desing v1.5 manual

Gábor Bodnár and Josef Schicho
Research Institute for Symbolic Computation (RISC-Linz)
Johannes Kepler University, A-4040, Linz, Austria

November 25, 2004

Abstract

In this document we present the *desing* package, a Maple program for embedded desingularization of subschemes of smooth varieties over fields of zero characteristic. We give the specifications of the provided procedures and the used data structures, together with the special features of the package. The program comes in precompiled form with the full source code available under the GNU GPL. The newest version with online documentation is available at the page: http://www.risc.uni-linz.ac.at/projects/basic/adjoints/blowup. This work was supported by the Austrian Science Fund (FWF) in the frame of the projects: P15551 (formerly SFB F1303 and P12662).

1 Introduction

The problem of desingularization briefly is, given an algebraic variety X, find a proper, birational morphism $f : Y \to X$, such that Y is a nonsingular variety. Without going into the history deeply, we just mention that the existence of desingularization in arbitrary dimension, over fields of characteristic zero was first proved by Hironaka [Hir64]. His proof was non constructive, and it took till the end of the XX century to come up with constructive versions for hypersurfaces [Vil89, Vil96, EV00, EV98] and [BM91, BM97]. Here we do not intend to go into the details of the theory, we ask the reader to consult the papers in the reference list, in particular we recommend the papers [BS00a, BS00c, BS00b, BS01, Bod00] on our approach.

This software package is an implementation of Villamayor's algoroithm, having been optimized in several subparts. The program constructs embedded desingularizations for subschemes of smooth varieties in arbitrary dimension, over a computable algebraically closed, field of characteristic zero. The most essential operations required by the algorithm are performed with Gröbner basis computations [Buc65, Buc85, Win88, BW93]. The package can use either the Gröbner package Maple 8 or the Algolib package of F. Chyzak or the Gb package (which is an interface to the C++ implementation) of J. C. Faugere.

The *desing* package has several additional features implemented besides the main functionality. It can provide output for human experts to read, it can handle continuations of interrupted computations, etc.

From version 1.3 on, several additional routines are available to compute vector spaces of adjoints, compute the dual hypergraph of the resolution, unify the sequence of blowing ups of the resolution, etc.

To our best knowledge, there is no other existing computer program that performs desingularization in such a general case.

The software is under continuous development since 1998. It started in the Adjoints Project (FWF P12662) (1998-2000), and continued, supported by the project Proving and solving over the reals (FWF F1303) (2000-2002), and recently by the project xplicit Resolution and Related Methods in Algebraic Geometry and Number Theory (FWF P15551) (2002-).

Please send any feedback and bug report to adjoints@risc.uni-linz.ac.at.

2 User manual

In this section we give the information a user has to know to use the functionality of the package. The most important parts are the requirements and the description of the basic functionality. The HTML export, the continued computations and the distributed computations are also treated here. Finally, we give short examples how to start computations using different features of the package.

2.1 Version updates

Main features of the *desing* package in version:

1.0

- basic functionality (section 2.3),
- text output for debugging (section 2.3),
- HTML output (section 2.5),
- partial result saving (section 2.6),
- the CASA package is required.

1.1

- new architecture corresponding to the one described in [Bod00],
- speedup in the resolution (section 4.37),
- either CASA or the *new* Groebner and Ore_algebra subpackages of the Mgfun package (version 2.2 or newer) are required (section 2.2),
- extended chart description, which appears also in the HTML output (section 2.5): blowing up and blowing down maps are included.
- fixed bugs in the singular locus computation routine for weighted basic objects, and in the procedure responsible for the preparation of the dimension reduction.
- a generalized blowup routine is added (section 4.7).

1.2

- revised handling of combinatorial information in the basic objects,

- possibility to output the maximum value of Villamayor's stratifying function locally over the current chart in the chart tree of the HTML output,
- in version 1.2.1 the calling sequence of the mgrobner procedures of CASA is automatically adjusted according to the version of CASA.

1.3

- extension of the scope of the algorithm to the non-hypersurface case after [EV01] (principalization of ideals is also supported),
- simplified handling of combinatorial information in the basic objects,
- simplified tests for deciding the necessity of separating components of a reducible hypersurface of maximal contact,
- it is possible to use a single variable to denote the inverse of several functions in the cover operations (optional feature),
- a new approach to cover blowing up varieties, which allows us to keep the dimension of the ambient space constant (optional feature),
- explicit test for normal crossing divisors which simplifies Villamayor's stratifying function, yielding sometimes much simpler resolutions (optional feature),
- new way of configuring the package, via the **desingcfg** table,
- computation of adjoints for hypersurfaces via the resolution of their singularities,
- computation of the dual hypergraph of the resolution,
- unification of the sequence of blowing ups of a resolution,
- generate resolution of singularities of projective varieties via dehomogenizations in each principal open subset automatically,
- possibility for the user to direct the algorithm (i.e. to tell it in which chart of the covering of the blowing up it should continue the resolution, or to choose from some predefined tree traversal strategies),
- Maple V Release 4 is not supported anymore,
- in version 1.3.1 a bug is removed from the algorithm that stops the resolution at the right moment in the non-hypersurface case (we thank S. Encinas for his help).

1.4

- this version requires Maple 8,
- a resolution strategy for reducible algebraic sets is introduced (`dlocext` and `dlocsep` subroutines),
- the package can be used without any other packages now, but it can use the Groebner and Ore_algebra subpackages of the Algolib package, or the Gröbner basis implementation of J. C. Faugère with the Gb interface Maple package,
- several bugs have been removed.

1.5

- the speedup strategy of normal crossings test has been rewritten, the corresponding feature is turned on by default now,
- computation of global sections of ideal sheaves on the variety W' (in the computed resolution $W' \to W$) is introduced,
- computation of integral closure of idels via their principalization,
- a few minor bugs were fixed.

2.2 Requirements

The package *desing* v1.5 is written for Maple 8 or more recent versions. No other packages are necessary to run the software, however it is possible to use the following packages for the Gröbner basis computations:

- Algolibhttp://algo.inria.fr/libraries/ by Frédéric Chyzak,
- Gbhttp://fgbrs.lip6.fr/jcf/Software/Gb/index.html by Jean-Charles Faugère.

If the packages are properly installed (please do not forget to register for the Gb package), the user can switch between using any of them by setting the

```
desingcfg[gblib]
```

configuration variable to one of the following possibilities:

`""` use the Gröbner basis implementation that comes with Maple.

`"algolib"` use the Algolib package.

`"Gb"` use the Gb package (interface to the C++ implementation).

The distributed feature is not available as the distMaple package does not supprt Maple 8 yet.

2.3 Basic functionality

The program provides an embedded desingularization of a given affine scheme embedded into some regular variety in n-dimensional affine or projective space. This functionality can be reached in the simplest case by the procedure `dloc` for the resolution of the singularities of an affine scheme. It expects the following parameters:

poly a polynomial from k[vars],

vars a list $x_1, \ldots, x_n$ of variables.

Since the program has to generate new names for newly introduced variables, we have to be careful to avoid name collisions. Therefore, we require that the variables generated by the pattern x_i are reserved for the program. The input problem has to be expressed by the first n variables, so that the rest becomes available for the program.

The output of this function is a list of charts, which is the same as the value of the global variable **resolvedChartList**. These charts cover the last blowing up variety in the resolution process, over which the embedded scheme is resolved. The corresponding members of the record of a chart are fully described in section 3.1.

The whole tree of the resolution is kept in the global variable **chartHistory**. To not generate this history, for instance to save memory, one has to set the configuration variable **dropHistory** to true (by default it is false) before starting the computation.

There is an additional routine to set up customized weighted basic objects and charts for resolution. The procedure is `dlocBO` and its full description can be found in section 4.22.

Remark: We optimize, to a certain extent, the covering of the blowing up variety by dropping some unnecessary charts. The selection strategy relies on the **FOCUS** entries of the charts (for more information see [Bod00, BS01]). The **FOCUS** entry of the initial chart record is set to be the generator(s) of the defining ideal `J` of the input scheme. This means that we might need an additional chart to completely cover the resulting blowing up variety, which contains everything outside the zero set of `J`. This chart is not generated by the algorithm, because it is uninteresting; it contains no point of the input scheme.

If one wants to resolve a projective variety via the resolution of its restrictions to the principal affine subsets of the projective space, one has to use the procedure `dlocpdh`. Its resolution data will be collected in **chartHistorypdh** for the resolutions over the principal open subsets. The `adjointspdh` procedure is then capable of processing this as input, returning the adjoints for the projective hypersurface (see section 4.2).

If the input is an affine scheme of codimension > 1 and not only the resolution of the singularities, but also the principalization of the defining ideal has to be computed the procedure `dlocpr` is to be called.

For debugging purposes a configuration variable, `desingDebug`, was created (which is false by default). When this flag is set to true, the program writes the data of the currently processed chart to the standard output on every major step of the computation. This feature is to be used in batch mode, when the standard output is redirected to a file.

To compute the adjoints vector space of a hypersurface, first one has to compute its resolution such that **chartHistory** contains the full data of the resolution, and the value of **globalChartCounter** is the number of chart records in **chartHistory**. The full description of the `adjoints` procedure can be found in 4.1.

With the `dualgraph` procedure, one can compute the dual hypergraph of a given resolution, which is stored in **chartHistory**. The full description of the procedure can be found in section 4.27.

The `blunify` procedure computes the "unification" of the sequence of blowing ups of a resolution, stored in **chartHistory**. More precisely, it returns a set of generators for an ideal in the coordinate ring of the ambient variety of the input problem, such that the blowing up of that ideal results an isomorphic variety of the last blowing up variety of the resolution. Or in other terms, the ideal encodes the sequence of blowing ups made in the resolution of the input scheme.

2.4 An alternative way to resolve reducible algebraic sets

The function `desing/dlocext` can be called when there is already a completed resolution stored in `chartHistory`.

Input:

> **J** a polynomial from k[VAR], defining the scheme to be desingularized, where VAR is the list of variables of the root chart in the resolution tree stored in `chartHistory`.

Output: The value of the `resolvedChartList` global variable, containing the data of the chart covering the final blowing up variety.

The effect is that the strict transform of J is computed along the map of the resolution which can be found in `chartHistory` and new basic objects are set up over the leaf-charts of the tree. Then the objects are resolved, such that the existing resolution is "extended" with the resolution of the strict transform of J.

