcal{I}(\{k_j\}, \text{R.S.}, \mu_{\text{R}}^2, \alpha_s(\mu_{\text{R}}^2), \alpha, \dots) = C(\epsilon) \left(\frac{A_2}{\epsilon^2} + \frac{A_1}{\epsilon} + A_0 \right) , \quad (1)$$

where R.S. defines the infrared regularisation scheme. The Laurent coefficients A_j are real-valued. The overall constant is given by

$$C(\epsilon) = \frac{(4\pi)^\epsilon}{\Gamma(1-\epsilon)} \left(\frac{\mu^2}{\mu_{\text{R}}^2} \right)^\epsilon = (4\pi)^\epsilon \frac{\Gamma(1+\epsilon)\Gamma(1-\epsilon)^2}{\Gamma(1-2\epsilon)} \left(\frac{\mu^2}{\mu_{\text{R}}^2} \right)^\epsilon . \quad (2)$$

3 New features of the interface

For the *initialisation phase*, we define a number of new keywords to allow for more options in the order/contract files. The valid keywords are listed in Appendix A.

There should be a check at the level of the order/contract files if all settings are fully supported. As the new standards are not backwards compatible, we propose to place the keyword `InterfaceVersion`, which can take the values `BLHA1` or `BLHA2`, in the order file.

Further, we define a new function

`OLP_Info(char[15] olp_name, char[15] olp_version, char[255] message)` to be called by the MC after `OLP_Start`. The function serves to keep track of the type and version of the OLP which has been used, and to encourage proper citation. The arguments are the name of the OLP, the version, and a string which contains information about the relevant publications, for example the bibtex identifier.

At *runtime*, two new functions, `OLP_SetParameter` and `OLP_PrintParameter`, described in more detail below, are defined.

The new function `OLP_EvalSubProcess` does not contain the passing of coupling constants anymore, as this is better done by `OLP_SetParameter` now. It also has a new argument appended which is useful for the treatment of unstable phase space points. With the new argument list, the function is not backwards compatible with the original standard. Therefore, to avoid confusion with different versions, the new function is called `OLP_EvalSubProcess2`.

```
OLP_EvalSubProcess2(int i,double* pp,double mu,double* rval,int* status)
```

The arguments are

- int `i`: one element array with the label of the subprocess
- `double* pp`: pointer to an array of momenta, conventions $(E_j, k_j^x, k_j^y, k_j^z, M_j)$ unchanged
- `double mu`: one element array with renormalisation scale
- `double* rval`: pointer to an array of return values
- `int* status`: pointer to a (one element) array denoting the status of the accuracy check.

Note: originally, the argument list of `OLP_EvalSubProcess` contained both μ_r and $\alpha_s(\mu_r)$: `OLP_EvalSubProcess(int i,double* pp,double mu,double alphas,double* rval)`.

However, $\alpha_s(\mu_r)$ can now be set using the new function `OLP_SetParameter` to pass also dynamical parameters. This setup seems also more convenient for mixed (e.g. QCD-EW) corrections or corrections where α_s at different scales should be used within the same calculation.

The length of the array `rval` is at least four, containing the Laurent coefficients A_2, A_1, A_0 and $|\text{Born}|^2$. However, for the case where colour/spin correlated matrix elements are returned, the array `rval` must be longer. Details about the labelling conventions for such cases are given in section 3.6.

The list of defined functions is given in Appendix B, where the C/C⁺⁺ version (with pointers in the argument list) is given. The C⁺⁺ version with function calls by reference as well as a Fortran (2003) module for binding with C/C⁺⁺ can be found at <http://phystev.in2p3.fr/wiki/2013:groups:sm:blha:api>.

3.1 Passing parameters

In the first version of the interface, the standard only allowed to pass a fixed amount of information at the level of the order/contract files. However, to be able to pass also dynamical parameters like running masses, and to have more flexibility in the definition of individual parameters, we suggest the following extension.

Parameters can be passed by the function

```
OLP_SetParameter(char* para, double* re, double* im, int* ierr)
```

where the first argument is a (pointer to a) string serving as a keyword for the parameter to be set, followed by two double precision numbers so that complex parameters can also be passed (in case of real parameters, the second double is zero). The integer in the fourth argument is set by the OLP to tell the MC whether the setting of the parameter was successful.

`ierr=1` means the parameter has been set successfully,
`ierr=0` means failure: issue an error message and stop,
`ierr=2` means that the parameter is unknown or the setting is ignored, but the program should proceed.

