UNEDA Call Interface Specification

Universal Engine for Decision Analysis

Version 7.21

UNEDA API Call Interface Specification

This is the API specification for the UNEDA (Universal Engine for Decision Analysis) software platform. It contains several layers, the most central of which are the DTL (Decision Tree Layer) functional API layer and its core calculation library TCL (Tree Core Layer). UNEDA is an open-source platform on which decision-analytic software can be built. In its basic form, it can handle methods that conform to classic probability and utility theory, but it can easily be extended to work with any method compatible with additive utilities. UNEDA brings the following features to any implementation:

- Multiple criteria and probabilistic decisions handled uniformly
- Imprecise input in the form of intervals or rankings
- Handling of output overlap from imprecision by belief distributions
- Multiple decision rules in accordance with utility theory
- Several types of sensitivity analyses

UNEDA was developed between 1994 and 2025, first at the Royal Institute of Technology and later at Stockholm University. The first implementations were instances of the framework described in the PhD thesis *Computational Decision Analysis* (Danielson, 1997).[1] The thesis framework was subsequently extended to handle decision trees of arbitrary depth rather than only a single level (hence the ‘T’ in the layer acronyms) and extended to handle multiple criteria rather than only a single criterion.

UNEDA has been used in a large number of projects over the years, both research and commercial. Most projects have built their own layers on top of UNEDA to create the functional interface they wanted for their applications. Two examples of additional layers are bundled together with the basic platform: UNEDA-CAR for cardinal ranking input (otherwise numerical) and UNEDA-SML for a stakeholder group extension. Users are encouraged to build their own interface layers and, if circumstances permit, publish them in order to make them publicly available.

The UNEDA source code is stored in a repository on GitHub and can be downloaded and used free of charge for any purpose.[2] Background material and documentation are available at the UNEDA website.[3]

UNEDA conforms to the theoretical foundations of prescriptive decision theory as described in the book *Foundations of Computational Decision Analysis* (Danielson, 2025).[4]

The UNEDA API commands are divided into 12 major groups: System, Structure, File, Weights, Probabilities, Values, Automatic Scale, Evaluation, Dominance, Miscellaneous, Error Handling, and DTI.

[1] https://people.dsv.su.se/~mad/Computational_Decision_Analysis.pdf

[2] https://github.com/uneda-cda/UNEDA

[3] https://people.dsv.su.se/~mad/UNEDA

[4] https://people.dsv.su.se/~mad/Foundations_of_Computational_Decision_Analysis.pdf

[5] https://creativecommons.org/licenses/by/4.0/

CONTENTS

IMPLEMENTATION

DTL package C language note

ANSI C, ISO C, and Standard C are early standards for the C programming language published by ANSI, ISO, and IEC respectively. The names refer to the original version of the standard known as C89. The first standard for C was published by ANSI. That document was subsequently adopted by ISO/IEC and later revisions published by ISO/IEC have been adopted by ANSI. The standard was completed in 1989 and ratified as *ANSI X3.159-1989 Programming Language C*. This version of the language is mostly referred to as C89 or ANSI C in order to distinguish it from C90 which was ratified by ISO/IEC as *ISO/IEC 9899:1990* with only formatting changes. Thus, the terms C89 and C90 refer to the same language and the terminology of C89 is used in this package since it was conceived in 1994 and then continuously evolved. One reason to stay with C89 is that most C compilers are actually C++ compilers having C as a proper subset. But this subset is quite often only C89 with parts of C99. Thus, portability is ensured by sticking to using mostly C89.

There are a few exceptions to the C89 adherence. They have mostly to do with function declarations. The following additional rules ensure optimal portability:

1. One-line comments from C99 using // are allowed
2. K&R-style functions from C89 are disallowed
3. Anything invalidated in C99 is disallowed, such as:
   - Implicit **int** function declarations
   - Function declarations without parameter specifications
   - Array type in struct without size specification

This should ensure that the UNEDA libraries will compile on any well-known platform using C compilers up to C23 and C++ compilers up to C++23, thus being as future-safe as possible.

REFERENCES

Danielson, M. *Computational Decision Analysis*, PhD thesis, Royal Institute of Technology, 1997.

Danielson, M. *Foundations of Computational Decision Analysis*, Second Edition, Sine Metu, Stockholm, 2025.

DATA TYPES

There are a number of predefined data types in the UNEDA package. These are used for communication between the user layer and UNEDA. Most are based either on *int* or on *double*.

```
typedef double a_vector[MAX_ALTS+1];
typedef a_vector ar_matrix[MAX_ALTS+1];
typedef int ai_vector[MAX_ALTS+1];
typedef ai_vector ai_matrix[MAX_ALTS+1];
typedef double h_vector[MAX_NOPA+1];
typedef h_vector h_matrix[MAX_ALTS+1];
typedef int o_matrix[MAX_ALTS+1][MAX_COPA+1];
typedef double e_matrix[MAX_RESULT+1][MAX_RESULTSTEPS];
typedef int t_row[MAX_NOPA+1];
typedef t_row t_matrix[MAX_ALTS+1];
typedef double ar_col[MAX_ALTS+1];
typedef double cr_col[MAX_CRIT+1];
typedef int ai_col[MAX_ALTS+1];
```

DATA STRUCTURES

The user statements are of two separate types, one for weight statements (user_w_stmt_rec) and the other for probability and value statements (user_stmt_rec).

```
struct user_w_stmt_rec {
      int n_terms;
      int crit[MAX_TERMS+1];
      int sign[MAX_TERMS+1];
      double lobo;
      double upbo;
      };

struct user_stmt_rec {
      int n_terms;
      int alt[MAX_TERMS+1];
      int cons[MAX_TERMS+1];
      int sign[MAX_TERMS+1];
      double lobo;
      double upbo;
      };
```

INDEXING

There are four separate ways of indexing a node or consequence, using either alternative and node number or a node sequence number and using either a total numbering (including intermediate nodes) or a final consequence numbering (excluding intermediate nodes). The numbering is depth-first per alternative in the tree. These four modes (plus two weight modes) are mapped in the table below, and for each command using indexing, the indexing mode is indicated.

Indexing type	Alt. + node	Node sequence	Weight
Total numbering	A1	B1	C1
Final numbering	A2	B2	C2

SYSTEM COMMANDS

Start DTL layer

Call syntax: DTL_init()

```
Return information:
OK    -
ERROR – state error
        frame in use
```

Call semantics: Perform initialisation of UNEDA resources and starts the DTL layer. This must be the first call to UNEDA.

Stop DTL layer

Call syntax: DTL_exit()

```
Return information:
OK    - number of entries written to trace log
ERROR – state error
        frame in use
        memory leak
```

Call semantics: Release resources in UNEDA. This should be the last call to UNEDA. Check the trace log immediately if positive return code.

Abort command

Two versions are available, one for threads or processes sharing addressing space (typically Java callers), the other for interrupt-driven inter-process communication (typically C/C++ callers).

Call syntax: DTL_abort()

```
Return information:
OK    - user abort queued
```

Call semantics: Must be called by a thread or process sharing address space with the rest of UNEDA. The user’s request for abort is registered in UNEDA. UNEDA looks for the nearest safe point to stop the calculation. If little remains of the calculation, it will run to the end with the ordinary return code and the call results are valid. If some more remains of the calculation, it will be aborted with the DTL_USER_ABORT return code and the call results are then invalid.

Call syntax: send SIGINT signal to the UNEDA process

```
Return information:
OK    - user abort queued
```

Call semantics: A mechanism for interrupt-driven inter-process communication. The master process sends an interrupt to a slave UNEDA process. The user's request for abort is registered in UNEDA. UNEDA looks for the nearest safe point to stop the calculation. If little remains of the calculation, it will run to the end with the ordinary return code and the call results are valid. If some more remains of the calculation, it will be aborted with the DTL_USER_ABORT return code and the call results are invalid.

STRUCTURE COMMANDS

Tree structure

Each alternative has its own tree for each criterion. The tree starts with an implicit decision node as node 0 (the root node). The decision tree is expressed as a vector of tree nodes for each alternative. A node is defined as follows:

```
typedef struct tt_node {
    char type;
    int next;
    int down;
    } ttnode;
```

'type' is the node type. Possible types are:

C	Consequence node
D	Decision node
E	Event node

'next' points to the next node at the same level, and 'down' points to the first child of the node (only if the node is an intermediate node of type D or E). The numbering is depth-first. The value zero indicates a null pointer.

Trees are constructed as node vectors, one for each alternative.

```
typedef ttnode ta_tree[MAX_COPA+1];
typedef ta_tree tt_tree[MAX_ALTS+1];
```

Create new frame

There are four types of frames, two *basic* (1-2) and two *compound* (3-4):

1) Flat PS-frame with probabilities, values, and a flat structure (one level, no tree).
2) Tree PS-frame with probabilities, values, and an event tree.
3) Flat PM-frame with probabilities, values, criteria weights, and a flat criteria structure. All criteria have their own event frames.
4) Tree PM-frame with probabilities, values, criteria weights, and a criteria tree. All criteria have their own event frames.

The compound types consist of a basic PM-frame containing the multi-criteria weight structure (tree or flat) and slots for holding criteria frames in the form of PS-sub-frames, thus creating an illusion of a single PM-frame. The PS-sub-frames are independent and possible to import and export to/from the PM-frame slots. If a slot is unoccupied, a stand-in evaluation of the slot is done for PM-frame evaluations. The stand-in evaluation corresponds to an empty PS-sub-frame.

In addition, there are three pseudo-types of frames that, when created, will be morphed into PM-frames. These types can be seen as syntactic sugar to simplify the creation of PM-frames. DM-frames are multi-criteria frames but without event trees, i.e. each criterion has exactly one consequence for each alternative. SM-frames are multi-stakeholder DM-frames, where each stakeholder has a unique set of criteria weights.

Call syntax (1): DTL_new_PS_flat_frame(int ufnbr, int n_alts, int n_cons[])

```
Return information:
OK    -
ERROR - input error
        frame unknown
        frame exists
        too many alternatives
        too many consequences
```

Call semantics: Creates a new probabilistic flat frame with one criterion and an initial flat structure as specified in the call. The frame receives the frame number ‘ufnbr’. A frame cannot have less than two alternatives. Each alternative must have at least one consequence. The frame is not loaded and can be filled with data prior to loading.