When more than one varieties were resolved using `desing/dlocext` the nonsingular strict transforms (identified by the TAGS) meet in normal crossings. To achieve an embedded resolution of the singularities of the union of all the so resolved varieties, we have to separate these transforms. This is done by `desing/dlocsep`. It works on the resolution stored in `chartHistory`, and takes no input parameters.

2.5 HTML output

An way for human experts to analyze the resolution is provided by the HTML exporter. This feature is controlled by the following configuration variables:

dumpHTML a flag that has to be set true to switch the export on,

htmlpath has to contain a name of an existing directory, in which the HTML files will be put,

htmlstyle an integer value: 0: In the chart tree, the local maximum of the Villamayor stratifying function over the given chart appears. 1: In the chart tree of the HTML output the standard identifiers appear for the charts.

Example:

```
desingcfg[dumpHTML] := true;
desingcfg[htmlpath] := '/tmp/html/';
desingcfg[htmlstyle] := 1;
```

When the HTML export is on, each node of the resolution tree appears as a separate file in the directory specified by the configuration variables. Unlike in case of `desingcfg[desingDebug] := true`, the chart records are printed only at the input of the three major procedures of the algorithm: *cover*, *exchange*, and *blowup*, plus at the resolved stage.

In the end of the computation when the tree is complete, an additional file is created which collects links to all the nodes in the tree: `tree.html`. In this table the blue fields indicate that cover operation took place on the colored node in the computation, green denotes exchange and red stands for blowing up.

The data contained by a node consists of the data of a chart and the data of all the basic objects associated to the chart at the time point of the export. The output contains the chart data:

```
VAR, DEP, IND, PDER, FOCUS, MAP, IMAP.
```

These are the corresponding entries of the chart descriptors, presented in section 3.1 (VAR stands for S).

The "stack" collects the basic objects of the chart. It contains weighted basic objects (WBO), tagged basic objects (TBO) and simple basic objects (SBO). Each TBO is created to have its singular locus equal to the weighted singular locus of the previous WBO. Each SBO has its singular locus restricted by the intersection of the maximal possible number of divisors from the `EM` component of the previous TBO. Each WBO is created to be one dimension less than the previous SBO (by appending an element of IND to N and computing the coefficient ideal).

For basic objects the following data are printed.
Weighted objects:

```
J, c, N, E, a, C
```

Tagged objects

```
J, N, E, EM, C
```

Simple objects:

```
J, N, H, E, C.
```

These data are the corresponding members of the descriptors of basic objects, presented in section 3.2.

On the page http://www.risc.uni-linz.ac.at/projects/basic/adjoints/blowup some examples of HTML outputs can be found.

2.6 Saving partial results

There is a possibility to set up automatic saving of the status of the work in a long computation. In case of a server breakdown or a period of intensive use of the system, the computations may be interrupted and continued later from the last saved stage.

This feature is controlled by the configuration variables:

saveWork a boolean flag, indicating whether the saving should be done,

saveTime a numeric value, providing the amount of time (in seconds) to be elapsed between two savings,

saveFile a name of a '.m' file, to which the data are saved.

Example:

```
desingcfg[saveWork] := true;
desingcfg[saveTime] := 1000;
desingcfg[saveFile] := '/tmp/html/savefile.m';
```

A computation can be continued by the `dcont` (or `pdcont`, in distributed environment) procedure, which takes only the name of the file from which the data can be retrieved. These backup files contain only the data that is essential for the computation to be carried on and give the whole final result (i.e. the charts that are resolved, the charts that need further processing and some administrative data for tracking interdependencies). This means that all the configuration information like, for instance, **htmlpath**, **saveFile** or the **clients** list in case of a distributed session has to be set by the user prior to the `dcont` or `pdcont` call. The single threaded and the distributed sessions can continue each other's interrupted work.

If an interrupted computation produced HTML export and the computation is not finished yet, the HTML file containing the resolution tree is not present. For the easier navigation, one can also generate such a tree-file for the partial results with the procedure `maketree`. It takes only one parameter, the name of the file in which the computation is saved.

2.7 Choosing the progress mode

Via the **progressmode** configuration variable one can set a method to choose the next chart record for resolution, from the pool of charts which contain unresolved basic objects. The variable is numeric, the default value is 0. Currently the following methods are selectable via their number:

0 depth first search (the pool of unresolved chart records acts as a stack),

1 breadth first search (the pool acts as a queue),

2 local maximum search: The major transformations (cover and blowup) produce from an input chart record one or more output chart records, which are put into the pool of unresolved charts. In this mode at each cover or blowup, the algorithm chooses from the newly produced chart records the one with maximal (partial) stratifying function value. (The maximum value of the function is completely computed for a chart record only when it contains a single resolved basic object, or when the chart is ready for blowing up.)

3 global maximum search: In this mode the algorithm chooses the chart record which has the maximal partial stratifying function value in the (entire) pool of unresolved charts.

4 local interactive mode: After each major operation, the produced charts are printed in detailed form, and the user can choose with which should the algorithm continue (by typing in the ID number of the chart). The user can also change the progress mode by typing a "−" before the given number.

5 global interactive mode: After each major operation, the list of chart IDs and corresponding (partial) stratifying function values in the pool of unresolved chart records are printed. The user can choose a chart record to continue with by typing in its ID, or can change the progress mode by typing a "−" before the given number.

Example:

```
desingcfg[progressmode] := 2;
```

2.8 Configuring special techniques

There are a few available special representation or computation techniques which are not used by default, however, in certain examples they might lead to the faster computation of the resolution.

2.8.1 FOCUS

By the configuration variable **usefocus** one can control if in each chart of the covering a list of equations, called FOCUS, is maintained to eliminate redundant charts. By default it is true, and one can deactivate the feature by setting

```
desingcfg[usefocus] := false;
```

2.8.2 Algebraic dependencies

The configuration variable **useDEP2** (default: false) controls the method which allocates new variables for inverses of functions from the coordinate ring of the chart.

If the variable is false, each invertible function gets its own variable, which denotes its inverse in the coordinate ring of the chart. This leads to the increase of variables, which is unfortunate from the point of view of Gröbner basis computations. On the other hand the polynomials with which we have to work are simpler than in the other case.

If the variable is true, there is a single variable allocated for the product of the invertible functions from the coordinate ring of the chart. In this case the number of variables can be kept constant as new invertible functions are introduced in the cover transformations. On the other hand the polynomials with which we have to work with can grow much larger and denser than in the other case.

2.8.3 Covering blowing up varieties

The configuration variable **skewcover** (default: false) controls the strategy by which the blowing up varieties are covered in blowup transformations.

If the variable is false, since the blowing up centers, locally over the charts, are always defined by regular parameters, the blowing up variety can be covered by affine charts in a straightforward manner. However, the dimension of the ambient affine space (i.e. the number of generators of the coordinate ring) of the new charts may in general increase.

If the variable is true, a sophisticated covering strategy is applied, in which the defining equations of the blowing up center are replaced by others, which allow us to define affine patches of the blowing up variety which can be embedded in affine spaces of the same dimension as the ambient affine space of the original chart. It can be proved that only finitely many charts are needed to completely cover the blowing up variety.

The configuration variable **rotationrange** (default: 10) also effects this feature, setting the upper bound for the coefficient of the polynomials used in the random rotations of the coordinate systems over the charts.

If **skewcover** is true, the program also sets **useDEP2** to true for the time of the computations.

This feature is of experimental nature and need not provide superior performance to the standard covering techniques. The `‘desing/adjoints‘`, `‘desing/dualgraph‘` and `‘desing/blunify‘` procedures do not work when the resoltuion is computed using this technique due to data incompatibility reasons. This covering strategy supresses the normal crossing test (section 2.8.4), a cooperative application of the two techniques might be implemented later.

The results related to this covering strategy are planned to be described in an upcoming paper.

2.8.4 Testing normal crossing

The configuration variable **usenctest** (default: false) controls the way a component of the Villamayor stratifying function is computed.

If the variable is false, the Villamayor function is computed as it is defined in the paper [EV00].

If the variable is true, the step when the maximum locus of the function is restricted to the intersection of the maximal number of exceptional divisors from E_{-} (see Definition 4.14 on page 160 of [EV00]) with certain prescribed properties, is computed differently. This time, instead of letting all the exceptional divisors which have the required properties to restrict the maximum locus, we select only those ones which violate the normal crossing property. This way, the dimension of the blowing up centers can be increased, and the resolution can often be simplified greatly.

The results are planned to be described in an upcoming paper.

2.9 Examples

In the following we give an example first for the sequential version, then for the distributed one with the `pdlocBO` procedure. A sequential computation can be started like:

```
with(desing);
desingcfg[htmlpath] := ‘/tmp/html/‘;
desingcfg[dumpHTML] := true;
desingcfg[usenctest] := true;
dloc([x3-x2^2+1, x1^2+x2^2+x3^2-1], [x1,x2,x3]);
dualgraph();
```

Which will result that the charts of the resolution tree (at operations cover, exchange, blowup) will be exported to HTML files to the directory `/tmp/html`. The normal crossing test will also be applied. All the other variables remain with their default values, which turns all other features off. Finally, after the resolution, the dual hypergraph is also computed, and, because the HTML output is enabled, an HTML export of the defining data of the graph will be generated in the file `/tmp/html/dualgraph.html`.

3 Data structures

In this section we present the complete data structures used by the program. We describe charts, basic objects, and global data structures which collect the results of the computations. The data of charts and basic objects are accessible with procedures whose names follow the same naming convention, see section 4. For the sake of completeness we also enumerate the configuration variables.

3.1 Charts

The primary data that describe a chart U are:

S a list of affine variables (generators of $k[S]$), (in the HTML output this field appears with the name VAR),

DEP a list of polynomials from $k[S]$ (the algebraic relations of S), $k[U] = k[S]/\langle DEP\rangle$,

DEP2 has role (and appears in the debugging and HTML outputs) only when the **useDEP2** configuration variable is true. It is the list of invertible functions over the chart.

IND a list of polynomials (representatives of elements from $k[U]$), that give rise to a system of regular parameters at every point of U,

PDER a matrix of polynomials from $k[S]$ (representatives of elements from $k[U]$), the partial derivatives of the generators of $k[S]$ w.r.t. the IND members,

MAP a list of polynomials from $k[S]$, representatives of the images of the initial generators: $x_1, \ldots, x_n$ in the current chart (blowing up map),

LMAP the list of images (polynomial representatives from k[S]) of the k-algebra generators of the parent chart in the resolution tree (local blowing up map),

IMAP a list of rational functions from $k(S')$, where S' is the S member of the initial chart; these are the images of the generators in S along the inverse of MAP (blowing down map).