Apart from setting individual parameters separately, there should also be the possibility to define some global settings in the order file. For example, `Model: SMdiag` should set the CKM matrix to unity globally.

The parameters that need to be set within a certain model can be defined in two ways. Either they are imported in `UFO` (Universal FeynRules Output) [7]

format, which is particularly useful for calculations beyond the Standard Model, or they can be passed with naming conventions as specified in Table 1 for the Standard Model. The number in parenthesis behind “mass” and “width” denotes the particle’s PDG code.

The **UFO** format also provides the SLHA identifiers [8–10]. Ideally, the SLHA identifiers should also be recognized by both MC and OLP, which would facilitate changing parameter values at runtime, without having to re-generate the code.

keyword	parameter
mass(5)	b quark mass
mass(6)	top quark mass
width(6)	top quark width
sw2	$\sin^2 \theta_w$
vev	SM vacuum expectation value
Gf	G_{Fermi}
VV12	V_{ud}
:	

Table 1

List of keywords to define parameters to be passed by the function **OLP_SetParameter**.

The function **OLP_SetParameter** can be called at runtime, for every phase space point, if used to define a dynamic parameter. Obviously it can also be called once (for each particular keyword/parameter that needs to be passed) if this is a static parameter needed only at the start of the run-time phase.

Further, we propose a routine **OLP_PrintParameter(char* filename)** giving out a list of the actual (static) parameter settings used in the calculation, where **filename** is the name of the output file. This routine is optional.

3.2 Treatment of unstable phase space points

At the level of the order file, a new keyword **TreatUnstable** can be used to define globally how to deal with unstable phase space points. Possible values can be

TreatUnstable discard, novirt, debug

When the run-time phase is started, the function `OLP_SetParameter(...)` can be used to have a flexible way to decide if a point is classified as unstable. This can be done by a parameter specifying the precision threshold which divides the points into “stable” or “unstable”, called `Precision`.

Precision 1e-4

The precision threshold, i.e. the dividing line between the classification as “stable” or “unstable”, should be a relative accuracy determined as a result of internal stability checks. The OLP should be free to use its preferred stability criteria internally.

The function `OLP_EvalSubProcess2` can be used to pass information about the quality of the phase space point. In the original standard, the last argument of this function was the return value, i.e. an array of four real-valued double precision numbers, denoting `PoleCoeff2`, `PoleCoeff1`, `PoleCoeff0`, `BornSquare`.

We propose to append, after this array, a single integer `status` encoding the outcome of the stability test. The Monte Carlo program will then proceed according to the flag set in `TreatUnstable`. `status=1` means the point has passed the stability test(s), `status=0` means that the phase space point did not pass. If `status=0` and the `TreatUnstable` flag was `discard`, the point is discarded, while in the case of `novirt`, the value for the virtual part delivered by the OLP is set to zero, and the histogram is filled with a point which is approximately Born level. In the case of `debug`, the phase space point should be printed to a file to allow further diagnostics (but not filled into the histogram). The format of such a debug file should be defined internally within the OLP.

It could further be useful to pass more detailed information, e.g. how many digits the OLP claims to be correct in the finite part of the virtual corrections, as the relative accuracy required by `Precision` might not always have been reached.

For this and other purposes, it was agreed to define a flag `Extra` for OLP specific parameters. An example from NJET [3] looks like

<code>Extra NJetSwitchAcc [default = 1e-5]</code>

The flag sets the relative accuracy at which higher precision floating point arithmetics will be used to evaluate the phase space point if double precision has not produced the desired accuracy.

3.3 Different powers of coupling constants, merging different jet multiplicities

So far, the interface was taylored to NLO calculations for a fixed jet multiplicity, and focused on QCD corrections rather than electroweak corrections.

However, mixed QCD-EW corrections, or expansions in parameters other than α_s or α , require a more flexible scheme to define the desired orders in coupling constants.

Further, recent developments [11–15] propose a merging method for matched NLO predictions with varying jet multiplicity. In order to calculate merged samples, the Monte Carlo program needs to ask the OLP for one-loop matrix elements with different jet multiplicities and therefore different powers of the coupling constant.

These situations can be accounted for by allowing different settings for different subprocesses, see Fig. 4. A setting is valid until it is explicitly overwritten.