Call syntax (2): DTL_new_PS_tree_frame(int ufnbr, int n_alts, int n_nodes[], tt_tree xtree)

```
Return information:
OK    -
ERROR - input error
        tree error
        frame unknown
        frame exists
        too many alternatives
        too many consequences
```

Call semantics: Creates a new probabilistic tree frame with one criterion and a tree as specified in the call. The frame receives the frame number ‘ufnbr’. A frame cannot have less than two alternatives. Each alternative must have at least one node. ‘n_nodes’ does not include the root node. The frame is not loaded and can be filled with data prior to loading. The tree is specified for each alternative as node pointers ‘next’ and ‘down’ for each node. ‘next’ points to the next node at the same level, and ‘down’ points to the children of the node (only if the node is an intermediate node). The value 0 indicates a null pointer.

Call syntax (3): DTL_new_PM_flat_frame(int ufnbr, int n_crit, int n_alts)

```
Return information:
OK    -
ERROR - input error
        tree error
        frame unknown
        frame exists
        too many criteria
        too many alternatives
```

Call semantics: Creates a new probabilistic multi-criteria frame with ‘n_crit’ criteria and ‘n_alts’ alternatives as specified in the call. The frame receives the frame number ‘ufnbr’. A frame cannot have less than two alternatives. The frame is not loaded and can be filled with data prior to loading.

Call syntax (4): DTL_new_PM_tree_frame(int ufnbr, int n_alts, int n_wtnodes, ta_tree wtree)

```
Return information:
OK    -
ERROR - input error
        tree error
        frame unknown
        frame exists
        too many criteria
        too many alternatives
```

Call semantics: Creates a new probabilistic multi-criteria tree frame with as many criteria as there are end nodes in the weight tree as specified in the call. The weight tree is supplied in the call, but the trees for the criteria are supplied in separate calls to DTL_new_PM_crit_tree or DTL_load_PM_crit. A frame cannot have less than two alternatives. The weight tree must have at least one node. ‘n_wtnodes’ does not include the root node. The frame is not loaded and can be filled with data prior to loading. The weight tree is specified for each alternative as node pointers ‘next’ and ‘down’ for each node. ‘next’ points to the next node at the same level, and ‘down’ points to the children of the node (only if the node is an intermediate node). The value 0 indicates a null pointer.

Call syntax (5): DTL_new_DM_flat_frame(int ufnbr, int n_crit, int n_alts)

```
Return information:
OK    -
ERROR - input error
        frame unknown
        frame exists
        too many criteria
        too many alternatives
```

Call semantics: Creates a new deterministic PM-frame with ‘n_crit’ criteria and ‘n_alts’ alternatives as specified in the call. Deterministic means that each alternative under each criterion has only one consequence, i.e. no event tree. The frame receives the frame number ‘ufnbr’. A frame cannot have less than two alternatives.

Call syntax (6): DTL_new_DM_tree_frame(int ufnbr, int n_alts, int n_wtnodes, ta_tree wtree)

```
Return information:
OK    -
ERROR - input error
```

tree error
frame unknown
frame exists
too many criteria
too many alternatives

Call semantics: Creates a new deterministic PM-tree with as many criteria as there are end nodes in the weight tree as specified in the call, and 'n_alts' alternatives. The weight tree (having 'n_wtnodes' nodes) is supplied in the call and deterministic stubs are created automatically for each criterion. Deterministic means that each alternative under each criterion has only one consequence, i.e. no event tree. The frame receives the frame number 'ufnbr'. A frame cannot have less than two alternatives.

Call syntax (7): DTL_new_SM_tree_frame(int ufnbr, int mode, int n_alts, int n_sh, int n_wtnodes, ta_tree wtree)

Mode: 0 Only copy stakeholder 1 to all other stakeholders
1 Create SM mother frame + copy
2 Create PS criteria frames + copy
3 Create SM mother frame + PS criteria frames + copy

Return information:
OK -
ERROR - input error
tree error
frame unknown
frame exists
too many stakeholders
too many criteria
too many alternatives

Call semantics: Creates a new deterministic combined stakeholder-criteria weight tree (having 'n_wtnodes' nodes) with as many stakeholders as specified in 'n_sh' and as many criteria as there are end nodes in the weight tree for a single stakeholder, and 'n_alts' alternatives. The weight tree is supplied in the call and deterministic stubs are created automatically for each stakeholder and criterion. Deterministic means that each alternative under each criterion has only one consequence, i.e. no event tree. The frame receives the frame number 'ufnbr'. A frame cannot have less than two alternatives. NOTE: The caller supplies the combined stakeholder-criteria tree in the call. It is up to the caller to supply a symmetric stakeholder hierarchy in which the lowest level contains the criteria (i.e. the frame has many stakeholder levels but only one criterion level) since this is a mostly stakeholder-focused function.

Create new criterion

Call syntax: DTL_new_PM_crit_tree(int crit, int n_nodes[], tt_tree xtree)

Return information:
OK -
ERROR - input error
tree error
criterion exists

```
        criterion unknown
        wrong frame type
        too many consequences
```

Call semantics: Creates a new criterion ‘crit’ with a tree as specified in the call. The criterion is added to the loaded PM-frame. The tree is specified for each alternative as node pointers ‘next’ and ‘down’ for each node. ‘next’ points to the next node at the same level, and ‘down’ points to the children of the node (only if the node is an intermediate node). The value 0 indicates a null pointer.

Load criterion from frame

Call syntax: DTL_load_PM_crit(int crit, int ufnbr)

```
Return information:
OK    -
ERROR – criterion unknown
        no such frame
        frame not loaded
        alternative mismatch
        wrong frame type
```

Call semantics: Imports the PS-frame ‘ufnbr’ to the criterion slot ‘crit’ in the loaded PM-frame. The user frame containing the PS-frame is disposed of.

Unload criterion to frame

Call syntax: DTL_unload_PM_crit(int crit, int new_ufnbr)

```
Return information:
OK    -
ERROR - criterion unknown
        no such frame
        frame not loaded
        frame exists
        wrong frame type
```

Call semantics: Exports criterion ‘crit’ in the loaded PM-frame into the PS-frame ‘ufnbr’. A user frame containing the PS-frame is created.

Delete a criterion

Call syntax: DTL_delete_PM_crit(int crit)

```
Return information:
OK    -
ERROR - criterion unknown
        frame not loaded
        wrong frame type
```

Call semantics: Deletes the criterion in the slot ‘crit’ from the loaded PM-frame. The criterion cannot subsequently be recovered into a PS-frame.

Check frame type

Call syntax: DTL_frame_type(int ufnbr, int *type)

```
Return information:
OK    -
ERROR – input error (ufnbr out of range)
```

Call semantics: Checks the type of the frame ‘ufnbr’. Supplying 0 as ‘ufnbr’ indicates the currently loaded frame. Returns the frame type in ‘type’ if the frame number is associated with a user frame in UNEDA and 0 otherwise.

Check criterion

Call syntax: DTL_PM_crit_exists(int crit, int *exists)

```
Return information:
OK    -
ERROR – criterion unknown
        frame not loaded
        wrong frame type
```

Call semantics: Checks if the criterion exists for a PM-frame. Returns TRUE in ‘exists’ if the criterion slot number ‘crit’ is associated with a frame and FALSE otherwise.

Dispose of frame

Call syntax: DTL_dispose_frame(int ufnbr)

```
Return information:
OK    -
ERROR – frame in use
        frame unknown
```

Call semantics: Dispose of resources belonging to frame ‘ufnbr’ and free the position for a new frame. NOTE: Frames can only be disposed of when no frame is open.

Load frame

Call syntax: DTL_load_frame(int ufnbr)

```
Return information:
OK    - for PM-frames: number of connected probability trees
ERROR – frame unknown
        frame corrupted
        frame in use
        inconsistent
```

Call semantics: Attempts to attach the frame ‘ufnbr’ to TCL. Bases are loaded and checked for consistency. If any base is inconsistent, the frame will not be attached (loaded).

Close frame

Call syntax: DTL_unload_frame()

Return information:
OK   -
ERROR - frame not loaded

Call semantics: Detach the frame from TCL and free the interface for new frames. NOTE: In case of internal problems in TCL, the frame might be detached without an explicit call to DTL_unload_frame.

Get frame name

Call syntax: DTL_frame_name(string(fname), int *ftype)

Return information:
OK   -
ERROR - frame not loaded

Call semantics: Returns the name and the type (1=PS, 2=PM) of the current frame.

Check load status

Call syntax: DTL_load_status(int *f_loaded)

Return information:
OK   -
ERROR -

Call semantics: Checks if any user frame is loaded. Returns the user frame number in 'f_loaded' if a frame is loaded and 0 otherwise.

FILE COMMANDS

UNEDA can read and write files with its frame content. The files have the extension .dmc and are text-based files. In addition, UNEDA can read files from the UCL core tester tool with the extension .ddt which are also text-based. The format of the file types are described in the source files.

Read frame from file

Call syntax: DTL_read_frame(int ufnbr, char *fn, char *folder , int mode)
Call syntax: DTL_read_ddt_frame(int ufnbr, char *fn, char *folder , int mode)

```
Return information:
OK    -
ERROR - file corrupt
        file/folder unknown
        frame exists
```

Call semantics: Reads the file ‘fn’ in folder ‘folder’ and creates a user frame ‘ufnbr’ from the file. The file should have been previously written by DTL_write_frame (if calling DTL_read_frame) or the UCT core tester tool (if calling DTL_read_ddt_frame). The former reads a .dmc file and the latter a .ddt file. If ‘mode’ is set (non-zero), the frame name found in the file will replace the content of ‘fn’.

Write frame to file

Call syntax: DTL_write_frame(char *fn, char *folder)

```
Return information:
OK    -
ERROR – frame not loaded
        frame corrupt
```

Call semantics: Writes the currently loaded user frame to the file ‘fn’ in folder ‘folder’ in .dmc or .ddt format.

WEIGHT COMMANDS

Weights can be criteria weights, stakeholder weights, or both. All kinds of weights in the weight hierarchy (tree) are treated in the same way within the two node categories: intermediate nodes and end (real) nodes.

Add weight statement

Call syntax: DTL_add_W_statement(struct user_w_stmt_rec* uwstmtp)

```
Return information:
OK    - statement number in the weight base
ERROR - input error
        frame not loaded
        wrong frame type
        too many statements
        too narrow statement
        inconsistent
```

Call semantics: Add the user weight statement w(node) = [lobo,upbo] to the weight base within the decision frame. The base is checked for consistency with respect to the new interval. In case of inconsistency, nothing is added to the base. Indexing type: C1. NOTE: ‘node’ is the node number in the weight tree.