TAGS in case of resolutions (and not ideal principalizations) after each completed resolution a new list of 0s and 1s is added to TAGS where the 1s mark those elements of `IND` that define the nonsingular strict transform over the chart. TAGS can contain more than one such list when subsequent components are resolved using `dlocext`. This data is used by `dlocsep`.

SCS a list of polynomials, generators of an ideal in $k[U]$, which is supposed to be the sections of an ideal sheaf over the variety for which U is a covering chart. This data is not used in the resolution. When a desingularization is completed, the SCS entries of the charts in the covering of the final blowing up variety can be set up by the user or another procedure and then the global section computation routines use these data.

The following data is necessary to administrate the evolution of the chart during the resolution process and to provide auxiliary information for optimization strategies:

Sn an integer, the next available index for the new variable x_{S_n},

DEP2V has role (and appears in the debugging and HTML outputs) only when the **useDEP2** configuration variable is true. It holds the single variable which denotes the inverse of the product of the functions in DEP2 (the variable is also a member of S).

FOCUS a list of polynomials from $k[S]$, defining the subvariety of the chart not being covered by charts with smaller identifiers,

SUBST a list of equations, the substitutions made so far in the chart with which the algorithm eliminated variables,

IDs is a list of data building up the identifier of the chart, and providing additional information on the relation to the parent and to the children of the chart in the resolution tree. Its data members are:

1. The index of the chart in **chartHistory**, i.e. the major identification number of the chart.
2. The identifier of the parent of the chart converted to a maple name (this is used in the HTML exporter to generate links to the parent).
3. The minor identification number of the chart (currently unused).
4. The transformation identifier: it encodes the transformation which was performed on the chart, or gives the state of the chart: 0 - the chart contains a single resolved basic object, 1 - the transformation on the chart was blowing up, 2 - the operation on the chart was exchange, 3 - the transformation of the chart was cover.
5. The maximum value of the stratifying function over the chart, defined by the current state of the resolution problem (encoded in the basic objects). The function value is represented by nested lists of numeric data, and it is generated for a chart record by the function `` `desing/vstfn` ``, described in section 4.89.
6. A list of triples that correspond to the children of the chart in the chart tree. The first member of a triple is the major identifier (i.e. index in **chartHistory**) of the child, the second is
   - in case of blowing up, the index of the equation in IND, which becomes the defining equation of the exceptional divisor over the child,
   - in case of exchange the index of the IND-element which gets exchanged,
   - in case of cover, the equation of the function which becomes invertible over the child.

   The third member of the triple is a list which

- contains pairs: (variable, index), encoding which new variable over the child corresponds to the proper transform of which defining equation (identified by the index to IND) of the blowing up center,
- is empty in case of exchange,
- contains pairs: (variable, equation), encoding which new variable over the child is the inverse of which equation.

The full ID of a chart is the name `IDs[1].IDs[3]`.

3.2 Basic objects

The basic objects are of three types: weighted (WBO), tagged (TBO) and simple (SBO). Any basic object is associated to a chart U.

A weighted basic object is described by the following data:

N a list of indices to IND that define the ambient algebraic set W of the basic object (in U),

J a list of pairs `[poly, exp]`, which are the generators for the generalized ideal of the basic object (`poly` is a representative of a $k[U]$ element, `exp` is a nonnegative rational number),

E the list of exceptional divisors, given as:

- a list of equations if the configuration variable **usenctest** or **skewcover** is true,
- a list of indices to the IND member of U otherwise.

C a list of lists of the same length as IND of the chart U of nonnegative rational numbers (the positive members indicate the history of J modificators).

c a rational number (the quotient of the order of the proper transform within the singular locus of the WBO and b, the required order),

a a list of nonnegative rational numbers with the same length as E (the weights of the exceptional divisors),

type a integer value, which is 0 in case of a WBO, 1 in case of a TBO, and 2 for an SBO.

A TBO has the same entries as a WBO except the entries: a, c; plus the entry:

EM the list of exceptional divisors of E_-, given as a sublist of E.

An SBO is described as a WBO except the entries: a, c; plus the entry:

H the exceptional divisors of the TBO of the same dimension, whose intersection restricts the blowing up center, given in the same way as E.

Additional data is stored in each basic object to avoid the computation of the singular locus whenever possible:

SET a list of polynomials from $k[U]$, the generators of the ideal of the singular locus of the basic object within W,

SL a list of polynomials from $k[S]$, the Gröbner basis of SET+H+N+DEP.

These data are used to hold the singular locus of the basic object with the corresponding Gröbner basis until we need to recompute them. It helps in situations like testing the emptiness of the singular locus, what we have to do several times in the algorithm. Whenever an operation induces a geometric change in the basic object or in the chart, we need to invalidate the stored data asking for recomputation at the point where we need them.

3.3 Chart-records and global variables

Basic objects are defined in context of charts, that is, there must always be a chart in which the basic object exists. By this correspondence the data of basic objects and charts that belong together are stored in a common data structure, which is called a chart-record (or by overloading the name, simply: chart).

A chart-record contains the data of exactly one chart, and a list of data of basic objects that belong to the chart. This list contains at least one WBO, and it works as a stack. The structure of this basic object stack comes from Lemma 6.12 in [EV00]. For any SBO in the stack we have that either it locally contains a codimension one subset, which will be the next blowing up center, or its singular locus is the same as the weighted singular locus of the WBO derived from it by coefficient object computation.

The nature of the computation requires to keep the chart-records in global variables. There are three main global variables containing charts with another one that store information about the tree structure of the charts. There are additional variables in connection with the special features of the package or for storing administrative data. The global variables are:

globalChartList The list chart-records which are not resolved yet.

resolvedChartList The list of resolved chart records (i.e. charts which contain a single resolved WBO).

chartHistory The array of all charts of the resolution (the whole resolution tree).

chartTree A list of records for the HTML exporter, with entries:

- the ID of the parent chart
- a list of pairs `[child_ID, info]` representing children of this chart, where `info` is the expression to be printed in the HTML table under the link to the corresponding child,
- an integer code of the operation that was performed on the parent chart (1–blowup, 2–exchange, 3–cover).

globalChartCounter An integer containing the next available index for a chart ID.

treeOut A list of node information in the resolution tree (used by the HTML exporter) with the structure:

- the ID of the chart of the node (a name),
- the number of leafs in the subtree of the resolution tree given by this node as its root (an integer),
- the depth of the node in the resolution tree (an integer),
- a boolean value which is true iff the node is a leaf,
- the code of the operation that was taken on the chart of the node (1–blowup, 2–exchange, 3–cover, 4–resolved).

desingcfg a table of configuration information for the package.

The **desingcfg** table contains the following configuration variables:

gblib string that indicates which Gröbner package is to be used: `""`: use the standard package in Maple, `"algolib"`: use the Algolib package, `"Gb"`: use the Gb package.

dropHistory (boolean) true - do not keep the chart history (i.e. keep only the leafs), false (default) - keep the whole chart tree,

desingDebug (boolean) true - print debug info; false (default) - print nothing,

dumpHTML (boolean) true - generate HTML output, false (default) - do not produce HTML (see also section 2.5),

htmlpath (string) the name of the directory (ending with a /) in which the HTML files should be created (default "", i.e. the current working directory), see also section 2.5,

htmlstyle (integer) 0 (default) - tree.html shows the maximal values of the stratifying function over the charts, 1 - tree.html shows the chart IDs (see also section 2.5),

saveWork (boolean) true - save the state of the computation periodically; false (default) - no saving (see also section 2.6),

saveTime (integer) the time interval (in seconds) between the saves (see also section 2.6),

saveFile (string) the name of the file in which the saving is done (see also section 2.6),

haltingmode (integer) a control parameter to stop the resolution in the non-hypersurface case: 0 - resolve the WBOs of largest dimension completely (i.e. achieve principalization of ideals), 1 - stop when the nonsingular blowing up center is the same as the proper part of the ideal of the WBO with largest dimension (resolution of singularities). This parameter is automatically set, by the procedures to be called for singularity resolution and principalization,

progressmode (integer) default: 0, see section 2.7,

useDEP2 (boolean) default: false, see section 2.8.2,

skewcover (boolean) default: false, see section 2.8.3,

rotationrange (integer) default: 10, see section 2.8.3,

usenctest (boolean) default: false, see section 2.8.4,

maxchartindex (integer) the size of the array allocated for chartHistory (default 65000).

The global variables **pglobalChartList, pglobalChartCounter, pchartHistory, pchartTree, presolvedChartList, pdesingDebug, pdumpHTML** has the same role as their counterparts, but they are used in the main thread in a distributed computation. The global variable **chartHistorypdh** stores the resolution data obtained over each principal open subset of the input projective chart in pairs (**chartHistory**, **globalChartCounter**).

The following global variables are used by certain procedures of the package to optimize performance (e.g. not to pass the same information to recursively called procedures over and over again):

desingmonseq used by `` `desing/monsequence` ``,

desingSingS, desingSingIND, desingSingPDER, desingSingSL used by `` `desing/singrc` `` and `` `desing/singrc2` ``,

4 Procedures

Here the procedures of the package are enumerated with brief descriptions. Most of them are useful only to perform a particular subtask in the algorithm, but there might be some which can also be useful by calling them from outside.

4.1 `` `desing/adjoints` ``

It is also an interface procedure under the short name `adjoints`. It computes the vector space of m-adjoints for a hypersurface, after its singularities has been resolved.

Input:

> **args(1)** a basis of the vector space V in the k-algebra of the ambient variety (i.e. chart) of the hypersurface in which the adjoints are to be computed, or the degree up to which the standard monomial basis is to be taken if the chart is an affine space,
>
> **args(2)** an integer $m \geq 1$ specifying the order of adjunction.

Output: A basis for the sub-vector space of m-adjoints in V with respect to the hypersurface whose desingularization is stored in the **chartHistory** global variable.