This setup can be used for merged samples as well as mixed QCD/EW corrections. It can also be used to pass additional information referring only to particular subprocesses, as indicated e.g. by `Extra AmplitudeType LeadingColour`. Certainly, it is the responsibility of the user to make sure that the approximations behind the keyword `LeadingColour` are the same in the OLP and the MC program. Therefore, it was also suggested that `LeadingColour` should be supplemented by `ColourExpansion`, where the power i in a $1/N^i$ expansion is specified.

3.4 Electroweak corrections

In the case of electroweak (EW) corrections, it is of particular importance to check the consistency of the parameters, for example the relation between M_Z , M_W and $\sin^2 \theta_w$. This can be done in the following way: The scheme is set in the order file by the keyword `EWScheme`, which can take the values `alphaGF` (also known as G_μ -scheme), `alpha0`, `alphaMZ`, `alphaRUN`, `alphaMSbar`, `UserDefined`. Then the parameters are set using `OLP_SetParameter`.

The OLP imports these parameters. The integer in the last argument only indicates if the import was successful or not. Consistency, according to the `EWScheme` defined in the order file, can only be checked after all parameters have been imported. Therefore, a second function was suggested to return information about the consistency of the parameters. This could be done for

```

AlphasPower 2
# process list 2j
1 1 -> 1 1
1 -1 -> 1 -1
1 -1 -> 2 -2
1 -1 -> 21 21
21 21 -> 21 21

AlphasPower 3
Extra AmplitudeType LeadingColour
# process list 3j
1 1 -> 21 1 1
1 -1 -> 21 1 -1
1 -1 -> 21 2 -2
1 -1 -> 21 21 21
21 21 -> 21 21 21

AlphasPower      4
AmplitudeType   Tree
# process list 4j
21 21 -> 21 21 21 21
1 -1 -> 21 21 21 21
1 -1 -> -2 2 -2 2
1 -1 -> -1 1 -1 1
21 21 -> 21 21 21 21

```

Fig. 4. Example of the part of an order file containing different settings for different sets of subprocesses.

each parameter individually, by a function

`OLP_GetParameter(char* para, double* re, double* im, bool),`

where `char` denotes the same keyword as in `OLP_SetParameter`. After the OLP has checked the dependencies, this function returns `false` in its last argument if the parameters are found not to be consistent within a given `EWScheme`. If the EW scheme is `UserDefined`, the fourth argument should by definition always return `true`. If the last argument has been set to `false` by the OLP, the program should stop.

The discussions in Les Houches 2013 however lead to the following: The EW scheme `UserDefined`, which leaves it to the OLP to ensure consistency, is the most likely one to be actually used by the OLPs. This makes the function `OLP_GetParameter` obsolete. Therefore the current agreement is to refrain from introducing a function `OLP_GetParameter`.

3.5 Extras

The keyword **Extra** can be used to write special requirements relevant to the OLP into the order file. Apart from writing them into the order file, the MC will ignore them (unless they are known and relevant to the MC).

For example, requirements concerning a colour expansion could be put under the **Extra** flag. The default is that amplitudes are summed over final state colours and polarisations and averaged over initial state colours and polarisations. It should be possible to change this by

```
Extra HelAvgInitial false
Extra ColAvgInitial false
Extra MCSymmetrizeFinal false
```

where `MCSymmetrizeFinal` refers to the symmetry factor introduced for identical particles in the final state.

Further, the keyword **Extra** is useful if the process is loop induced, i.e. if the leading order process already contains a loop. Therefore the keyword `LoopInduced` has been introduced. In such cases the Monte Carlo program could adjust the phase space integration to take into account the fact that a tree level matrix element does not exist. However, such a procedure may not be available in the Monte Carlo program. To indicate that this is not a required standard, the case is flagged as an “Extra”: `Extra AmplitudeType LoopInduced`.

The **Extra** flag can also be used to define that MC and OLP should use the MINLO [16] procedure, if the latter is available on both sides.

3.6 Colour and polarisation information

Going beyond the generic case of colour and helicity summed matrix elements it becomes difficult to satisfy special needs of different programs with one global standard. The options below can be considered as suggestions how to set up an interface, but the details probably remain to be implemented and tested individually between specific programs.

The MC can ask for colour- or spin-correlated matrix elements by writing one (or several) of the following keywords into the order file:

AmplitudeType ccTree	colour correlated tree amplitude
AmplitudeType ccLoop	colour correlated loop amplitude
AmplitudeType scTree	spin correlated tree amplitude
AmplitudeType scLoop	spin correlated loop amplitude

The defaults for `AmplitudeType` are `Loop` and `CHSummed`. If only `Tree` or `Loop` are specified, this automatically means `CHSummed`.