Change bounds of weight statement

Call syntax: DTL_change_W_statement(int stmt_number, double lobo, double upbo)

```
Return information:
OK    -
ERROR - input error
        statement error
        frame not loaded
        wrong frame type
        too narrow statement
        inconsistent
```

Call semantics: Change the existing user weight statement w(node) = [old_lobo,old_upbo] to w(node) = [lobo,upbo] in the weight base. The base is checked for consistency with respect to the change. In case of inconsistency, the call is rolled back and nothing is changed in the base. NOTE: 'node1' and 'node2' are node numbers in the weight tree.

Replace weight statement

Call syntax: DTL_replace_W_statement(int stmt_number, struct user_w_stmt_rec* uwstmtp)

```
Return information:
OK    -
ERROR - input error
        statement error
        frame not loaded
        wrong frame type
        too narrow statement
        inconsistent
```

Call semantics: Replace the user weight statement with w(node) = [lobo,upbo] in the weight base. The base is checked for consistency with respect to the new interval. In case of inconsistency, the call is rolled back and nothing is replaced in the base. Indexing type: C1. NOTE: 'node' is the node number in the weight tree.

Delete weight statement

Call syntax: DTL_delete_W_statement(int stmt_number)

```
Return information:
OK    - number of statements remaining in the weight base
ERROR - input error
        frame not loaded
        wrong frame type
```

Call semantics: The user weight statement with position 'stmt_number' in the weight base is deleted from the base. All statements with higher positions within the base are shifted one position down.

Set weight midpoint

Call syntax: DTL_add_W_mid_statement(struct user_w_stmt_rec* uwstmtp)

```
Return information:
OK    -
ERROR - input error
        statement error
        frame not loaded
        wrong frame type
        inconsistent
```

Call semantics: Add the user weight midpoint w(node) = [lobo,upbo] to the weight base within the decision frame. The base is checked for consistency with respect to the new interval. In case of inconsistency, nothing is added to the base. Indexing type: C1. NOTE: ‘node’ is the node number in the weight tree.

Remove weight midpoint

Call syntax: DTL_delete_W_mid_statement(struct user_w_stmt_rec* uwstmtp)

```
Return information:
OK    -
ERROR - input error
        statement error
        frame not loaded
        wrong frame type
```

Call semantics: The user weight midpoint w(node) = [lobo,upbo] in the weight base is deleted from the base. This can be considered unlocking the midpoint. Indexing type: C1. NOTE: ‘node’ is the node number in the weight tree.

Set weight range box

Call syntax: DTL_set_W_box(h_vector lobox, h_vector upbox)

```
Return information:
OK    -
ERROR - wrong frame type
        frame not loaded
        inconsistent
```

Call semantics: Range statements for all criteria weights (in two vectors ‘lobox’ and ‘upbox’ indexed as [node]) are added to the weight base at the same time. The base is checked for consistency with respect to all new ranges. In case of inconsistency, nothing is added to the base. Indexing type: C1. NOTE: ‘node’ is the node number in the weight tree.

Set weight midpoint box

Call syntax: DTL_set_W_mbox(h_vector lobox, h_vector upbox)
Call syntax: DTL_set_W_mbox1(h_vector mbox)

```
Return information:
OK    -
ERROR - wrong frame type
        frame not loaded
        inconsistent
```

Call semantics: Midpoints for all criteria (in one or two vectors indexed as [node]) are added to the weight base at the same time. An inactive entry is marked -1.0 in ‘lobox’ or ‘mbox’. The base is checked for consistency with respect to all new midpoints. In case of inconsistency, nothing is added to the base. Indexing type: C1. NOTE: ‘node’ is the node number in the weight tree.

Remove weight midpoint box

Call syntax: DTL_remove_W_mbox()

```
Return information:
OK    -
ERROR - wrong frame type
        frame not loaded
```

Call semantics: Range statements added by DTL_set_W_box for all criteria are removed from the weight base. Range statements added by DTL_add_W_statement for any criteria remain in the weight base.

Get weight hull

Call syntax: DTL_get_W_hull(int global, h_vector lobo, h_vector mid, h_vector upbo)

```
Return information:
OK    -
ERROR - wrong frame type
        frame not loaded
        too many consequences
```

Call semantics: The global (‘global’=1) or local (‘global’=0) hull and midpoint are returned in three vectors ‘lobo’, ‘mid’, and ‘upbo’ indexed as [node]. Indexing type: C1. NOTE: ‘node’ is the node number in the tree.

Reset weight base

Call syntax: DTL_reset_W_base()

```
Return information:
OK    -
ERROR - frame not loaded
```

Call semantics: Deletes all weight statements in the weight base. The weight range box is also deleted.

PROBABILITY COMMANDS

All events modelled as event trees have probabilities associated with them. The probabilities are standard Kolmogorov probabilities and conform to the standard axioms. Thus, the requirement of every probability variable is that all its consistent instantiations are consistent with the requirement to sum to one over all events modelled as exhaustive and mutually exclusive.

Add probability statement

Call syntax: DTL_add_P_statement(int crit, struct user_stmt_rec* ustmtp)

```
Return information:
OK    - statement number in the probability base
ERROR - input error
        criterion unknown
        statement error
        frame not loaded
        wrong frame type
        too many statements
        too narrow statement
        inconsistent
```

Call semantics: Add the user probability statement p(alt:node) = [lobo,upbo] to the probability base within the criterion ’crit’. The base is checked for consistency with respect to the new interval. In case of inconsistency, nothing is added to the base. Indexing type: A1. NOTE: ‘lobo’ and ‘upbo’ contain local probabilities.

Change bounds of probability statement

Call syntax: DTL_change_P_statement(int crit, int stmt_number, double lobo, double upbo)

```
Return information:
OK    -
ERROR - input error
        criterion unknown
        statement error
        wrong frame type
        too narrow statement
        inconsistent
```

Call semantics: Change the existing user probability statement p(alt:node) = [old_lobo,old_upbo] to p(alt:node) = [lobo,upbo] in the probability base of the criterion ‘crit’. The base is checked for consistency with respect to the change. In case of inconsistency, nothing is changed in the base. NOTE: ‘lobo’ and ‘upbo’ contain local probabilities.

Replace probability statement

Call syntax: DTL_replace_P_statement(int crit, int stmt_number, struct user_stmt_rec* ustmtp)

```
Return information:
OK    -
ERROR - input error
        criterion unknown
        statement error
        frame not loaded
        wrong frame type
        too narrow statement
        inconsistent
```

Call semantics: Replace the user probability statement with p(alt:node) = [lobo,upbo] in the probability base of the criterion ‘crit’. The base is checked for consistency with respect to the new interval. In case of inconsistency, nothing is replaced in the base. Indexing type: A1. NOTE: ‘lobo’ and ‘upbo’ contain local probabilities.

Delete probability statement

Call syntax: DTL_delete_P_statement(int crit, int stmt_number)

```
Return information:
OK    - number of statements remaining in the probability base
ERROR - input error
        criterion unknown
        frame not loaded
        wrong frame type
```

Call semantics: The user probability statement with position ‘stmt_number’ in the probability base of the criterion ‘crit’ is deleted from the base. All statements with higher positions within the base are shifted one position down.

Set probability midpoint

Call syntax: DTL_add_P_mid_statement(int crit, struct user_stmt_rec* ustmtp)

```
Return information:
OK    -
ERROR - input error
        criterion unknown
        statement error
        frame not loaded
        wrong frame type
        inconsistent
```

Call semantics: Add the user probability midpoint p(alt:node) = [lobo,upbo] to the probability base within the decision frame of the criterion ‘crit’. The base is checked for consistency with respect to the new interval. In case of inconsistency, nothing is added to the base. Indexing type: A1. NOTE: ‘lobo’ and ‘upbo’ contain local probabilities.

Remove probability midpoint

Call syntax: DTL_delete_P_mid_statement(int crit, struct user_stmt_rec* ustmtp)

```
Return information:
OK    -
ERROR - input error
        criterion unknown
        statement error
        frame not loaded
```

Call semantics: The user probability midpoint for the consequence (alt:node) in the probability base of the criterion ‘crit’ is deleted from the base. This can be considered unlocking the midpoint. Indexing type: A1.

Set probability range box

Call syntax: DTL_set_P_box(int crit, h_matrix lobox, h_matrix upbox)

```
Return information:
OK    -
ERROR - wrong frame type
        criterion unknown
        frame not loaded
        inconsistent
```

Call semantics: Range statements for all consequences (in two matrices ‘lobox’ and ‘upbox’ indexed as [alt][node]) are added to the probability base of the criterion ‘crit’ at the same time. The base is checked for consistency with respect to all new ranges. In case of inconsistency, nothing is added to the base. Indexing type: A1. NOTE: ‘lobox’ and ‘upbox’ must contain local probabilities.

Set probability midpoint box

Call syntax: DTL_set_P_mbox(int crit, h_matrix lobox, h_matrix upbox)
Call syntax: DTL_set_P_mbox1(int crit, h_matrix mbox)

```
Return information:
OK    -
ERROR - wrong frame type
        criterion unknown
        frame not loaded
        inconsistent
```

Call semantics: Midpoints for all consequences (in one or two matrices indexed as [alt][node]) are added to the probability base of the criterion ‘crit’ at the same time. An inactive entry is marked -1.0 in ‘lobox’ or ‘mbox’. The base is checked for consistency with respect to all new midpoints. In case of inconsistency, nothing is added to the base. Indexing type: A1. NOTE: ‘lobox’ and ‘upbox’ must contain local probabilities.

Remove probability midpoint box

Call syntax: DTL_remove_P_mbox(int crit)

```
Return information:
OK    -
ERROR - wrong frame type
        criterion unknown
        frame not loaded
```

Call semantics: Midpoints added by DTL_set_P_mbox for all consequences are removed from the probability base of the criterion ‘crit’. Range statements added by DTL_add_P_statement for any consequences remain in the probability base.

Get probability hull

Call syntax: DTL_get_P_hull(int crit, int global, h_matrix lobo, h_matrix mid, h_matrix upbo)

```
Return information:
OK    -
ERROR - wrong frame type
        criterion unknown
        frame not loaded
        too many consequences
```

Call semantics: The global (‘global’=1) or local (‘global’=0) hull and midpoint of the criterion ‘crit’ are returned in three matrices ‘lobo’, ‘mid’, and ‘upbo’ indexed as [alt][node]. Indexing type: A1.

Reset probability base

Call syntax: DTL_reset_P_base(int crit)

```
Return information:
OK    -
ERROR - frame not loaded
        criterion unknown
```

Call semantics: Deletes all statements in the probability base. The probability range box is also deleted.