Before the call to `` `desing/adjoints` `` one has to resolve the singularities of the hypersurface in question via a `dloc`, `pdloc` call. Example:

```
dloc(x1^4+x2^4+x3^2*x4^2, [x1,x2,x3,x4]);
adjoints(1,1);
```

4.2 `desing/adjointspdh`

It is also an interface procedure under the short name `adjointspdh`. The input-output specification of the procedure is the same as the one of `desing/adjoints`. This procedure should be used to compute the adjoints if the singularities of a projective hypersurface were resolved by the `desing/dlocpdh` or `desing/pdlocpdh` routines. If the first argument is a positive integer, the procedure takes the standard basis of monomials of that degree in the graded coordinate ring of the ambient projective space.

4.3 `desing/adjointsr`

Recursive computation of adjoint conditions for a subtree of the resolution tree.

Input:

- **V** a basis of the vector space V in the k-algebra of the chart identified by IDP in which the adjoints are to be computed,
- **m** the order of adjunction (≥ 1),
- **IDP** the identifier of the chart in the chartHistory for which the computation is to be done.

Output: The set of constraints for being an m-adjoint in V with respect to the singularities which are resolved in the subtree with root IDP.

4.4 `desing/blcompc`

Recursive computation of the unified blowing up center in a subtree of the resolution tree.

Input: **ID** the identifier of the root of the subtree in **chartHistory** for which the computation is to be done.

Output: The ideal in the coordinate ring of the chart with identifier ID, whose blowing up is isomorphic to the sequence of blowing ups encoded in the subtree with root ID.

4.5 `desing/blowup`

Performing blowing up operation on a chart-record along a given center. The center is defined as the intersection of hypersurfaces defined by elements from `IND`. The procedure creates new chart-records whose charts cover the blown up variety and the basic objects are transformed properly along the stack.

Input:

- **O** a chart-record,
- **C** a list of pairwise different indices to the `IND` member of the chart of the chart-record,
- **gp** boolean, true if the transformation is a “goodpoint” blowing up, false in the generic case.

Output: a list of chart-records covering the blowing up variety.

The input chart is moved to **chartHistory** and the operation is registered in **chartTree**. The value of **globalChartCounter** is increased as needed.

4.6 ‘desing/blowup2‘

Performing blowing up operation on a chart-record along a given center. The center is defined either by giving indices to `IND` or by giving the defining equations of normal crossing hypersurfaces. The procedure creates new chart-records whose charts cover the blown up variety and the basic objects are transformed properly along the stack. The new charts are embedded into the affine ambient space of the input chart (i.e. the number of k-algebra generators remain the same).

Input:

- **O** a chart-record,
- **C** a list of pairwise different indices to the `IND` member of the chart of the chart-record or the generators of normal crossing hypersurfaces defining the blowing up center,
- **gp** boolean, true if the transformation is a “goodpoint” blowing up, false in the generic case.

Output: a list of chart-records covering the blowing up of the input chart along the canter C, such that all charts has ambient affine space with the same dimension as the input chart.

The input chart is moved to **chartHistory** and the operation is registered in **chartTree**. The value of **globalChartCounter** is increased as needed.

This variant of blowup is used when **skewcover** is true.

4.7 ‘desing/blowupJ‘

An additional blowing up procedure that is not used in the algorithm but might be useful for the user to compute blowing ups of ideals.

Input:

- **W** a list of polynomials from $k[S]$,
- **J** a list of representatives of elements from $k[S]/\langle W\rangle$,
- **S** a list of variables.

Output: a list of three sublists:

- a list of polynomials from $k[S][Y]$, generators for the blown up ideal (it is homogeneous in the y's),
- the list `S`,
- the list of new variables `Y`, of the pattern y_i.

The variables can be anything except t and the ones of the pattern y_i for all nonnegative i. The procedure selects the minimal initial generator set for `J`.

4.8 `desing/blunify`

This is also an interface procedure under the short name `blunify`. It computes a finite set of generators for an ideal whose blowing up is isomorphic to the sequence of blowing ups computed by the algorithm in the resolution tree (stored in **chartHistory**).

Input: –

Output: A list of elements from the coordinate ring of the input chart of the resolution stored in **chartHistory**.

To unify the sequence of blowing ups of a resolution made by a `desing/dlocpdh` or `desing/pdlocpdh` call over the ith principal open subset of P^3 ($i = 1, 2, 3, 4$), set

```
chartHistory := eval(chartHistorypdh[i][1]):
globalChartCounter := chartHistorypdh[i][2]:
```

before calling `desing/blunify`().

4.9 `desing/branchSize`

Calculation of the number of leafs in the given subtree of the resolution tree. The procedure is used by `desing/dumpTree` to adjust the sizes of the blocks of the table containing the resolution tree.

Input:

- **ID** a name, the ID of the root of the subtree in the resolution tree,
- **level** an integer, the depth of the node `ID` in the tree.

Output: An integer, the number of leafs in the subtree with root `ID`.

The procedure puts node information in the global variable `treeOut`.

4.10 `desing/ccvv`

Coordinate computation of a vector in a vector space with respect to a given basis.

Input:

- **V** a basis of a vector space in the k-algebra of the chart,
- **P** the polynomial whose coordinates in has to be determined with respect to V,
- **S** the list of variables of the chart (cf. `VAR`),
- **C** (output parameter) the coordinates of P in V.

Output: true if the given vector is in the vector space, false otherwise.

4.11 ‘desing/chtohtml‘

Writing the data of a chart into a file in HTML format. The name of the file is built up from the ID of the chart and the suffix ‘.html’. The files are put in the directory specified by the global variable `htmlpath`.

Input:

- **O** the chart-record to be exported to HTML,
- **ID** a name, the ID of the chart,
- **parent** a name, the ID of the parent of the chart,
- **C** a list of the format of the second sublist of an element of `chartTree`, the list of children of the chart,
- **Msg** a name, the message to be written above the links to the children.

Output: –

4.12 ‘desing/clean‘

Separating singular loci of different dimension of an SBO. The procedure checks whether the singular locus of the SBO on the top of the stack of a chart-record, w.r.t. a list hypersurfaces defined by elements from the `IND` member of the chart has components of different dimension. If so, it covers the chart by new charts in which these components are separated.

Input:

- **Chart** the chart-record to be checked,
- **z** a list of pairwise different indices to the `IND` member of the chart,
- **DC** the coefficient ideal of `J` of the top SBO w.r.t. `z`.

Output: A list of “clean” charts, with no singular locus components of different dimension w.r.t. `z`.

4.13 ‘desing/coeff‘

Computing the coefficient ideal of the ideal member of an SBO w.r.t. a list of elements from the `IND` member of the chart.

Input:

- **J** the corresponding member of an SBO,
- **z** a list of pairwise different indices to the `IND` member of the chart of the SBO,
- **PDER, IND, DEP** the corresponding members of the chart of the SBO,
- **N** the corresponding member of the SBO,

Output: The coefficient ideal of `J` w.r.t. `z`, in the format of the J component of a basic object.

4.14 `desing/coeffC`

Computing the coefficient ideal of the `C` member of an SBO w.r.t. a list of elements from the `IND` member of the chart.

Input:

- **C** the corresponding member of an SBO,
- **z** a list of pairwise different indices to the `IND` member of the chart of the SBO,

Output: The coefficient ideal of `C` w.r.t. `z`, in the format of the `C` component of a basic object.

The procedure updates the rational values of the functions in `C`.

4.15 `desing/colonideal`

Computing the ideal quotient of two ideals in a given ring.

Input:

- **f** a list of polynomials from $k[S]$, the generators for the ideal "in the numerator",
- **g** a list of polynomials from $k[S]$, the generators for the ideal "in the denominator",
- **S** a list of variables,
- **DEP** a list of polynomials from $k[S]$, the generators for the factoring ideal ($k[U] = k[S]/\langle \mathrm{DEP} \rangle$).

Output: A list of polynomials from $k[S]$, the generators of the quotient ideal.

The polynomials of the parameters `f`, `g` are understood as representatives for elements from $k[U]$. The parameters `S` and `DEP` are usually the corresponding members of a chart.

4.16 `desing/compcoeff`

Computing the coefficient of a generator in a representation of an element of the ideal in terms of the given generators.

Input:

- **F** a list of polynomials from $k[S]$,
- **p, g** a polynomial from $k[S]$,
- **S** a list of variables.

Output: A polynomial c from $k[S]$, where $g = pc + F_1 c_1 + \ldots + F_{|F|} c_{|F|}$.

This procedure requires the new revision of the Gröbner package of Frederic Chyzak (not the one being distributed with Maple V Release 5).

4.17 `‘desing/completecover‘`

This procedure tests whether the blowing up variety is completely covered by the charts generated in `‘desing/blowup2‘`, or further charts are necessary.

Input:

- **F** the unrotated equations of the images of the exceptional divisors for the charts computed so far,
- **FC** the unrotated equations of the inverted functions for each blowing ups made so far,
- **CN** the equations of the blowing up center,
- **DEP2** the DEP2 entry of the original chart (whose blowing up is being computed),
- **S** the variables of the original chart, without `DEP2V`,
- **m** the codimension of the center in the original chart.

Output: boolean: true if the blowing up variety is completely covered by the charts generated so far, false otherwise.

4.18 `‘desing/cover‘`

Performing cover operation on a chart-record.

Input:

- **O** a chart-record,
- **CP** a list of polynomials from $k[S]$, representatives for elements from $k[U]$, where `S` comes from the chart, and $k[U]$ is the coordinate ring of the chart.

Output: A list of chart-records.

The elements defined by the parameter `CP` has to generate the ideal $\langle 1\rangle$ in $k[U]$.

The procedure selects the minimal initial generator set for the ideal of the smallest possible degrees. Then creates a new chart for each remaining generator, in which the zero set of the generator is not part of the chart.

The procedure adopts to the chosen representation of invertible functions (determined by **useDEP2**).

The input chart is moved to `chartHistory` and the operation is registered in `chartTree`. The value of `globalChartCounter` is increased as needed.

4.19 `‘desing/coverx‘`

Performing a cover operation then exchange operations in the new charts.

Input:

- **O, CP** the same as of `‘design/cover‘`,
- **EXP** a list of polynomials from $k[S]$, representatives for the elements to be put into the `IND` member of the charts,

POSP a list of indices to IND, the positions where the exchange has to be performed,

Z output parameter, the list of singletons containing the indices in the new charts that were eventually exchanged.

Output: A list of chart-records, with the indicated exchanges made.