Typically, both tree and loop amplitudes will be needed if colour correlations are to be kept. The new setup to define individual settings for blocks of subprocesses can be used in this case, with the effect that certain subprocesses will be called twice, once for the tree case and once for the loop case. Example:

```

AmplitudeType ccTree
1 -1 -> 21 1 -1
21 21 -> 21 21 21

AmplitudeType ccLoop
1 -1 -> 21 1 -1
21 21 -> 21 21 21

```

If the amplitude returned by the OLP is not colour and helicity summed, the values to be returned by `OLP_EvalSubProcess2` form a matrix in colour and/or polarisation space. To pass the values in an unambiguous way, it is necessary to (a) define a basis, and (b) define the order how to write the matrix elements into the array `rval` returned by the OLP.

Colour

To define a basis in colour space, the colour flow decomposition [17] seems to be the most convenient. The matrix T_{ij} in colour space is symmetric, therefore it is sufficient to pass only T_{ij} for $j \geq i$. The most straightforward convention is to increment first j for fixed i , and then increment i , i.e. to pass the vector $T_{11}, T_{12}, \dots, T_{1n}, T_{22}, T_{23}, \dots, T_{2n}, \dots, T_{(n-1)n}, T_{nn}$.

Helicity

The first problem is again the question how to pass information about the basis in polarisation space used by the MC/OLP. For massless particles, the helicity basis is most convenient. For massive fermions, the light-like vectors which define the light-cone decomposition (see e.g. [18, 19]) should be communicated. For massive vector bosons, the situation is more tricky. The least ambiguous way seems to be the specification of the polarisation vectors. The consensus reached in the discussion allows for two options:

`PolvecsAll`: specify polarisation vectors for all particles

`PolvecsMVB`: specify polarisation vectors for (polarised) massive vector bosons

only, for the remaining particles specify reference vectors for helicity projection.

ToDo: The order in which the individual helicity amplitudes are written into the vector `rval` still needs to be defined!

Obviously, if one of the contractors cannot provide the detailed colour or helicity information requested, the calculation should exit at the stage of `OLP_Start`.

3.7 Restrictions such as diagram filters, exploitation of special symmetries, etc.

The keyword `ExcludedParticles` in the order file can be used to remove unwanted particles from the code generation. Restrictions like confining the set of diagrams to resonant diagrams only can be set in the MC input card. The one-loop provider should make sure that all settings are coherent with possible filters in the OLP program. It was found that anything more than that is too error prone. However, OLP specific restrictions can be imposed using the keyword `Extra`.

4 Conclusions

This writeup summarises the update of the standard interface between Monte Carlo programs and one-loop matrix element providers which has been initiated at the Les Houches 2009 workshop on Physics at TeV Colliders, called the “Binoth Les Houches Accord (BLHA)”. The setup meanwhile has been implemented by several groups and facilitates the automation of NLO calculations.

The experience gained meanwhile with the original setup fed into the discussion about an extension of the standards, such that it can be used in a wider and more flexible context. The outcome of the discussion between a large number of Monte Carlo and one-loop providers is summarized in the present document, which is intended to serve as a reference for the new standards. This should increase the flexibility of both Monte Carlo programs to import virtual corrections where available and of one-loop providers to team up with different Monte Carlo programs. This is an important step forward, as different MC’s and OLP’s have different focus and strengths concerning for example the multiplicity of final states, particle masses, electroweak corrections, BSM

capability, etc. We therefore hope that BLHA version 2 will contribute to the goal of having NLO as a standard for the comparison of LHC data to theory.

Acknowledgements

We would like to thank the CERN TH/LPCC Institute on SM at the LHC for hospitality and for providing a stimulating environment to discuss these matters. We would also like to thank the Les Houches 2013 organizers for providing a platform (e.g. Wikipages) and again a stimulating environment to work out this Accord. GH would like to thank everybody who contributed to the creation and implementation of the Accord, both BLHA1 and BLHA2.

A List of valid keywords for order/contract file

A.1 Required keywords

InterfaceVersion: Can take the values `BLHA1` or `BLHA2`. This clarifies already in the initialisation phase if the new standards are supported by both MC nad OLP.

Model: `SMdiag`, `SMnondiag`, `MSSM`.

For BSM standards, the `UFO` [7] is most convenient.

AlphasPower: integer which specifies the α_s power of the Born cross section. (Can also be used for sub-processes, power refers to cross section rather than amplitude level).