VALUE COMMANDS

Add value statement

Call syntax: DTL_add_V_statement(int crit, struct user_stmt_rec* ustmtp)

```
Return information:
OK    - statement number in the value base
ERROR - input error
        criterion unknown
        statement error
        frame not loaded
        too many statements
        too narrow statement
        inconsistent
```

Call semantics: Add the user value statement v(alt:node) = [lobo,upbo] to the value base within the decision frame of the criterion ‘crit’. The base is checked for consistency with respect to the new interval. In case of inconsistency, for a loaded frame, nothing is added to the base. Indexing type: A1.

Change bounds of value statement

Call syntax: DTL_change_V_statement(int crit, int stmt_number, double lobo, double upbo)

```
Return information:
OK    -
ERROR - input error
        criterion unknown
        statement error
        too narrow statement
        inconsistent
```

Call semantics: Change the existing user value statement v(alt:node) = [old_lobo,old_upbo] to v(alt:node) = [lobo,upbo] in the value base of the criterion ‘crit’. The base is checked for consistency with respect to the change. In case of inconsistency, nothing is changed in the base.

Replace value statement

Call syntax: DTL_replace_V_statement(int crit, int stmt_number, struct user_stmt_rec* ustmtp)

```
Return information:
OK    -
ERROR - input error
        criterion unknown
        statement error
        frame not loaded
        too narrow statement
        inconsistent
```

Call semantics: Replace the user value statement with v(alt:node) = [lobo,upbo] in the value base of the criterion ‘crit’. The base is checked for consistency with respect to the new interval. In case of inconsistency, nothing is replaced in the base. Indexing type: A1.

Delete value statement

Call syntax: DTL_delete_V_statement(int crit, int stmt_number)

```
Return information:
OK    - number of statements remaining in the value base
ERROR - input error
        criterion unknown
        frame not loaded
```

Call semantics: The user value statement with position ‘stmt_number’ in the value base is deleted from the base of the criterion ‘crit’. All statements with higher positions within the base are shifted one position down.

Set value midpoint

Call syntax: DTL_add_V_mid_statement (int crit, struct user_stmt_rec* ustmtp)

```
Return information:
OK    -
ERROR - input error
        criterion unknown
        statement error
        frame not loaded
        inconsistent
```

Call semantics: Add the user value midpoint v(alt:node) = [lobo,upbo] to the value base within the criterion ‘crit’. The base is checked for consistency with respect to the new interval. In case of inconsistency, nothing is added to the base. Indexing type: A1.

Remove value midpoint

Call syntax: DTL_delete_V_mid_statement(int crit, struct user_stmt_rec* ustmtp)

```
Return information:
OK    -
ERROR - input error
        criterion unknown
        statement error
        frame not loaded
```

Call semantics: The user value midpoint for the consequence (alt:node) in the value base of the criterion ‘crit’ is deleted from the base. This can be considered unlocking the midpoint. Indexing type: A1.

Set value range box

Call syntax: DTL_set_V_box(int crit, h_matrix lobox, h_matrix upbox)

```
Return information:
OK    -
ERROR - wrong frame type
        criterion unknown
        frame not loaded
        inconsistent
```

Call semantics: Range statements for all consequences (in two matrices ‘lobox’ and ‘upbox’ indexed as [alt][node]) are added to the value base of the criterion ‘crit’ at the same time. The base is checked for consistency with respect to all new ranges. In case of inconsistency, nothing is added to the base. Indexing type: A1.

Set modal value box

Call syntax: DTL_set_V_modal(int crit, int mode, h_matrix lobox, h_matrix modex, h_matrix upbox)

```
Mode:  0 = Default
      +1 = Clear mbox first
      +2 = Set box at end

Return information:
OK    - number of modal values entered
ERROR - wrong frame type
        criterion unknown
        frame not loaded
        inconsistent
```

Call semantics: Range and mode statements for all consequences (in three matrices ‘lobox’, ‘modex’, and ‘upbox’ indexed as [alt][node]) are added to the value base of the criterion ‘crit’ at the same time. The matrix ‘modex’ is filled with mode values (compared to the mbox which is instead filled with midpoint values). In case of inconsistency, nothing is added to the base. Indexing type: A1.

Set value midpoint box

Call syntax: DTL_set_V_mbox(int crit, h_matrix lobox, h_matrix upbox)
Call syntax: DTL_set_V_mbox1(int crit, h_matrix mbox)

```
Return information:
OK    -
ERROR - wrong frame type
        criterion unknown
        frame not loaded
        inconsistent
```

Call semantics: Midpoints for all consequences (in one or two matrices indexed as [alt][node]) are added to the value base of the criterion ‘crit’ at the same time. An inactive entry is marked -1.0 in ‘lobox’ or ‘mbox’. The base is checked for consistency with respect to all new midpoints. In case of inconsistency, nothing is added to the base. Indexing type: A1.

Remove value midpoint box

Call syntax: DTL_remove_V_mbox(int crit)

```
Return information:
OK    -
ERROR - wrong frame type
        criterion unknown
        frame not loaded
```

Call semantics: Midpoints added by DTL_set_V_mbox for all consequences are removed from the value base of the criterion ‘crit’.

Get value hull

Call syntax: DTL_get_V_hull(int crit, h_matrix lobo, h_matrix mid, h_matrix upbo)

```
Return information:
OK    -
```

```
ERROR - frame not loaded
        criterion unknown
        too many consequences
```

Call semantics: The hull and the midpoint of the criterion ‘crit’ are returned in three matrices ‘lobo’, ‘mid’, and ‘upbo’ indexed as [alt][node]. Indexing type: A1.

Get value modals

Call syntax: DTL_get_V_modal(int crit, h_matrix modal)
Call syntax: DTL_check_V_modality(int crit, int Ai, int Aj)

```
Return information:
OK    - number of non-modal variables or bases (DTL_OK if all modal)
ERROR - wrong frame type
        criterion unknown
        frame not loaded
        too many consequences
```

Call semantics: The most likely (modal) value point of the criterion ‘crit’ is returned in a matrix ‘modal’ indexed as [alt][node]. For checking the true modality of the value base, the number of variables with overhang (‘crit’>0) or value bases where overhang exists in at least one node (‘crit’=0) for alternatives ‘Ai’ and ‘Aj’ is returned (‘crit’=0 scans all criteria, ‘Ai’=0 scans all alternatives). Indexing type: A1.

Reset value base

Call syntax: DTL_reset_V_base(int crit)

```
Return information:
OK    -
ERROR - frame not loaded
        criterion unknown
```

Call semantics: Deletes all value statements for criterion ‘crit’ in the value base. The value box is also deleted.

AUTOMATIC SCALE COMMANDS

The automatic scale functionality is a user layer implementation on top of UNEDA (which differentiates this section from all the other sections in the specification). It does not have the privileges of UNEDA functions but has to call them like any other user. The add-in is provided as syntactic sugar to alleviate the implementation of criteria value scales that differ from the built-in [0,1] scales in DTL. The autoscale function for a particular criterion is turned **on** by calling DTL_set_AV_box and turned **off** by calling DTL_reset_AV_scale. NOTE: Since this is a layer on top of the value base, and as such using the standard calls DTL_set_V_box, DTL_set_V_mbox, and DTL_set_V_modal, care must be taken to reset the scales if subsequent standard [0,1] calls are being used without this layer (i.e., in essence, bypassing the layer). In general, a user layer on top should manage its own integrity on top of the package.

Set autoscale value box

Call syntax: DTL_set_AV_box(int crit, int rev, int renorm, h_matrix lobox, h_matrix upbox)
Call syntax: DTL_set_AV_modal(int crit, int mode, int rev, int renorm, h_matrix lobox, h_matrix modalx, h_matrix upbox)

```
Return information:
OK    -
ERROR - input error
        criterion unknown
        inconsistent
```

Call semantics: Range statements for all consequences (in two or three matrices 'lobox' and 'upbox' (plus 'modalx' for the modal version of the call) indexed as [alt][node]) are added to the value base of the criterion 'crit'. These ranges can consist of arbitrary values and the autoscale will be set accordingly. The parameter 'rev' indicates whether higher values are preferred (set to 0) or lower values (set to 1, reverse scale). The parameter 'renorm' indicates if the criteria weights should be renormalised as a consequence of scale changes. For the parameter 'mode', refer to the specification of DTL_set_V_modal. This call turns the autoscale functionality on. The base is checked for consistency with respect to all new ranges. In case of inconsistency, nothing is added to the base and the scale remains unchanged. Indexing type: A1.

Set autoscale value midpoint box

Call syntax: DTL_set_AV_mbox(int crit, h_matrix lobox, h_matrix upbox)
Call syntax: DTL_set_AV_mbox1(int crit, h_matrix mbox)

```
Return information:
OK    -
ERROR – input error
        criterion unknown
        inconsistent
```

Call semantics: Midpoints for all consequences (in one or two matrices indexed as [alt][node]) are added to the value base of the criterion 'crit'. An inactive entry is marked -1.0 in 'lobox' or 'mbox'. The values refer to the scale set implicitly in a call to DTL_set_AV_box which must precede this call. The base is checked for consistency with respect to all midpoints. In case of inconsistency, nothing is added to the base. Indexing type: A1.

Get criterion scale

Call syntax: DTL_get_AV_crit_scale(int crit, double *v_min, double *v_max)

```
Return information:
OK    -
ERROR - frame not loaded
        criterion unknown
```

Call semantics: Get the scale endpoints of a criterion scale.

Set multi-criteria scale

Call syntax: DTL_set_AV_MC_scale(double v_min, double v_max)

```
Return information:
OK    -
ERROR - wrong frame type
        frame not loaded
        input error
```

Call semantics: Sets the endpoints of the multi-criteria scale. Further, it is only allowed for PM-frames (PS are set automatically). To have lower values being preferred (reverse scale), enter v_min larger than v_max. NOTE: Only the MC scale is allowed to be set manually, otherwise the meaning of value statements would change.

Copy multi-criteria scale

Call syntax: DTL_copy_AV_MC_scale(int crit)

```
Return information:
OK    -
ERROR - wrong frame type
        frame not loaded
        criterion unknown
```

Call semantics: Copies the endpoints of the scale of the criterion 'crit' specified in the call onto the multi-criteria scale. This call equalises the two scales' endpoints.

Reset multi-criteria scale

Call syntax: DTL_reset_AV_MC_scale()

```
Return information:
OK    -
ERROR - wrong frame type
        frame not loaded
```

Call semantics: Resets the autoscale for the multi-criteria scale. This call turns the autoscale functionality off for multi-criteria.