The procedure functions like `‘design/cover‘`, but it performs an additional exchange operation on the new charts. The `Z` parameter is necessary because the generators of `CP` might be rearranged and some of them might be dropped, and we need to know which polynomials were exchanged in the output charts.

The procedure adopts to the chosen representation of invertible functions (determined by **useDEP2**).

4.20 `‘desing/dcontinue‘`

It is also an interface procedure under the short name `dcont`. The procedure continues a computation from a saved state.

Input: **df** The specification of the ‘.m’ file from which the saved computation has to be retrieved.

Output: The same as of `‘desing/dloc‘`.

4.21 `‘desing/dloc‘`

It is also an interface procedure under the short name `dloc`. It performs the desingularization of an affine scheme. That is, the algorithm considers the situation in a chart of `globalChartList`, and performs an operation on this chart independently from the others.

Input:

J a polynomial from k[VAR], defining the scheme to be desingularized,

VAR a list of variables $x_1, \ldots, x_n$.

Output: The value of the `resolvedChartList` global variable, containing the data of the chart covering the final blowing up variety.

In order to avoid name collisions the caveat of section 2.3 has to be held.

4.22 `‘desing/dlocBO‘`

It is also an interface procedure under the short name `dlocBO`. It has the same functionality as of `‘desing/dloc‘`, but it starts the resolution from a user given basic object and chart. Its parameter list consists of equations to indicate which members of the chart-record are set.

Input: *Required parameters:*

J=‘polylist’ a list polynomials in the variables of VAR, generating the ideal J.

VAR=‘namelist’ a list of affine variables (by convention $x_1, \ldots, x_n$).

Optional parameters:

DEP='polylist' a list of polynomials from k[VAR], the algebraic relations of VAR. The variety $\mathbf{Z}$(DEP) must be nonsingular (by default DEP=[]).

IND='polylist' a list of polynomials from k[VAR], representatives of equivalence classes of $k[\text{VAR}]/\langle\text{DEP}\rangle$, which must give rise to a system of regular parameters in each point of $\mathbf{Z}$(DEP) by simple translation (by default IND=VAR considered as polynomials).

PDER='polylists' a list of lists of polynomials from k[VAR], representatives of equivalence classes of $k[\text{VAR}]/\langle\text{DEP}\rangle$. PDER stands for an $n \times |\text{IND}|$ matrix, which in the ij-th position contains the partial derivative of the i-th affine variable w.r.t. the j-th parameter. If IND is given, PDER must be specified (otherwise, by default PDER is the unit $n \times n$ matrix).

FOCUS='polylist' a list of polynomials from k[VAR], generators for the subvariety of the chart which is not covered by other charts.

b='rlist' a list of positive rational numbers, the weights of the generators of J. The length of the list must be equal to the number of generators. (by default `b=[seq(1, i=1..nops(J))]`).

E='ilist' a list of pairwise different indices to IND, the list of exceptional divisors. (by default E=[])

a='rlist' a list of non negative rational numbers, the weights of the exceptional divisors. If E is given 'a' must also be specified, and the length of the list must be equal to the number of exceptional divisors, (otherwise, by default a=[]).

C='rlists' lists of the same cardinality as of IND, containing nonnegative rational numbers. They encode generators of the generalized ideal of the object that are power products of IND members, storing in the i-th position the exponent of the i-th IND member, divided by the rational component of the generalized ideal generator being encoded.

Output: The same as of `'desing/dloc'`.

The parameters mean the corresponding data of a basic object and a chart.

4.23 `'desing/dlocext'`

This function can be called when there is already a completed resolution stored in `chartHistory`.

Input:

J a polynomial from k[VAR], defining the scheme to be desingularized, where VAR is the list of variables of the root chart in the resolution tree stored in `chartHistory`.

Output: The value of the `resolvedChartList` global variable, containing the data of the chart covering the final blowing up variety.

The effect is that the strict transform of J is computed along the map of the resolution which can be found in `chartHistory` and new basic objects are set up over the leaf-charts of the tree. Then the objects are resolved, such that the existing resolution is "extended" with the resolution of the strict transform of J.

4.24 `‘desing/dlocpdh‘`

Desingularization of a projective variety, defined by a homogeneous equation, via the resolution of the singularities of its affine patches, obtained by taking intersections with the principal open subsets of the ambient projective space.

The input-output specification is the same as of `‘desing/dloc‘`. The resolution trees are stored in the global variable **chartHistorypdh**.

4.25 `‘desing/dlocpr‘`

Principalization of the ideal of an affine scheme in the input chart. In this case the algorithm does not stop when the proper transform of the input scheme becomes nonsingular and has normal crossing with the exceptional divisors, but performs one additional blowing up which makes the total transform of the defining ideal principal.

The input-output specification is the same as of `‘desing/dloc‘`.

4.26 `‘desing/dlocsep‘`

When more than one varieties were resolved using `‘desing/dlocext‘` the nonsingular strict transforms (identified by the TAGS) meet in normal crossings. To achieve an embedded resolution of the singularities of the union of all the so resolved varieties, we have to separate these transforms. This is done by `‘desing/dlocsep‘`. It works on the resolution stored in `chartHistory`, and takes no input parameters.

4.27 `‘desing/dualgraph‘`

It is also an interface procedure under the short name `dualgraph`. It computes the dual hypergraph of the normal crossing total transform obtained by the resolution algorithm. In this hypergraph a node is an exceptional divisor, and there is a hyper-edge between a given subset of nodes if the corresponding exceptional divisors have a common point. The procedure takes the resolution information from the global variables **chartHistory** and **globalChartCounter**.

Input: –

Output: A pair (A, M) where A is the list of vertices and M is the list of hyperbonds. A vertex in A is a pair where the first element is a list of pairs (ID, EQ) where ID is the identifier of the chart in which the exceptional divisor of the vertex appears given by the equation EQ and the second element of the vertex is the stratifying function value.

There is the question of what is to be considered as 'one blowing up' (or respectively 'one exceptional divisor'). We use the following conventions:

- Decomposing the (complete, normal crossing) exceptional divisor into irreducible components would be too costly; therefore we do not do it.
- Because the blowing up centers are computed according to Villamayor's stratifying function it seems to be the most reasonable decision to take those blowing ups (exceptional divisors) as one which are defined by blowing up centers along which the stratifying function attains the same value.
- The indices of exceptional divisors in the stratifying function value for a monomial blowing up depend on the chart. Thus we replace local indices by global ones to make the references clean.

If the **dumpHTML** configuration variable is true the data of the hypergraph is exported into the `dualgraph.html` file in the same directory as the other files.

To compute the dual graph for the output of a `‘desing/dlocpdh‘` or `‘desing/pdlocpdh‘` call over the ith principal open subset of P^3 $(i = 1, 2, 3, 4)$, set

```
chartHistory := eval(chartHistorypdh[i][1]):
globalChartCounter := chartHistorypdh[i][2]:
```

before calling `‘desing/dualgraph‘`().

4.28 `‘desing/dumpTree‘`

It generates the HTML code for a resolution tree, and writes it to the file `tree.html` in the directory given by `htmlpath`.

Input: –

Output: –

The procedure takes its data from the global variables `chartTree` and `htmlpath`.

4.29 `‘desing/exchange‘`

Performing exchange operation on a chart-record with chart U.

Input:

O a chart-record,

f a polynomial form $k[S]$ (a representative of an element of $k[U]$),

p an index to the `IND` member of the chart,

a a polynomial from $k[S]$, the inverse of $\partial_{\mathrm{IND}_p} f$.

Output: a new chart-record with `IND[p]` exchanged to `f`.

The input chart is moved to `chartHistory` and the operation is registered in `chartTree`. The value of `globalChartCounter` is increased by one.

4.30 `‘desing/exdsh‘`

Computes the sections of an ideal sheaf supported on the excptional locus.

Input:

- **LSC** a list of pairs (V, n), where V is a string which identifies the blowing up center that gave rise to the exceptional component in question using the Villamayor function value that characterizes the center in the resolution process. The natural number n gives the exponent of the factor of the sheaf supported on the selected exceptional divisor. By default the exponent is 0.

Output: -

The procedure assumes that `chartHistory` contains the chart tree of a computed seqence of blowing ups and puts the sections of the computed ideal sheaf into the SCS entries of the charts of the covering of the final blowing up variety.

4.31 `‘desing/exdshrc‘`

An auxiliary procedure for `‘desing/exdsh‘`, tracing the weak transform of exceptional components from the blowing up centers from which they arise to the charts of the covering of the final blowing up variety.

Input:

- **chi** the ID of the chart to be processed,
- **ei** the index (in IND) of the defining equation of the exceptional component in question,
- **ex** the exponent of the exceptional component in question.

Output: -

The procedure assumes that `chartHistory` contains the chart tree of a computed seqence of blowing ups and puts the sections of the computed ideal sheaf into the SCS entries of the charts of the covering of the final blowing up variety.

4.32 `‘desing/expseqggs‘`

Checks if an ideal supported on the exceptional locus of the sequence of blowing ups stored in `chartHistory` with a given exponent sequence for the components identified by Villamayor function values is generated by global sections.

Input:

- **LSC** a list of pairs (V, n), where V is a string which identifies the blowing up center that gave rise to the exceptional component in question using the Villamayor function value that characterizes the center in the resolution process. The natural number n gives the exponent of the factor of the sheaf supported on the selected exceptional divisor. By default the exponent is 0.

Output: True if the global sections generate the ideal sheaf, false otherwise.

The procedure assumes that `chartHistory` contains the chart tree of a computed seqence of blowing ups.

4.33 `desing/findt`

Computation of the Γ function defined in section 5 page 165-166 in [EV00].

Input: **C** the weight (or exponent) list of the exceptional divisors in the examined WBO (the `a` member of the WBO).

Output: A list of indices to `E` (or equivalently to `a`) that are chosen by the Γ function.

The `E` list of a chart is by no means a global list of exceptional divisors. Although the chronological order is correct, some exceptional divisors, which are outside the chart, do not appear. Thus the indices, and consequently the locally computed stratifying function value is not sufficient to detect patching of centers along the chart covering. When this is needed in `desing/dualgraph`, we reconstruct the global indices for the exceptional divisors in all the charts and replace the local ones with them.

4.34 `desing/findt2`

Computation of the Γ function defined in section 5 page 165-166 in [EV00]. In this case the list of exceptional divisors are not specified by indices to the `IND` entry of the chart, but they are given by equations; therefore an arbitrary pair need not have a common point.