CorrectionType: the type of higher order correction should be specified. Standard values are `QCD` (α_s), `EW` (α).

IRregularisation: the infrared regularisation scheme used. Possible choices for QCD are `CDR`, `DRED`.

TreatUnstable: defines how to treat unstable phase space points. Possible values are `discard`, `novirt`, `debug`. If the function `OLP_EvalSubProcess2` returns `status=0` (phase space point did not pass) as a result of OLP internal stability checks, the MC will do the following:

discard: discard this phase space point

novirt: value for the virtual part is set to zero, histogram is filled with remaining value

debug: print the point to a file, do not fill it into histograms.

Precision: precision threshold which divides points into “stable” or “unstable”. Should be a relative accuracy, resulting from OLP internal stability checks.

A.2 Optional keywords

- AlphaPower:** integer which specifies the α power of the Born cross section.
The default is zero.
- AmplitudeType:** could take the values `ccLoop` (colour correlated loop), `scLoop` (spin correlated loop), `ccTree` (colour correlated tree), `scTree` (spin correlated tree), `LeadingColour`, `Tree`, `LoopInduced`. The default is `Loop`.
`Tree` and `Loop` without “`cc`” or “`sc`” imply `CHsummed`.
- MassiveParticles:** defines a list of massive particles at the level of the order file, for example `MassiveParticles 5 6`. The separator is a space. This also implies that the light quark masses are set to zero.
- LightMassiveParticles:** useful if mass regularisation instead of dim.reg. is used (EW), defines set of particles where only $\log(m)$ terms are kept, but not $\mathcal{O}(m)$ terms.
- ExcludedParticles:** can be used to exclude particles which are contained by default in `Model`. The particles should be listed behind the keyword, denoted by their PDG codes.
- MassiveParticleScheme:** a standard choice is `OnShell`.
- Extra:** This keyword can be used to write special requirements relevant to the OLP into the order file, for example `Extra AmplitudeType LoopInduced`. Apart from writing them into the order file, the MC will ignore them (unless they are known and relevant to the MC).
- SubdivideSubprocess:** this flag tells the user if a given process is represented in a split form to allow for multi-channel Monte Carlo sampling. May be useful to return colour/spin correlated matrix elements.
- EWScheme:** used schemes (discussed in the text) can be flagged by the keywords `alpha0`, `alphaMZ`, `alphaGF`, `alphaRUN`, `alphaMSbar`, `UserDefined` (default).
- WidthScheme:** defines the treatment of unstable particles. Standard values are `ComplexMass`, `FixedWidth`, `RunningWidth`, `PoleApprox`.
- PolvecsAll:** specify polarisation vectors for all particles.
- PolvecsMVB:** specify polarisation vectors for (polarised) massive vector bosons only, while for fermions specify reference vectors for light-cone decomposition.

A.3 List of keywords contained in the original proposal which probably will be retired

- MatrixElementSquareType:** Replaced by `AmplitudeType`, declared `CHsummed` as default. Originally this flag was intended to specify colour (C) and helicity (H) treatment. Possible values were defined as `CHsummed`, `Csummed`, `Hsummed`, `NOTsummed`.

The new default `CHsummed` also implies an average over initial state colours and polarizations. The flag `Extra` could be used to accommodate for the

keywords `HeAvgInitial`, `ColAvgInitial`, `MCSymmetrizeFinal`, which can be set to false if the factors included by default should be switched off.

ModelFile: the model file from which parameters have to be read. Keyword replaced by `Model`.

OperationMode: the operating mode of the OLP. This optional flag was intended to specify OLP defined conventions or approximations to the one-loop contribution, e.g. `CouplingsStrippedOff`, `LeadingColour`, `HighEnergyLimit`, etc.

The keyword `CouplingsStrippedOff` turned out to be ambiguous in the presence of EW couplings and therefore will not be used any longer.

ResonanceTreatment: has been replaced by `WidthScheme`, as, for example, the complex mass scheme also concerns non-resonant propagators.