Get multi-criteria scale

Call syntax: DTL_get_AV_MC_scale(double *v_min, double *v_max)

```
Return information:
OK    -
ERROR - wrong frame type
        frame not loaded
```

Call semantics: Get the scale endpoints of the multi-criteria scale.

For all conversion functions, the scale type should be supplied. The types are encoded as constants as follows:

		rel	neg	|x|	scale type	constants
Type field:	1	N	N	N	absolute	ABS_SCALE
	2	Y	Y	N	difference	DIFF_SCALE
	3	Y	N	Y	distance	DIST_SCALE
	4	Y	Y	Y	reverse diff	REVD_SCALE (*)

Legend: rel = relative scale (default absolute)
neg = allow negative norm input [-1,0]
|x| = trim to non-negative output [0,a]

(*) = Reverse difference scale: it treats a reverse
scale [b,a] (b>a) as if it were a scale [a,b]

Notes:
1. Conversions are necessary only when the user scale is set to be different from [0,1] (scale dependent).
2. The parameter 'crit' is a criterion number, not an evaluation (partial = crit<0) marker.
3. Interval calls handle reverse scales, use them instead of vector calls for intervals.

Convert to autoscale user vector

Call syntax: DTL_get_AV_user_vector(int crit , int type, int size, double v_val[], double av_val[])

Return information:
OK -
ERROR - input error
frame not loaded
criterion unknown

Call semantics: Get the scale transformation of a vector of values in criterion ‘crit’ from a [0,1] scale to the determined user scale.

Convert to single autoscale user value

Call syntax: DTL_get_AV_user_value(int crit , int type, double v_val, double *av_val)

Return information:
OK -
ERROR - input error
frame not loaded
criterion unknown

Call semantics: Get the transformation of a single value in criterion ‘crit’ from a [0,1] scale to the determined user scale.

Convert to autoscale user intervals

Call syntax: DTL_get_AV_user_intervals(int crit , int type, int size, double v_lobo[], double v_upbo[], double av_lobo[], double av_upbo[])
Call syntax: DTL_get_AV_user_interval(int crit , int type, double v_lobo, double v_upbo, double *av_lobo, double *av_upbo)

```
Return information:
OK    -
ERROR - input error
        frame not loaded
        criterion unknown
```

Call semantics: Gets the scale transformation of vectors of upper and lower interval values (or a single interval pair) in criterion 'crit' from a [0,1] scale to the determined user scale. Note that this call should be used instead of two separate value calls since this function can handle reverse scales.

Convert to autoscale norm vector

Call syntax: DTL_get_AV_norm_vector(int crit , int type, int size, double av_val[], double v_val[])

```
Return information:
OK    -
ERROR - input error
        frame not loaded
        criterion unknown
```

Call semantics: Get the scale transformation of a vector of results in criterion 'crit' to a [0,1] scale from the determined user scale.

Convert to single autoscale norm value

Call syntax: DTL_get_AV_norm_value(int crit , int type, double av_val, double *v_val)

```
Return information:
OK    -
ERROR - input error
        frame not loaded
        criterion unknown
```

Call semantics: Get the transformation of a single value in criterion 'crit' to a [0,1] scale from the determined user scale.

Convert to autoscale norm intervals

Call syntax: DTL_get_AV_norm_intervals(int crit , int type, int size, double av_lobo[], double av_upbo[], double v_lobo[], double v_upbo[])

Call syntax: DTL_get_AV_norm_interval(int crit , int type, double av_lobo, double av_upbo, double *v_lobo, double *v_upbo)

```
Return information:
OK    -
ERROR - input error
        frame not loaded
        criterion unknown
```

Call semantics: Get the scale transformation of two vectors of upper and lower interval results (or a single interval pair) in criterion 'crit' to a [0,1] scale from the determined user scale.

Check autoscale values

Call syntax: DTL_check_AV_user_values(int crit, int type, int count, ...)
Call syntax: DTL_check_AV_norm_values(int type, int count, ...)

```
Return information:
OK    -
ERROR - input error
        frame not loaded
        criterion unknown
```

Call semantics: Check that the supplied list of 'count' values (max 10, in separate arguments) are within the scale range. This is a variadic function call that accepts a varying number of arguments (indicated by the ellipsis).

EVALUATION COMMANDS

For most evaluation commands, multi-criteria evaluations are invoked by supplying 'crit'=0. Partial evaluations of the multi-criteria weight tree can be invoked by 'crit'<0, where |crit| is the node number to start at. It must be an intermediate node.

Evaluate frame

Call syntax:

```
DTL_evaluate_frame(int crit, int method, int Ai, int Aj, e_matrix e_result)
DTL_evaluate_full(int crit, int method, int Ai, int Aj, e_matrix e_result)
```

```
Method subfields:
Eval:    0 Delta
         4 Gamma
         8 Psi
        12 Digamma

Expand:  0 Default
     0x040 Matrix converging to 50% pdf
     0x080 Matrix converging to 50% pdf + swap midpoints
     0x0c0 Matrix converging to mass point (same as default)
     0x100 Matrix converging to mass point + interpolation
```

Return information:

```
OK    -
ERROR - input error
        criterion unknown
        alternative unknown
        wrong method
        frame not loaded
```

Call semantics: Evaluate the criterion ‘crit’ of the loaded frame. All alternatives are evaluated using the Delta, Gamma, Psi, or Digamma rule. For the requested alternative(s) ‘Ai’ (and ‘Aj’), the result is stored in ‘e_result’. For DTL_evaluate_full, each result has the form of a matrix {min,mid,max} x {mass-steps}, with values from increasing mass. For DTL_evaluate_frame, only the first step is filled in. The expand subfield is only valid for DTL_evaluate_full in which there are 21 values corresponding to masses of 0%-100% in 5% steps. Interpolation means that the derivative of the first step is aligned with the rest of the steps resulting in nicer and smoother graphs. ‘Aj’ is relevant only for Delta and Digamma evaluations. For Digamma, ‘Aj’ contains a bitmap with the selected alternatives starting with alternative 1 in the lowest bit in the map.

Evaluate all criteria

Call syntax: DTL_evaluate_omega(int Ai, int mode, cr_col o_result, ci_col o_rank)

Evaluate the alternative ‘Ai’ of the loaded frame w.r.t. all criteria one at a time.

```
Mode:    0 Ordering
         1 Olympic ranking
         2 Strict ranking
         3 Group ranking
        +4 Renormalisation
```

```
Return information:
OK    -
ERROR - input error
        alternative unknown
        frame not loaded
        wrong frame type
```

Call semantics: An alternative is evaluated in each criterion by the Omega rule (“part-worth”). The result is stored in ‘o_result’ indexed with criterion number and the rank or order in ‘o_rank’. ‘mode’ is 0 for an ordering, 1 for an olympic ranking, 2 for a strict ranking, and 3 for a group ranking. ‘o_result’ contains each contribution in percent of the entire MC scale (‘mode’+0) or in percent of the omega EV (‘mode’+4). o_result[0] contains the full Omega EV for alternative ‘Ai’ (coinciding with mid for a Psi evaluation). If ‘Ai’ is 0, an average of all alternatives is returned.

Evaluate all criteria at first level

Call syntax: DTL_evaluate_omega1(int Ai, int mode, cr_col o_result, ci_col o_node)

Evaluate the alternative ‘Ai’ of the loaded frame w.r.t. the total contribution from each weight at the first weight tree level.

```
Mode:    0 Percent of MC scale
         4 Renormalisation
```

```
Return information:
OK    -
ERROR - input error
        alternative unknown
        frame not loaded
        wrong frame type
```

Call semantics: An alternative is evaluated in each node at the first weight tree level by the Omega rule (“part-worth”). The result is stored in ‘o_result’ indexed with node number in ‘o_node’. ‘o_result’ contains each contribution in percent of the entire scale (‘mode’=0) or in percent of the omega EV (‘mode’=4). o_result[0] contains the full Omega EV for alternative ‘Ai’ (coinciding with mid for Psi evaluation). If ‘Ai’ is 0, an average of all alternatives is returned.

Weight tornado

Call syntax: DTL_get_W_tornado(int mode, h_matrix t_lobo, h_matrix t_upbo)
Call syntax: DTL_get_W_tornado_alt(int alt, int mode, h_vector t_lobo, h_vector t_upbo)

```
Mode subfield:
Type:    0 Standard evaluation, explicit mass point kept
         1 Explicit mass point removed before calculations
        +2 Belief mass-based instead of expected value-based
```

```
Return information:
OK    -
ERROR - frame not loaded
        input error
        wrong frame type
        alternative unknown
```

Call semantics: The weight sensitivity tornado of all alternatives (first call) or the alternative ‘alt’ (second call) is returned in two matrices (first call) or vectors (second call) ‘t_lobo’ and ‘t_upbo’. If ‘alt’ is negative, the weight tornado for a single criterion node is returned instead. ‘mode’=0 is with the explicit mass point kept and 1 is without an explicit mass point. For each node, the [t_lobo,t_upbo] interval shows how much the mass point shifts when the respective weights are set to their minima and maxima one at a time. Indexing type: A1. NOTE: ‘node’ is the node number in the weight tree.

Probability tornado

Call syntax: DTL_get_P_tornado(int crit, int mode, h_matrix t_lobo, h_matrix t_upbo)

Mode subfield:

```
Type:    0 Standard evaluation, explicit mass point kept
         1 Explicit mass point removed before calculations
        +2 Belief mass-based instead of expected value-based
```

```
Return information:
OK    -
ERROR - frame not loaded
        input error
        criterion unknown
```

Call semantics: The probability sensitivity tornado of the criterion ‘crit’ is returned in two matrices ‘t_lobo’ and ‘t_upbo’ indexed as [alt][node]. ‘mode’ 0 is with the explicit mass point kept and 1 is without an explicit mass point. Adding 2 to ‘mode’ yields belief mass-based evaluation instead of expected value-based which takes some more CPU power. For each node, the [t_lobo,t_upbo] interval shows how much the mass point shifts when the respective probabilities are set to their minima and maxima one at a time. Indexing type: A1.