Input:

- **a** the list of weights of the exceptional divisors,
- **E** the list of defining equations of the exceptional divisors,
- **S** the list of variables of the chart
- **DEP** the list of algebraic dependencies of S.

Output: A list of indices to `E` that are chosen by the Γ function.

The same remarks apply as for `desing/findt`.

4.35 `desing/gb`

This is a procedure performs Gröbner basis computation using the package selected by `desingcfg[gblib]`.

Input:

- **F** a list of polynomials in the variables [op(B1),op(B2)],
- **B1** the first block of variables of the ordering
- **B2** the second block of variables of the ordering

At least one of the blocks must contain a variable and then the defined ordering is the block degree-reverse-lexicographic one with B1-variables being bigger than B2-ones.

Output: A list of polynomials forming a Gröbner basis for the input with respect to the given ordering.

4.36 `desing/genbygsec`

The procedure checks if an ideal sheaf on the final blowing up variety of the blowing up sequence stored in `chartHistory` is generated by the global sections in the SCS entry of `chartHistory[0]`.

Input: -

Output: True if the global sections generate the sheaf and false otherwise.

4.37 `desing/goodpoints`

Performing monomial blowing ups to resolve a basic object when the condition of the speedup criterion is fulfilled.

Input: **Chart** a chart record with a WBO on top of its stack which fulfills the speedup criterion.

Output: –

The procedure performs the blowing ups that make the WBO resolved and puts the resulting chart-records into `globalChartList`.

4.38 `desing/gsec`

Compute the global sections of an ideal sheaf I on W' (in the resolution $W' \to W$). The sections of I should be put into the SCS entries of the charts in the `chartHistory` global array which form the covering of W' (the final blowig up variety in a resoltuion computed by *desing*).

Input: –

Output: The list of generators of the ideal of global sections on W.

4.39 `desing/gsecrc`

Compute the global sections of an ideal sheaf I on W' (in the resolution $W' \to W$). The sections of I should be put into the SCS entries of the charts in the `chartHistory` global array which form the covering of W' (the final blowig up variety in a resoltuion computed by *desing*). The difference to `desing/gsec` is that this procedure does not push the I directly out onto W (via the MAP entries of the charts), but it traverses the resolution tree recursively and collecting the data from the charts accumulatively (using the LMAP entries).

Input:

- **n** the ID of the chart over which the global sections of I should be computed from the data coming from the subtree of the resolution tree whose root is chart with ID: n.

Output: The list of generators of the ideal of global sections on the chart with ID: n.

4.40 `desing/gsecexd`

Compute global sections of an ideal sheaf with support on the exceptional divisors on W' (where $W' \to W$ is the last computed resolution) specifying a power for each exceptional divisor (identified by values of the Villamayor function).

Input:

LSC A list of pairs (VFN, n), where VFN is a value of the Villamayor function which was attained on the blowing up center that defined the blowing up in which the exceptional divisor in question was created, and n is the power to which the corresponding factor of the ideal sheaf should be risen.

Output: The list of generators of the ideal of global sections on W.

4.41 `desing/haltingmode1`

When the **haltingmode** configuration variable is 1, the resolution of singularities must stop when the next computed blowing up center is the same as the proper part of the ideal of the WBO with largest dimension. The procedure performs this test.

Input:

O the chart which is to be blown up (i.e. a full chart record),

z the defining equation of the blowing up center (indices to the `IND` entry of O).

Output: (boolean) true if the proper part (J, C) of the WBO of largest dimension is contains (or its zero set is contained in the zero set of) the blowing up center given by z, false otherwise.

4.42 `desing/idealproduct`

Computing a set of generators for the product ideal from the generators of the factors.

Input:

I1,I2 list of polynomials

Output: a list of generators for (I1)*(I2).

4.43 `desing/idealpower`

Computing a set of generators for the power of an ideal.

Input:

I a list of polynomials (set of generators)

n a natural number

Output: a list of generators for (I1)^n.

4.44 `desing/intersect`

Computing a set of generators for the intersection of a list of ideals in a given list of indeterminates.

Input:

> **S** the list of indeterminates
>
> **I1,I2,...** lists of generators of ideals

Output: a list of generators for (I1) intersection (I2) intersection ...

4.45 `desing/inverse`

Computing the inverse of an invertible element in a ring.

Input:

> **f** a polynomial from $k[S]$ (a representative of an element of $k[S]/\langle DEP\rangle$),
>
> **DEP** a list of polynomials from $k[S]$,
>
> **S** a list of variables.

Output: a polynomial from $k[S]$, the inverse of `f` in $k[S]/\langle DEP\rangle$.

4.46 `desing/iscontained`

Testing whether a given ideal is contained in another one.

Input:

> **A, B** a list of polynomials from $k[S]$ (representatives of the generators of an ideal in $k[S]/\langle DEP\rangle$),
>
> **DEP** a list of polynomials from $k[S]$,
>
> **S** a list of variables.

Output: true iff $\langle A\rangle \subseteq \langle B\rangle$ in $k[S]/\langle DEP\rangle$.

4.47 `desing/isempty`

Testing whether a list of polynomials generate the ideal $\langle 1\rangle$.

Input:

> **F** a list of polynomials from $k[S]$ (representatives of the generators of an ideal in $k[S]/\langle DEP\rangle$),
>
> **DEP** a list of polynomials from $k[S]$,
>
> **S** a list of variables.

Output: true iff $1 \in \langle F\rangle$.

4.48 `desing/isemptyS`

A compromising "speedup" solution for testing whether a list of polynomials has nonempty common zero set.

Input: the same as of `desing/isempty`.

Output: false if the common zero set is not empty, true if the complete emptiness test is required.

The results are not too convincing when this procedure is in use, so it might get unused in the package.

4.49 `desing/isnc`

Testing whether a given list of hypersurfaces over some affine chart has normal crossing.

Input:

F a list of coordinate ring elements, defining the hypersurfaces to be tested,

S the list of variables of the chart,

DEP the list of dependencies of S,

IND the regular parameters of the chart,

PDER the differentiation matrix over the chart,

args(6) (optional) anything, e.g. the boolean value: true.

Output: (boolean) true if the hypersurfaces defined by F have normal crossing, false otherwise.

If the last argument is given, the equations in in F without the last one are assumed to define hypersurfaces with normal crossing, and only the last equation is checked against the previous ones.

4.50 `desing/isnc2`

The functionality is similar to the one of `desing/isnc`, but the returning value provides more information.

Input: The same as of `desing/isnc`, except that there is no optional sixth argument.

Output: A list of indices to F, the maximal cardinality lexicographically minimal list of divisors which are not normal crossing with the hypersurface defined by the last equation in F.

In this procedure it is assumed that the hypersurfaces defined by equations in F without the last equation have normal crossing, and only the last equation is tested against the previous ones.

4.51 `desing/maxnnc`

Computation of a subset of exceptional divisors from the EM entry of a tagged object that pose problems in the transversality problem in the resolution and which is canonical in a certain sense (see the paper with working title “Explicit Normal Crossings Tests for Faster Desingularization”).

Input: **H** the list of c-1'st partial derivatives,

- **EM** the list of indices to IND (defining the suspicious exceptional divisors),
- **DI** the list of indices to IND defining those IND elements with respect to which partial differentiation can be performed,
- **SL** generators of the ideal of the singular locus (including DEP),
- **S** the list of indeterminates,
- **IND** the regular system of parameters over the chart,
- **PDER** the PDER matrix for IND.

Output: A list of indices to EM, selecting those exceptional divisors that have no normal crossings with a hypersurface of maximal contact in H and a in a certain sense canonical.

4.52 `desing/maketree`

It is also an interface procedure under the short name `maketree`. It retrieves the resolution tree from a saved computation and exports it in HTML format into the file `tree.html` in the directory given by `htmlpath`. This is a wrapper for the procedure `desing/dumpTree` specializing it to this task.

Input: The same as of `desing/dcontinue`.

Output: –

Since the `tree.html` file is by default saved only at the end of the computation, to browse the HTML pages of the charts of an unfinished computation, one has to generate the tree manually with this procedure.

4.53 `desing/monsequence`

Computation of the list of monomials up to a given degree in a recursive fashion.

Input:

- **S** the list of variables to build monomials
- **mon** the monomial which is computed in the previous recursive call (it should be set to 1 at the initial call),
- **d** the degree up to which generate monomials.

Output: -

The procedure uses the global variable, **desingmonseq**, which has to be set to the empty list before calling the the procedure. The resulting monomial sequence is obtained in the **desingmonseq** variable.

4.54 ‘desing/next_comb‘

Enumeration of all the j-element sublists of a list of given number of elements.

Input: **L** a list of flags (0,1), providing a characteristic list for the list whose sublists has to be generated.

Output: a list of flags, the next configuration in the enumeration. If there is no more sublists to be generated, it returns the empty list.

The procedure first has to be called with `L` containing first j 1's then padding 0's. Afterwards it has to be called with the previous configuration it gave.

4.55 ‘desing/ord‘

Computation of the order update for a generalized ideal of the structure presented in section 3.2.

Input: **J, N, S, IND, DEP, PDER** the corresponding members of a basic object and its chart (see section 3.2, and 3.1).

Output: a rational number, the order update of the exponent components of the generalized ideal `J`.

4.56 ‘desing/pder‘

Performing partial differentiation w.r.t. the parameter system of a chart.

Input:

- **S** a list of variables,
- **f** a polynomial from $k[S]$,
- **p** an index to the `IND` member of the chart,
- **PDER** a list of lists of polynomials (the corresponding member of the chart).

Output: the partial derivative of `f` w.r.t. `IND[p]`.

4.57 ‘desing/prepareSL‘

Update the `SET` and `SL` members of the top basic object in the stack of a chart-record.

Input: **Chart** a chart record.

Output: the same chart-record with the update done.

The data entries `SET`, `SL` are considered to be invalid, if they are empty.

4.58 ‘desing/printchlist‘

Printing the data of a chart-record to the standard output in a somewhat formatted way.

Input: **Chart** a chart record.

Output: –

4.59 `‘desing/pupdate‘`

During distributed computation the scheduled jobs finish in an unknown order; therefore the ID allocation has to be corrected for every chart returned from a client machine. This correction is done by this procedure making the references of the chart-records to each other in the resolution tree consistent.