B List of defined functions

We list here the definition of the functions needed for a working interface in the C/C++ version (with pointers in the argument list). A Fortran (2003) module for binding with C/C++ can be found at

<http://phystev.in2p3.fr/wiki/2013:groups:sm:blha:api>.

```
#ifndef __OLP_H__
#define __OLP_H__

#ifndef __cplusplus
extern "C" {
#endif

void OLP_Start(char* fname, int* ierr);
void OLP_Info(char[15] olp_name,char[15] olp_version,char[255] message);
void OLP_SetParameter(char* para, double* re, double* im, int* ierr);
void OLP_PrintParameter(char* filename); // (optional)
void OLP_EvalSubProcess2(int i,double* pp,double mu,double* rval,int* status);

#ifndef __cplusplus
}
#endif // __cplusplus
#endif // __OLP_H__
```

References

- [1] T. Binoth, F. Boudjema, G. Dissertori, A. Lazopoulos, A. Denner, et al. A Proposal for a standard interface between Monte Carlo tools and one-loop programs. *Comput.Phys.Commun.*, 181:1612–1622, 2010.
- [2] C.F. Berger, Z. Bern, Lance J. Dixon, Fernando Febres Cordero, D. Forde, et al. Precise Predictions for $W + 3$ Jet Production at Hadron Colliders. *Phys.Rev.Lett.*, 102:222001, 2009.
- [3] Simon Badger, Benedikt Biedermann, Peter Uwer, and Valery Yundin. Numerical evaluation of virtual corrections to multi-jet production in massless QCD. *Comput.Phys.Commun.*, 184:1981–1998, 2013.
- [4] Gavin Cullen, Nicolas Greiner, Gudrun Heinrich, Gionata Luisoni, Pierpaolo Mastrolia, et al. Automated One-Loop Calculations with GoSam. *Eur.Phys.J.*, C72:1889, 2012.
- [5] T. Gleisberg, Stefan Hoeche, F. Krauss, M. Schonherr, S. Schumann, et al. Event generation with SHERPA 1.1. *JHEP*, 0902:007, 2009.
- [6] Tanju Gleisberg and Frank Krauss. Automating dipole subtraction for QCD NLO calculations. *Eur.Phys.J.*, C53:501–523, 2008.
- [7] Celine Degrande, Claude Duhr, Benjamin Fuks, David Grellscheid, Olivier Mattelaer, et al. UFO - The Universal FeynRules Output. *Comput.Phys.Commun.*, 183:1201–1214, 2012.
- [8] Peter Z. Skands, B.C. Allanach, H. Baer, C. Balazs, G. Belanger, et al. SUSY Les Houches accord: Interfacing SUSY spectrum calculators, decay packages, and event generators. *JHEP*, 0407:036, 2004.
- [9] T. Hahn. SUSY Les Houches Accord 2 I/O made easy. *Comput.Phys.Commun.*, 180:1681–1693, 2009.
- [10] B.C. Allanach, C. Balazs, G. Belanger, M. Bernhardt, F. Boudjema, et al. SUSY Les Houches Accord 2. *Comput.Phys.Commun.*, 180:8–25, 2009.
- [11] Stefan Hoeche, Frank Krauss, Marek Schonherr, and Frank Siegert. QCD matrix elements + parton showers: The NLO case. *JHEP*, 1304:027, 2013.
- [12] Thomas Gehrmann, Stefan Hoche, Frank Krauss, Marek Schonherr, and Frank Siegert. NLO QCD matrix elements + parton showers in e^+e^- to hadrons. *JHEP*, 1301:144, 2013.
- [13] Rikkert Frederix and Stefano Frixione. Merging meets matching in MC@NLO. *JHEP*, 1212:061, 2012.
- [14] Keith Hamilton, Paolo Nason, Carlo Oleari, and Giulia Zanderighi. Merging H/W/Z + 0 and 1 jet at NLO with no merging scale: a path to parton shower + NNLO matching. *JHEP*, 1305:082, 2013.

- [15] Gionata Luisoni, Paolo Nason, Carlo Oleari, and Francesco Tramontano. Merging HW/HZ + 0 and 1 jet at NLO with no merging scale using the POWHEG BOX interfaced to GoSam. 2013.
- [16] Keith Hamilton, Paolo Nason, and Giulia Zanderighi. MINLO: Multi-Scale Improved NLO. *JHEP*, 1210:155, 2012.
- [17] Vittorio Del Duca, Lance J. Dixon, and Fabio Maltoni. New color decompositions for gauge amplitudes at tree and loop level. *Nucl.Phys.*, B571:51–70, 2000.
- [18] D. Maitre and P. Mastrolia. S@M, a Mathematica Implementation of the Spinor-Helicity Formalism. *Comput.Phys.Commun.*, 179:501–574, 2008.
- [19] Gavin Cullen, Maciej Koch-Janusz, and Thomas Reiter. Spinney: A Form Library for Helicity Spinors. *Comput.Phys.Commun.*, 182:2368–2387, 2011.