Criteria probability tornado

Call syntax: DTL_get_MCP_tornado(int crit, int mode, h_matrix t_lobo, h_matrix t_upbo)

```
Mode subfield:
Type:    0 Standard evaluation, explicit mass point kept
         1 Explicit mass point removed before calculations
        +2 Belief mass-based instead of expected value-based
```

```
Return information:
OK    -
ERROR - frame not loaded
        input error
        wrong frame type
        criterion unknown
```

Call semantics: The criterion-weighted probability tornado of the criterion ‘crit’ is returned in two matrices ‘t_lobo’ and ‘t_upbo’ indexed as [alt][node]. ‘mode’ 0 is with the explicit mass point kept and 1 is without an explicit mass point. Adding 2 to ‘mode’ yields belief mass-based evaluation instead of expected value-based which takes some more CPU power. For each final consequence node, the [t_lobo,t_upbo] interval shows how much the mass point shifts when the respective values are set to their minima and maxima one at a time and how much this influences the total weighted expected value. Indexing type: A1.

Value tornado

Call syntax: DTL_get_V_tornado(int crit, int mode, h_matrix t_lobo, h_matrix t_upbo)

```
Mode subfield:
Type:    0 Standard evaluation, explicit mass point kept
         1 Explicit mass point removed before calculations
        +2 Belief mass-based instead of expected value-based
```

```
Return information:
OK    -
ERROR - frame not loaded
        input error
        criterion unknown
```

Call semantics: The value sensitivity tornado of the criterion ‘crit’ is returned in two matrices ‘t_lobo’ and ‘t_upbo’ indexed as [alt][node]. ‘mode’ 0 is with the explicit mass point kept and 1 is without an explicit mass point. Adding 2 to ‘mode’ yields belief mass-based evaluation instead of expected value-based which takes some more CPU power. For each final consequence node, the [t_lobo,t_upbo] interval shows how much the mass point shifts when the respective values are set to their minima and maxima one at a time. Indexing type: A1.

Criteria value tornado

Call syntax: DTL_get_MCV_tornado(int crit, int mode, h_matrix t_lobo, h_matrix t_upbo)

```
Mode subfield:
Type:    0 Standard evaluation, explicit mass point kept
         1 Explicit mass point removed before calculations
        +2 Belief mass-based instead of expected value-based
```

```
Return information:
OK    -
ERROR - frame not loaded
        input error
        wrong frame type
        criterion unknown
```

Call semantics: The criterion weighted value tornado of the criterion ‘crit’ is returned in two matrices ‘t_lobo’ and ‘t_upbo’ indexed as [alt][node]. ‘mode’ 0 is with the explicit mass point kept and 1 is without an explicit mass point. Adding 2 to ‘mode’ yields belief mass-based evaluation instead of expected value-based which takes some more CPU power. For each final consequence node, the [t_lobo,t_upbo] interval shows how much the mass point shifts when the respective values are set to their minima and maxima one at a time and how much this influences the total weighted expected value. Indexing type: A1.

> Note: In the highly unusual case of split midpoints, i.e. when the midpoint box contains different upper and lower entries, weight and probability tornados will not always be consistent. Using split midpoints is not encouraged, but if they are being used, refrain from using even-mode tornado functions on such frames.

Consequence influence

Call syntax: DTL_get_cons_influence(int crit, int mode, h_matrix result)

```
Mode:    0 Local EV contribution
```

```
         1 Global WEV contribution
```

```
Return information:
OK    -
ERROR - frame not loaded
        input error
        criterion unknown
```

Call semantics: The influence of the consequences of the criterion ‘crit’ is returned in the matrix ‘result’ indexed as [alt][node]. ‘mode’ is 0 for a local result (i.e. within the criterion) and 1 for a global result (i.e. contribution from the criterion to the weighted expected value). For each final consequence node, the value shows how much the mass point of this particular consequence influences the (weighted) expected value. Indexing type: A1.

Compare alternatives

Call syntax: DTL_compare_alternatives(int crit, int method, double belief_level, ar_col lo_value, ar_col up_value)

```
Method subfield:
Eval:    4 Gamma
         8 Psi
```

```
Return information:
OK    -
ERROR - frame not loaded
        input error
        criterion unknown
```

Call semantics: Compares alternatives based on ‘method’ for the criterion ‘crit’. The comparison is made using belief mass. The desired belief level in the range [0,1] must reside in ‘belief_level’ when calling the function. The result is a support range [lo_value[Ai],up_value[Ai]] for each alternative Ai (from 1 to n_alts).

Mass difference between alternatives

Call syntax: DTL_delta_mass(int crit, int mode, ar_matrix delta_value, ar_matrix delta_mass)

```
Interpolation modes:
 0: Standard (raw) mass matrix = no interpolation
 1: In upper triangle: (i) no mass row may decrease going to the right &
    (ii) no mass column may increase going down (row prioritisation)
 2: In upper triangle: (i) no mass column may increase going down &
    (ii) no mass row may decrease going to the right (column prioritisation)
 3: In upper triangle: no mass row may decrease going to the right (row only)
-1: In upper triangle: no mass column may increase going down (column only)
(In lower triangle: the opposite way around because [Aj,Ai] mirrors [Ai,Aj])
```

```
Return information:
OK    -
ERROR - frame not loaded
```

```
        input error
        criterion unknown
```

Call semantics: Returns a matrix ‘delta_value’ with the Delta values and a matrix ‘delta_mass’ with cdf mass of the Deltas (differences) for each pair [Ai,Aj] of alternatives.

Rank alternatives

Call syntax: DTL_rank_alternatives(int crit, int mode, double gamma_tolerance, double omega_tolerance, ai_col gamma_rank, ai_col omega_rank, ar_col gamma_value, ar_col omega_value)

```
Mode:  -3 Delta dominance, hard/strict ranking
       -2 Psi support level, hard/strict ranking
       -1 Gamma cdf, hard/strict ranking
        0 Gamma EV, olympic ranking
        1 Gamma EV, hard/strict ranking
        2 Gamma EV, hard/strict ranking with tiebreaker
        3 Gamma EV, group ranking
```

```
Return information:
OK    - ok
        differing ranks
ERROR - frame not loaded
        input error
        criterion unknown
```

Call semantics: Obtains the ordinal and cardinal rankings (from 1 to n_alts) of all alternatives based on (i) Omega values (mass points) and on (ii) the Gamma evaluations for the criterion ‘crit’ for ‘mode’≥0. For ‘mode’<0 see the mode list above. The cardinal ranking vectors (range: [0,1]) that the ordinal rankings (range: [1..n]) are based on are returned. The function returns DTL_DIFFERING_RANKS if the two ordinal ranking vectors are not identical. The closeness tolerances must be in the range [0.0,0.1] (0.0 for sharp ordinal ranking).

Daisy chain

Call syntax: DTL_daisy_chain2(int crit, int mode, double radius, ai_col omega_rank, ar_col daisy_value, ar_col omega_value)

The original (classic) calls - for simplicity and compatibility
DTL_daisy_chain: no parameters
DTL_daisy_chain1: only mode parameter

Call syntax: DTL_daisy_chain(int crit, ai_col omega_rank, ar_col daisy_value, ar_col omega_value)

Call syntax: DTL_daisy_chain1(int crit, int mode, ai_col omega_rank, ar_col daisy_value, ar_col omega_value)

```
Mode:   0 Return absolute omega EV values (default)
        1 Return relative omega EV values
       +2 Mix belief mass and omega EV within the radius
```

```
Return information:
OK    -
ERROR - frame not loaded
        input error
        criterion unknown
```

Call semantics: Obtains the ordinal and daisy chain (dominance-based) rankings (from 1 to n_alts) of all alternatives based on (i) Omega values (mass points) and on (ii) the pairwise dominance of the alternatives as ranked by the Omega function. The outer mix cut point is called the radius. It can range between 0.0 and 0.5, where 0.0 means no mixing and 0.5 means mixing Deltas being up to half the value scale apart. Larger radii than that are unreasonable.

Pie chart

Call syntax: DTL_pie_chart2(int crit, int mode, double moderation1, double moderation2, ar_col pie_value)
Call syntax: DTL_pie_chart1(int crit, double moderation, ar_col pie_value)
Call syntax: DTL_pie_chart(int crit, ar_col pie_value)

```
The original (classic) calls - for compatibility and simplicity:
DTL_pie_chart:  no parameters + compatibility algorithm
DTL_pie_chart1: one combined moderation parameter
  Positive moderation modifies the daisy chain as a basis for the chart
  Negative moderation modifies the starting point (anchor) of the pie chart
```

```
Method subfield:
Mode:   0 Compatibility mode (older algorithm – not in use anymore)
        1 Modern default mode
       +2 Mix belief mass and EV within the radius
```

Parameters for ‘mode’=1 (ineffectual for ‘mode’=0):
Moderation1 controls how much of its mass the best alternative distributes along the daisy chain. 0.0 means keep all (default), 1.0 is maximum effect.
Moderation2 controls how much of their mass the other alternatives distribute along the daisy chain. 0.0 means keep all (default), 1.0 is maximum effect.

```
Return information:
OK    -
ERROR - frame not loaded
        input error
        criterion unknown
```

Call semantics Obtains the cardinal ranking (from 1 to n_alts) of all alternatives based on the mass distribution of Gamma evaluations for the criterion ‘crit’. The ranking is a relative (proportional) ranking intended for e.g. pie charts. The elements in the ranking sum up to 100%. To obtain unnormalised scores with belief mass, use calls to either DTL_daisy_chain or DTL_rank_alternatives instead. The cardinal ranking vector (range: [0,1]) is returned.

Remaining mass at result level

Call syntax: DTL_get_mass_above(int crit, double lo_level, double *mass)
Call syntax: DTL_get_mass_below(int crit, double up_level, double *mass)
Call syntax: DTL_get_mass_range(int crit, double lo_level, double up_level, double *mass)

```
Return information:
OK    -
ERROR - output error
        input error
        frame not loaded
        criterion unknown
```

Call semantics: Obtains the fraction [0,1] of the mass remaining above/below a specific result level in the evaluation result of the criterion ‘crit’ (or between the given levels in case of DTL_get_mass_range. ‘lo/up_level’ must be in the range [-1,1] for evaluations Delta, Gamma, or Digamma, and [0,1] for Psi. The fraction is returned in ‘mass’. The call must be preceded by an evaluation. This can be seen as the remaining mass “above/below” a specified result level (or both for DTL_get_mass_range) in a traditional evaluation. In that sense, it works perpendicularly to the other mass calls. A two-sided range is obtained either by using DTL_get_mass_range or by two calls to the above or below function with the respective interval endpoints and subtracting the results.

Belief density at result level

Call syntax: DTL_get_mass_density(int crit, double ev_level, double *density)

```
Return information:
OK    -
ERROR - output error
        input error
        frame not loaded
        criterion unknown
```

Call semantics: Obtains the density of belief for a specific result level in the evaluation result of the criterion ‘crit’. ‘ev_level’ must be in the range
[-1,1] for evaluations Delta and Gamma, and [0,1] for Psi. The belief density is returned in ‘density’, i.e. how much belief there is in this specific result level. The call must be preceded by an evaluation. This can be seen as the rate of decrease in the remaining mass at a specified EV level.