Input: **one** a list of the same structure as the output of `‘desing/resolveone‘`.

Output: –

The global variables

```
pglobalChartList, presolvedChartList, pchartTree
pchartHistory, pglobalChartCounter
```

are modified as needed.

4.60 `‘desing/resolvedobject‘`

Treating a chart-record with resolved basic object on the top of its stack.

Input: **Chart** a chart record.

Output: –

If the resolved basic object is the first in the stack, the chart-record is moved into `resolvedChartList`, otherwise it is removed from the stack and the chart-record is moved into `globalChartList`.

4.61 `‘desing/resolvem‘`

Resolution of a monomial WBO on the top of the stack of a chart-record.

Input: **Chart** a chart-record.

Output: –

The chart-records get into

```
resolvedChartList, globalChartList, chartHistory
```

appropriately.

4.62 `‘desing/resolvem2‘`

Resolution of a monomial WBO on the top of the stack of a chart-record. The input-output specification is the same as of `‘desing/resolvem‘`. This procedure is used when the configuration variable **skewcover** is true.

4.63 `‘desing/resolveone‘`

Performing one major step in the resolution which can result changes in the global variables storing partial results (see section 3.3).

Input: Chart a chart-record.

Output: a list of the structure:

```
[globalChartList, resolvedChartList, chartHistory, chartTree]
```

This procedure runs in distributed environment, on a client machine. The global variables are initialized with each call, therefore they contain the results of the resolution step of the input chart-record.

4.64 `‘desing/resolves‘`

Resolution of an SBO on the top of the stack of a chart-record.

Input:

Chart a chart-record,

Newobject a boolean indicating whether the SBO is newly created.

Output: –

The procedure starts the resolution of the SBO, and it can lead to changes in the global variables

```
resolvedChartList, globalChartList,
chartHistory, chartTree, globalChartCounter.
```

4.65 `‘desing/resolves2‘`

Resolution of an SBO on the top of the stack of a chart-record. The input-output specification is the same as of `‘desing/resolves‘`. This procedure is used when the configuration variable **skewcover** is true.

4.66 `‘desing/resolvesc‘`

Continuing the resolution of an SBO after the test of `‘desing/clean‘` is performed. It might well be that cover operation was necessary and it produced several new chart-records with the same SBOs in them.

Input:

OL a list of chart-records,

z a list of pairwise different indices to the `IND` member of the chart, being disjoint to the `N` member of the SBO,

DCOEFF the coefficient ideal of the `J` member of the SBO w.r.t. `z`.

Output: –

The procedure either leads to a blowing up operation or it creates a new WBO of one dimension less than the SBO.

4.67 `‘desing/resolvesc2‘`

The functionality of the procedure is the analogue of the one of `‘desing/resolvesc‘`, with the same input-output specifications. This procedure is used when the configuration variable **skewcover** is true.

4.68 `‘desing/resolvesx‘`

Continuing the resolution of an SBO when there was no element found in the `IND` member of the chart, which defines a hypersurface of maximal contact to the singular locus. In this case an exchange operation (and possibly also a cover) was necessary.

Input:

- **OL** a list of chart-records,
- **CLEAN** a boolean that indicates whether the chart has already been tested by `‘desing/clean‘`,
- **Z** the list of exchanged `IND` elements in the format of the output variable (`Z`) of `‘desing/coverx‘`

Output: –

The procedure might extend `resolvedChartList`, or it can lead to a blowing up operation or to a `‘desing/resolvesc‘` call.

4.69 `‘desing/resolvesx2‘`

The functionality of the procedure is the analogue of the one of `‘desing/resolvesx‘`, with the same input-output specifications. This procedure is used when the configuration variable **skewcover** is true.

4.70 `‘desing/resolvet‘`

Resolution of a TBO on the top of the stack of a chart-record.

Input:

- **Chart** a chart-record,

Output: –

The procedure treats a TBO. It updates the order of the ideal, testing resolvedness, and if the object is not resolved, it creates a new SBO, it adds the elements of `EM` to the list `H` and performs cover operation if necessary when the singular locus is away from the intersection of the hypersurfaces of `H`.

4.71 `‘desing/resolvet2‘`

Resolution of a TBO on the top of the stack of a chart-record. The input-output specification is the same as of `‘desing/resolvet‘`. This procedure is used when the configuration variable **skewcover** is true.

4.72 `‘desing/resolvew‘`

Resolution of a WBO on the top of the stack of a chart-record.

Input: Chart a chart-record.

Output: –

The procedure updates the order of the `J` component of the WBO, and decides whether

- it is a resolved object,
- `‘desing/goodpoints‘` is to be called,
- a new TBO is to be created and passed to `‘desing/resolvet‘`,
- or it is a monomial WBO and `‘desing/resolvem‘` is to be called.

4.73 `‘desing/resolvew2‘`

Resolution of a WBO on the top of the stack of a chart-record. The input-output specification is the same as of `‘desing/resolvew‘`. This procedure is used when the configuration variable **skewcover** is true.

4.74 `‘desing/rotation‘`

When the **skewcover** configuration variable is true the blowing up varieties are covered by chart which are obtained by first describing the blowing up center in some rotated coordinate system such that in a chart of the covering of the blowing up allows the elimination of maximal number of variables, i.e. the number of generators for the k-algebra of the new chart remains constant. The procedure applies such a rotation.

Input:

S the variables of the original chart without `DEP2V`,

C the generators of the nonsingular blowing up center with `DEP2V` being eliminated from them,

m the codimension of the center,

SR1 (output parameter) the rotated variables,

SR2 (output parameter) the variables with respect to which we rotated the ones in $SR1$ (the unchanged ones),

SROT (output parameter) the equations of the substitution of the rotation,

ISROT (output parameter) the equations of the substitution of the inverse rotation.

Output: The Gröbner basis of the center computed with elimination order for variables in $SR2$, after a rotation such that the projection to the subspace of the rotated variables is generically one-to-one.

4.75 `desing/rpupdate`

This procedure is the recursive version of `desing/pupdate`. The functionality is the same, but it calls itself recursively to traverse the subtree of the resolution tree that was created by a `desing/resolveone`.

Input:

- **one** a list of the same structure as the output of `desing/resolveone`,
- **ID** the ID of the root of the subtree which is treated under this level of recursion,
- **parent** the new (corrected) ID of the parent of the root.

Output: the new (corrected) ID of the root.

The chart-records with final IDs are stored in the corresponding "p-variables", and the other global variables are updates as needed. The global variables

```
pglobalChartList, presolvedChartList,
pchartTree, pchartHistory, pglobalChartCounter
```

might be changed in this procedure.

4.76 `desing/sing`

Computing the generators of the singular locus of a basic object coming from the `J` component of the object.

Input:

- **J, N, S, IND, PDER** the corresponding members of a basic object and its chart (see section 3.2, and 3.1).

Output: a list of polynomials generating the subideal of the singular locus of the basic object, which comes from the `J` component.

The output also contains the polynomials indicated by `N`.

4.77 `desing/singC`

Computing the generators of the singular locus of a basic object coming from the `C` component of the object.

Input:

- **C, IND** the corresponding members of a basic object and its chart (see section 3.2, and 3.1).

Output: a list of polynomials generating the subideal of the singular locus of the basic object, which comes from the `C` component.

4.78 `desing/singa`

Computing the generators of the singular locus of the monomial part with respect to a given threshold.

Input:

> **E, a, IND** the corresponding members of a basic object and its chart (see section 3.2, and 3.1).
>
> **r** a rational value $0 \leq r < 1$.

Output: a list of polynomials generating the the points of the zero set of the monomial part, represented by `E` and `a`, which are at least of order $1 - r$.

4.79 `desing/singrc`

Computing the partial derivatives of an element of the coordinate ring of a chart in a recursive fashion skipping zero derivatives.

Input:

> **F** a pair (f, b) where f is an element of the coordinate ring of some underlying chart and b is a nonnegative integer,
>
> **t** differentiating parameter indicator (in the subtree of recursive calls only partial derivatives with respect to parameters with index greater than or equal to t are used), in the initial call it should be 1.

Output: -

The procedure uses the global variables: **desingSingS**, **desingSingIND**, **desingSingPDER**, which store the data of the chart for F. The partial derivatives of f up to order $b - 1$ will be generated in the global variable **desingSingSL**.

4.80 `desing/singrc2`

The functionality of the procedure is analogous to the one of `desing/singrc` with the same input-output specification, global variables and side effect. This procedure applies the same recursive scheme as `desing/singrc` but it does not skip the zero derivatives.

4.81 `desing/singw`

Computing the singular locus of a WBO. The difference between this procedure and `desing/sing` is that in case of a WBO the monomial parts are also to be considered in the singular locus computation. This leads to a combinatorial approach which determines the singular locus of the WBO as union of intersections of hypersurfaces that correspond to partial derivatives up to a certain order, from the proper and the monomial part of the ideal of the WBO.

Input:

> **J, N, E, a, c, C, S, IND, DEP, PDER** the corresponding members of a WBO and its chart (see section 3.2, and 3.1).

Output: a list of polynomials generating the ideal of the singular locus of the WBO.

The output also contains the polynomials indicated by `N`.

4.82 `desing/substitute`

Performing variable elimination by substitution in a chart-record.

Input:

- **O** a chart record,
- **SUB** the list of pairs `[k, eq]`, where `k` is an index to the `S` member of the chart, and `eq` is the substitution equation for it.

Output: the chart-record with the given substitutions performed.

The substitutions eliminates the listed affine variables of the chart, reducing drastically the time needed for Gröbner basis computations on its data.

4.83 `desing/testintegrity`

Performing a necessary test of integrity of the chart data of a chart-record.

Input: **O** a chart-record.

Output: true if the chart-record passed the test, false otherwise.

Because of the performance hit this procedure generates, it is only to be used for debugging purposes. It writes the results on the standard output. The periodical checking of chart integrity can be turned on by assigning the configuration variable `dointegritycheck` in the **desingcfg** table.

4.84 `desing/transform`

Computing the transform of the generalized ideal of a basic object of a chart-record in a blowup operation.

Input:

- **J, S, N, DEP** the corresponding members of a basic object and its chart (see section 3.2, and 3.1).
- **p** a polynomial from $k[S]$, a representative of the exceptional divisor w.r.t. which the transform has to be computed.

Output: the transform of `J` (for the format see section 3.2).

4.85 'desing/transformw'

Computing the transform of the generalized ideal of a WBO of a chart-record in a blowup operation.

Input:

- **J, S, N, DEP** the corresponding members of a basic object and its chart (see section 3.2, and 3.1).
- **p** a polynomial from $k[S]$, a representative of the exceptional divisor w.r.t. which the transform has to be computed.
- **r** a rational number between 0 and 1 (inclusive), the ratio of the order of the proper part of the ideal within the singular locus of the WBO, and the order of the proper part without restrictions.

Output: the transform of `J` (for the format see section 3.2).

4.86 'desing/vfn2str'

Conversion of a stratifying function value to string.

Input: **v** the stratifying function value as computed by `'desing/vstfn'` for a given chart record.

Output: The value converted to string with the notation explained below.

We used the definition of the stratifying function which can be found in the paper [EV00].

The values of the subfunctions which are computed in different dimensions are separated by parentheses or brackets. The values of the generic (nontrivial) branch of the function appear in brackets, while the values of the monomial and good-point branches in parentheses (please recall that any value in parentheses is smaller than any other in brackets (provided that they measure resolution problems of the same dimension)).

The order components of the values should be understood as rational orders computed for powered ideals (see, [BS00a, BS01, Bod00]), which are just the `c` components of weighted basic objects. In the same context, the exponents in the a functions can also well be rational numbers.

The last component of the parenthesized function values stands for the list of exceptional divisors defining the blowing up center in the monomial case. The numbers in these lists index the global exceptional divisor list (therefore they can be different from the numbers that appear in the chart data or in the chart tree).

Asterisks mark the case when the codimension-one part of the maximum stratum is not empty, thus the maximal function value is 'infinity'.

4.87 'desing/vfncmp'

Comparison of two stratifying function values.

Input: **va,vb** two stratifying function values as computed by `'desing/vstfn'`.

Output: true if the first function value is greater than the second, otherwise false.

4.88 `‘desing/vfncmpo‘`

Comparison of two stratifying function values defined by two chart records.

Input: **a,b** two chart records with their `IDs` entry containing the stratifying function values as computed by `‘desing/vstfn‘`.

Output: true if the function value of a is greater than the one of b, otherwise false.

This procedure is a wrapper for `‘desing/vfncmp‘`.

4.89 `‘desing/vstfn‘`

Computation of the maximum of the Villamayor stratifying function over a chart record.

Input:

- **O** the chart record from which the function value is to be extracted,
- **C** the the defining indices of the blowing up center in case of monomial objects, which is the result of the Γ function.

Output: A list of lists, where each sublist contains the part of the function value which is extracted from basic objects of the same dimension. The sublists go through the basic objects in decreasing dimension, and the members of a sublist correspond with the definitions of [EV00].

Asterisks mark the case when the codimension-one part of the maximum stratum is not empty, thus the maximal function value is 'infinity'.

The maximal function value over a given chart can be considered completely computed only when the center is determined (i.e. at the point when the algorithm is about to perform a blowing up). In the intermediate stages (for instance at nodes in the HTML tree which represent cover or exchange operations) the function value is only partially computed, or some components (marked by “?”) can even be undetermined.

4.90 `‘desing/writef‘`

Writing out long lines in a file wrapped at the given maximal length.

Input:

- **fd** the descriptor of the target file,
- **a** a name, the text to be written in the file,
- **b** a positive integer, the maximal length of lines.

Output: –

This procedure is used in `‘desing/chtohtml‘` to break long lines into ones that fit better on the screen.

4.91 The access routines

All the data of a chart-record has to be accessed by procedures provided for this purpose. The group of procedures which reads certain data fields out are named by the pattern on the left, those which set the data are on the right

`'desing/access/get*'`	`'desing/access/set*'`.

The 'get' routines take a single parameter, which is either a chart-record (in context of chart data), or a basic object (in context of basic object data) and they return the corresponding data member.

The 'set' routines take two parameters. The first is either a chart-record or a basic object (like above) in which the corresponding field has to be set, and the second is the new value of that field. They return the new object with the appropriate field set.

The complete list is of these procedures is:

`'desing/access/getBO'`	`'desing/access/addBO'`
`'desing/access/getBOList'`	`'desing/access/setBOList'`
`'desing/access/getC'`	`'desing/access/setC'`
`'desing/access/getDEP'`	`'desing/access/setDEP'`
`'desing/access/getDEP2'`	`'desing/access/setDEP2'`
`'desing/access/getDEP2V'`	`'desing/access/setDEP2V'`
`'desing/access/getE'`	`'desing/access/setE'`
`'desing/access/getEM'`	`'desing/access/setEM'`
`'desing/access/getFOCUS'`	`'desing/access/setFOCUS'`
`'desing/access/getTAGS'`	`'desing/access/setTAGS'`
`'desing/access/getH'`	`'desing/access/setH'`
`'desing/access/getIDs'`	`'desing/access/setIDs'`
`'desing/access/getIMAP'`	`'desing/access/setIMAP'`
`'desing/access/getLMAP'`	`'desing/access/setLMAP'`
`'desing/access/getMAP'`	`'desing/access/setMAP'`
`'desing/access/getSCS'`	`'desing/access/setSCS'`
`'desing/access/getIND'`	`'desing/access/setIND'`
`'desing/access/getJ'`	`'desing/access/setJ'`
`'desing/access/getN'`	`'desing/access/setN'`
`'desing/access/getPDER'`	`'desing/access/setPDER'`
`'desing/access/getS'`	`'desing/access/setS'`
`'desing/access/getSET'`	`'desing/access/setSET'`
`'desing/access/getSL'`	`'desing/access/setSL'`
`'desing/access/getSUBST'`	`'desing/access/setSUBST'`
`'desing/access/getSn'`	`'desing/access/setSn'`
`'desing/access/geta'`	`'desing/access/seta'`
`'desing/access/getc'`	`'desing/access/setc'`
`'desing/access/gettype'`	`'desing/access/settype'`

The procedures `getBOList` and its pair access the basic object stack of a chart-record (which is actually a list). The procedures `getBO` and `addBO` are somehow exceptions in terms of their parameters. Both of them receives a basic object stack as their first parameter and has the index of the basic object to be accessed as an additional parameter.

There are the following additional access routines:

`desing/access/addGlobal` it takes a chart record and an integer, and inserts the chart-record into `globalChartList` at the given position. If the position is out of range, the chart-record is inserted to the closest possible place.

`desing/access/emptyBO` it has no parameter and it returns an initial (empty) basic object.

`desing/access/emptych` it has no parameter and it returns an initial (empty) chart-record.

`desing/access/removeBO` it removes the basic object of index given by its second parameter from the stack given by its first parameter. It returns the the new stack.

`desing/access/replace` it takes the parameters `L` a list, `i` an integer, `O` any object. It replaces the `i`-th element of `L` by `O` and returns the new list.

References

[BM91] E. Bierstone and P. Milman. A simple constructive proof of canonical resolution of singularities. In T. Mora and C. Traverso, editors, *Effective methods in algebraic geometry*, pages 11–30. Birkhäuser, Boston, 1991.

[BM97] E. Bierstone and P. Milman. Canonical desingularization in characteristic zero by blowing up the maximum strata of a local invariant. *Invent. math.*, 128(2):207–302, 1997.

[Bod00] G. Bodnár. *Algorithmic Resolution of Singularities*. PhD thesis, Johannes Kepler University, RISC-Linz, 2000.

[BS00a] G. Bodnár and J. Schicho. Automated resolution of singularities for hypersurfaces. *Journal of Symbolic Computation*, 30(4):401–428, 2000.

[BS00b] G. Bodnár and J. Schicho. A computer program for the resolution of singularities. In H. Hauser, J. Lipman, F. Oort, and A. Quirós, editors, *Resolution of Singularities, A research textbook in tribute to Oscar Zariski*, volume 181 of *Progress in Mathematics*, pages 231–238. Birkhäuser, Boston, 2000.

[BS00c] G. Bodnár and J. Schicho. An improved algorithm for the resolution of singularities. In C. Traverso, editor, *Proceedings of ISSAC 2000*, pages 29–36, New York, 2000. Association for Computing Machinery.

[BS01] G. Bodnár and J. Schicho. Two computational techniques for singularity resolution. *Journal of Symbolic Computation*, 32(1-2):39–54, 2001.

[Buc65] B. Buchberger. *An Algorithm for Finding a Basis for the Residue Class Ring of a Zero-Dimensional Polynomial Ideal*. PhD thesis, Universität Innsbruck, Institut für Mathematik, 1965. German.

[Buc85] B. Buchberger. Gröbner Bases: An Algorithmic Method in Polynomial Ideal Theory. In N. K. Bose, editor, *Recent Trends in Multidimensional Systems Theory*, chapter 6. D. Riedel Publ. Comp., 1985.

[BW93] T. Becker and V. Weispfenning. *Gröbner bases - a computational approach to commutative algebra.* Graduate Texts in Mathematics. Springer, New York, 1993.

[EV98] S. Encinas and O. Villamayor. Good points and constructive resolution of singularities. *Acta Math.*, 181:109–158, 1998.

[EV00] S. Encinas and O. Villamayor. A course on constructive desingularization and equivariance. In H. Hauser, J. Lipman, F. Oort, and A. Quirós, editors, *Resolution of Singularities, A research textbook in tribute to Oscar Zariski*, volume 181 of *Progress in Mathematics*, Boston, 2000. Birkhäuser.

[EV01] S. Encinas and O. Villamayor. A new theorem of desingularization over fields of characteristic zero. Preprint, 2001.

[Hir64] H. Hironaka. Resolution of singularities of an algebraic variety over a field of characteristic zero I-II. *Ann. Math.*, 79:109–326, 1964.

[Sch98] W. Schreiner. Distributed maple - user and reference manual. Technical Report 98-05, RISC-Linz, Univ. Linz, A-4040 Linz, 1998.

[Vil89] O. Villamayor. Constructiveness of Hironaka's resolution. *Ann. Scient. Ecole Norm. Sup. 4*, 22:1–32, 1989.

[Vil96] O. Villamayor. Introduction to the algorithm of resolution. In *Algebraic geometry and singularities, La Rabida 1991*, pages 123–154. Birkhäuser, 1996.

[Win88] F. Winkler. A p-adic approach to the computation of gröbner bases. *J. Symb. Comp.*, 6:287–304, 1988.