Support level mass

Call syntax: DTL_get_support_mass(int crit, double belief_level, double *lobo, double *upbo)
Call syntax: DTL_get_support_lower(int crit, double belief_level, double *lobo, double *upbo)
Call syntax: DTL_get_support_upper(int crit, double belief_level, double *lobo, double *upbo)

```
Return information:
OK    -
ERROR - output error
```

```
        input error
        frame not loaded
        criterion unknown
```

Call semantics: Obtains the interval [0,1] within which ‘belief_level’ fraction of the mass remaining resides in the evaluation result of the criterion ‘crit’. ‘belief_level’ must be in the range [0.5,0.999]. The calculations are the result of a B-normal evaluation. The interval is returned as [lobo,upbo]. The call must be preceded by an evaluation. This can be seen as the mass supporting the evaluation result (mass point).

Risk aversion value

Call syntax: DTL_get_aversion_value(int crit, double risk_aversion, double *ra_value)

```
Return information:
OK    -
ERROR - output error
        input error
        frame not loaded
        criterion unknown
```

Call semantics: Obtains the extended expected value (the aversion value) which reflects the risk aversion of the caller, and at which a corresponding larger fraction of the mass resides below in the evaluation result of the criterion ‘crit’. ‘risk_aversion’ is a risk avoidance parameter that must be in the range ±[0.0, 3.0/lg(2)]. The parameter 1.0 is recommended and implies that the caller is satisfied with ¾ of the mass being below this value (compared to ½ for the ordinary expected value). A negative parameter implies a risk-prone attitude. The extended expected value is returned in ‘ra_value’. The call must be preceded by an evaluation. The normal expected value would correspond to a risk aversion of zero.

Security levels

Call syntax: DTL_sec_level(int crit, double v_min, s_matrix s_result)

```
Return information:
OK    -
ERROR - frame not loaded
      - input error
      - criterion unknown
```

Call semantics: For a PS- or PM-frame, the security level ‘v_min’ specified in the call is evaluated for the criterion ‘crit’. The result has the form of a matrix containing probabilities that the security level is violated (i.e. that the value of the final outcome is ‘v_min’ or less) for each alternative. Currently, there are three such sets for each alternative: min, mid, and max. They are stored in the matrix s_result[alt][set] where ‘alt’ is the sequence number of the alternative and ‘set’ is min, mid, or max.

A note on belief mass functions

```
Let a,b,c be real numbers in [0,1]
Let s be the lower endpoint of the scale [0,1] (Psi) or [-1,1] (Delta, Gamma)
Let d,e,p be real numbers (points) on the scale [s,1]
Let I(d,e) f(x)dx be the Lebesgue integral from d to e over f(x)
Let dens(x) be a belief density function with I(s,1) dens(x)dx = 1

In theory, the most natural would be a three-way belief function:
a = Belief in interval below point p = I(s,p) dens(x)dx
b = Belief in the point p itself     = I(p,p) dens(x)dx
c = Belief in interval above point p = I(p,1) dens(x)dx

For normal density:
b = 0
a + c = 1
a + b + c = 1

For Dirac density:
b = 1
a + c = 0
a + b + c = 1

But the most efficient implementation is a two-way function:
a = Belief in interval at and below point p
c = Belief in interval at and above point p

For normal density:
a + c = 1

For Dirac density (not at scale endpoints):
a = c = 1/2

For Dirac density (at scale lower endpoint = s):
a = 0
c = 1

For Dirac density (at scale upper endpoint = 1):
a = 1
c = 0

The two-way implementation works perfectly for normal cases but requires special
attention for pointwise masses.

The function DTL_get_support_mass does not know whether it is being called by a
function having s=-1 or s=0, so it will return the following:

For Dirac density (at Delta/Gamma/Digamma scale lower endpoint s=-1):
a = 0
c = 1

For Dirac density (at psi scale lower endpoint s=0):
a = c = 1/2
```

BELIEF DOMINANCE COMMANDS

Pairwise belief dominance

Call syntax: DTL_get_dominance(int crit, int Ai, int Aj, double *cd_value, int *d_order)

```
Return information:
OK    -
ERROR – same alternative
        input error
        frame not loaded
        criterion unknown
```

Call semantics: Obtains the belief dominance between the two alternatives Ai and Aj in the criterion ‘crit’. The type of dominance is indicated in ‘d_order’. The output field ‘d_order’ is zero for no dominance, 1 for a first-order (stronger) dominance, and 2 for a second-order (weaker) dominance. The output ‘cd_value’ shows how much one alternative is superior to the other, in terms of EV. If Ai dominates Aj, the result ‘cd_value’ will be positive, and if v.v. negative. If ‘d_order’ is zero then ‘cd_value’ is undefined. For a definition of belief dominance, see the literature on stochastic dominance which is the same concept.

Belief dominance matrix

Call syntax: DTL_get_dominance_matrix(int crit, double threshold, ai_matrix dominance_mx)

```
Return information:
OK    -
ERROR – input error
        frame not loaded
        criterion unknown
```

Call semantics: Obtains the belief dominance between all alternatives in the criterion ‘crit’. The smallest difference that should be considered a dominance is indicated in ‘threshold’ within the range [0.0, 0.1]. The type of dominance between two alternatives Ai and Aj is indicated in dominance_mx[Ai][Aj]. It is zero for no dominance, 1 for first-order dominance, and 2 for second-order dominance. For a definition of belief dominance, see the literature. NOTE: dominance_mx[Ai][Aj] and dominance_mx[Aj][Ai] cannot both be non-zero at the same time.

Belief non-transitive dominance matrix

Call syntax: DTL_get_nt_dominance_matrix(int crit, double threshold, ai_matrix dominance_mx)

```
Return information:
OK    -
ERROR – input error
        frame not loaded
        criterion unknown
```

Call semantics: Obtains the non-transitive belief dominance between all alternatives in the criterion ‘crit’. The smallest difference that should be considered a dominance is indicated in ‘threshold’ within the range [0.0, 0.1]. By non-transitive it is meant that if Ai dominates Ak and Ak dominates Aj, then the information that Ai dominates Aj is, while true, transitively superfluous and thus excluded in this call (included in the above transitive call). The type of dominance between two alternatives Ai and Aj is indicated in dominance_mx[Ai][Aj]. It is zero for no dominance, 1 for first-order dominance, and 2 for second-order dominance. Note that if, for example, Ai 2-order dominates Ak and Ak 1-order dominates Aj, then the fact that Ai dominates Aj is not transitively superfluous since it deals with different dominance concepts. NOTE: dominance_mx[Ai][Aj] and dominance_mx[Aj][Ai] cannot both be non-zero at the same time.

Belief dominance rank

Call syntax: DTL_get_dominance_rank(int crit, int mode, int strict, double threshold, ai_vector dom_rank)

```
Mode parameter:
Ranking: 0 Group ranking
         1 Olympic ranking
         2 Hard/sharp ranking
```

```
Return information:
OK    -
ERROR – input error
        frame not loaded
        criterion unknown
```

Call semantics: Obtains the ranking of all alternatives in the criterion ‘crit’ based on belief dominance. The smallest difference that should be considered a dominance is indicated in ‘threshold’ within the range [0.0, 0.1]. The parameter ‘strict’ indicates if both first and second-order dominance should be taken into account (strict=FALSE, recommended) or only first-order dominance (strict=TRUE, can yield unintuitive results). The alternative(s) that are not dominated by any other is/are given a ranking number of 1. Those that are only dominated by alternative(s) ranked 1 are given a ranking number of 2, and so on.

Cardinal dominance matrix

Call syntax: DTL_get_cardinal_dominance_matrix(int crit, double threshold, int strict, ar_matrix cardinal_mx)

```
Return information:
OK    -
ERROR – input error
        frame not loaded
        criterion unknown
```

Call semantics: Obtains the cardinal belief dominance between all alternatives in the criterion ‘crit’. This means that it is not only an indication of the existence of dominance but contains information on the strength of the dominance. The smallest difference that should be considered dominant is indicated in 'threshold' within the range [0.001, 0.1]. The strength of the dominance between two alternatives Ai and Aj is indicated in cardinal_mx[Ai][Aj]. It is zero for no dominance and a positive number (0.0, 1.0] for dominance. NOTE: cardinal_mx[Ai][Aj] and cardinal_mx[Aj][Ai] cannot both be positive at the same time.

Absolute criteria dominance matrix

Call syntax: DTL_get_abs_dominance_matrix(double threshold, ai_matrix dominance_mx)

```
Return information:
OK    -
ERROR - input error
        frame not loaded
        criterion unknown
```

Call semantics: Obtains the absolute (unweighted) dominance between all alternatives in the PM-frame. The smallest difference that should be considered a dominance is indicated in 'threshold' within the range [0.0, 0.1]. The type of dominance between two alternatives Ai and Aj is indicated in dominance_mx[Ai][Aj]. It is zero for no dominance, 1 for first-order dominance, and 2 for second-order dominance. This call is for excluding alternatives before evaluation based on dominating inferiority.

MISCELLANEOUS COMMANDS

Library release version

Call syntax: DTL_get_release(string(relstrg))
Call syntax: DTL_get_release_long(string(relstrg))

```
Return information:
OK    -
```

Call semantics: Obtains the release version of UNEDA. The format for the standard version is "**M.F.T**", where **M**=main, **F**=functional, and **T**=technical version numbers. The format for the long version is "**M.F.T** [**dddd**]", where **dddd** is the number of days the library has existed and the rest is as in the standard version.

Library capacity

Call syntax: DTL_get_capacity(string(capstrg))
Call syntax: DTL_get_J_properties(string(J_strg))

```
Return information:
OK    -
```

Call semantics: Obtains the library capacity. Returns a string (or JSON object) "max_frames max_crit max_alt max_nodes max_nopa max_cons max_copa max_stmts" with the maximum number of frames (max_frames), criteria (max_crit), alternatives (max_alt), nodes (max_nodes and max_nopa), consequences (max_cons and max_copa) and statemenes (max_stmts) respectively.

Number of weight statements

Call syntax: DTL_nbr_of_W_stmts()

```
Return information:
OK    - number of weight statements in the current frame
ERROR - frame not loaded
```

Call semantics: Returns the number of weight statements in the currently loaded frame.

Number of probability statements

Call syntax: DTL_nbr_of_P_stmts(int crit)

```
Return information:
OK    - number of probability statements in the current frame
ERROR - frame not loaded
        criterion unknown
```

Call semantics: Returns the number of probability statements of the criterion 'crit' in the currently loaded frame. For 'crit'=0, the total number of probability statements in the frame is returned.

Number of value statements

Call syntax: DTL_nbr_of_V_stmts(int crit)

```
Return information:
OK    - number of value statements in the current frame
ERROR - frame not loaded
        criterion unknown
```

Call semantics: Returns the number of value statements of the criterion 'crit' in the currently loaded frame. For 'crit'=0, the total number of value statements in the frame is returned.

Number of weights

Call syntax: DTL_nbr_of_weights()

```
Return information:
OK    - number of weight nodes in the current frame
ERROR - frame not loaded
```

Call semantics: Returns the number of weight nodes in the currently loaded frame.

Number of criteria

Call syntax: DTL_nbr_of_crit()

```
Return information:
OK    - number of criteria in the current frame
ERROR - frame not loaded
```

Call semantics: Returns the number of criteria in the currently loaded frame.

Number of alternatives

Call syntax: DTL_nbr_of_alts()

```
Return information:
OK    - number of alternatives in the current frame
ERROR - frame not loaded
```

Call semantics: Returns the number of alternatives in the currently loaded frame.

Total number of consequences

Call syntax: DTL_total_cons(int crit)

```
Return information:
OK    - number of consequences in the specified alternative
ERROR - frame not loaded
        criterion unknown
```

Call semantics: Returns the total number of consequences in all alternatives of the criterion 'crit' in the currently loaded frame. For 'crit'=-1, the total number of consequences in the frame is returned. Indexing type: B2.

Number of consequences

Call syntax: DTL_nbr_of_cons(int crit, int alt)

```
Return information:
OK    - number of consequences in the specified alternative
ERROR - frame not loaded
        criterion unknown
        alternative unknown
```

Call semantics: Returns the number of consequences in the specified alternative of the criterion 'crit' in the currently loaded frame. Indexing type: B2.

Total number of nodes

Call syntax: DTL_total_nodes(int crit)

```
Return information:
OK    - number of nodes in all alternatives in total
ERROR - frame not loaded
        criterion unknown
```

Call semantics: Returns the total number of nodes in all alternatives of the criterion ‘crit’ in the currently loaded frame (0 for weights). For ‘crit’= -1, the total number of nodes in the frame is returned. Indexing type: B1.

Number of nodes

Call syntax: DTL_nbr_of_nodes(int crit, int alt)

```
Return information:
OK    - number of nodes in the specified alternative
ERROR - frame not loaded
        criterion unknown
        alternative unknown
```

Call semantics: Returns the number of nodes in the specified alternative of the criterion ‘crit’ in the currently loaded frame. Indexing type: B1.

ERROR HANDLING

All UNEDA calls (except DTL_get_errtxt) return a number of type *rcode* which serves as an information carrier and error code at the same time. In the event of an error, a negative number is returned. The caller should interpret the error code and take action accordingly. The numbers are found in DTL.h.

Get error text

Call syntax: char *DTL_get_errtxt(int drc)
Call syntax: char* DTL_get_errtxt_p(rcode drc)
Call syntax: DTL_get_errtxt_i(rcode drc, char* str, unsigned* len)
Call syntax: DTL_get_errtxt_i16(rcode drc, char* str, unsigned* len, bool LE)

```
Return information:
OK    - pointer to error text
ERROR – pointer to text "- RCODE OUT OF RANGE -"
```

Call semantics: Returns the text string that corresponds to the supplied DTL error number in C-style, Pascal-style, or in situ (8-bit or 16-bit chars). For the 16-bit call, the Boolean LE indicates little-endian architecture (else big-endian)

Check error code

Call syntax: DTL_error(int drc)

```
Return information:
0 – the return code ‘drc’ contains only information
1 - the return code ‘drc’ contains an error
```

Call semantics: Returns the severity of the return code ‘drc’ supplied. The ‘drc’ code should originate from a previous UNEDA call. The function takes care of both DTL and TCL error codes.

Call syntax: DTL_error2(int drc)

Return information:
0 – the return code 'drc' contains information, output valid
1 – the return code 'drc' contains information, output invalid
2 - the return code 'drc' contains an error

Call semantics: Returns the severity of the return code 'drc' supplied. The 'drc' code should originate from a previous UNEDA call. The function takes care of both DTL and TCL error codes and categorises them as severe (2) or not (1).

Check user-caused error code

Call syntax: DTL_u_error(int drc)

Return information:
0 – the return code 'drc' contains information or user mistake
1 - the return code 'drc' contains an error not caused by a user

Call semantics: Returns the severity of the return code 'drc' supplied. The 'drc' code should originate from a previous UNEDA call. The function takes care of both DTL and TCL error codes.

Call syntax: DTL_u_error2(int drc)

Return information:
0 – the return code 'drc' contains information, output valid
1 – the return code 'drc' contains information or user mistake, output invalid
2 - the return code 'drc' contains an error

Call semantics: Returns the severity of the return code 'drc' supplied. The 'drc' code should originate from a previous UNEDA call. The function takes care of both DTL and TCL error codes and categorises them as severe (2) or not (1).

DTL error codes

DTL_KERNEL_ERROR

The error occurred in the TCL layer. This value is not returned alone but instead added to the TCL error code.

DTL_INPUT_ERROR

One of the input parameters contained invalid information.

DTL_TREE_ERROR

The tree structure supplied is invalid or the tree description contained invalid information.

DTL_OUTPUT_ERROR

The requested output from the DTL function could not be generated. This usually refers to a request for impossible evaluation data.

DTL_FRAME_EXISTS

The frame number already exists. No more frames can have the same number.

DTL_FRAME_UNKNOWN

The requested frame number does not exist. Either it is not created, or the number is out of range.

DTL_FRAME_IN_USE

An attempt to delete or in another way eliminate a frame that is currently attached (loaded).

DTL_FRAME_NOT_LOADED

An attempt to use frame commands while no frame is loaded.

DTL_FRAME_CORRUPT

Internal error. The frame has been rendered corrupt, either by modifications outside of TCL or because of an internal error in TCL.

DTL_WRONG_FRAME_TYPE

An attempt to issue a PS/PM-only command to a DM frame or vice versa.

DTL_WRONG_STATEMENT_TYPE

The user statement passed in the call is inappropriate for the type of frame currently loaded.

DTL_CONS_OVERFLOW

Too many consequences in the problem for DTL to handle. This should be prohibited in the user interface at an earlier point (use MAX_CONS).

DTL_CRIT_OVERFLOW

Too many criteria in the problem for DTL to handle. This should be prohibited in the user interface at an earlier point (use MAX_CRIT).

DTL_ALT_OVERFLOW

Too many alternatives in the problem for DTL to handle. This should be prohibited in the user interface at an earlier point (use MAX_ALT).

DTL_NODE_OVERFLOW

Too many nodes in the tree for DTL to handle. This should be prohibited in the user interface at an earlier point (use MAX_NODES).

DTL_DIFFERING_RANKS

The rankings obtained with Omega values (midpoint) and Gamma values are not the same. The results are correct but not in accordance with each other.

DTL_SCALE_CHANGE

The automatic scale has changed due to a new value box being loaded. A new value midpoint box must be loaded using DTL_set_AV_mbox.

DTL_SYS_CORRUPT

The internal data structures of DTL or TCL are misaligned.

DTL_STATE_ERROR

A call to DTL is made when DTL is in the wrong initialisation state.

DTL_CRIT_UNKNOWN

The requested criterion does not exist. The criterion number is within the valid range, but no criterion has been installed at this position.

DTL_CRIT_EXISTS

The requested criterion does already exist. A criterion has been installed at this position.

DTL_ALT_UNKNOWN

The alternative does not exist.

DTL_ALT_MISMATCH

The added criterion does not have the same number of alternatives as the frame.

DTL_NAME_MISSING

The frame has not been given a name pointer.

DTL_NAME_TOO_LONG

The frame name has too many characters.

DTL_NAME_EXISTS

The frame name exists already in another frame.

DTL_STMT_ERROR

Syntax error in the input statement.

DTL_WRONG_METHOD

The method field contains an illegal value.

DTL_WRONG_TOLERANCE

The tolerance in the call is not within range.

DTL_CRIT_MISSING

A criterion is missing in a PM-frame and stand-in evaluation is not allowed.

DTL_TOO_FEW_ALTS

Too few alternatives were specified in the call.

DTL_INCONSISTENT

The supplied statement is inconsistent.

DTL_NOT_ALLOWED

The call is not allowed at this time.

DTL_FILE_UNKNOWN

The supplied filename is not a file in the current folder.

DTL_WEAK_MASS_DISTR

Due to skew in the belief mass, the distributions are compressed.

DTL_USER_ABORT

The call was prematurely aborted by the user. No call results are available.

DTL_BUSY

Two threads have called UNEDA in parallel. Since the code is not re-entrant, his thread has to wait for the first to finish. Guard against mix-ups of threads in the calling application.

DTL_LOGFILE_ERROR

Unable to open or write to the call sequence trace log file.

DTL_MEMORY_LEAK

At reconciliation time, allocated memory still remains in use even though it should all be freed. Internal error in DTL.

DTL_BUFFER_OVERRUN

The string supplied was too short for the data returned.

DTL error numbers

DTL_KERNEL_ERROR	-100
DTL_INPUT_ERROR	-101
DTL_TREE_ERROR	-102
DTL_OUTPUT_ERROR	-103
DTL_FRAME_EXISTS	-104

DTL_FRAME_UNKNOWN	-105
DTL_FRAME_IN_USE	-106
DTL_FRAME_NOT_LOADED	-107
DTL_FRAME_CORRUPT	-108
DTL_WRONG_FRAME_TYPE	-109
DTL_WRONG_STATEMENT_TYPE	-110
DTL_CONS_OVERFLOW	-111
DTL_CRIT_OVERFLOW	-112
DTL_LOGFILE_ERROR	-113
DTL_INCONSISTENT	-114
DTL_DIFFERING_RANKS	-115
DTL_STMT_ERROR	-116
DTL_SYS_CORRUPT	-117
DTL_ALT_OVERFLOW	-118
DTL_NODE_OVERFLOW	-119
DTL_CRIT_MISSING	-120
DTL_TOO_FEW_ALTS	-121
DTL_USER_ABORT	-122
DTL_STATE_ERROR	-123
DTL_CRIT_UNKNOWN	-124
DTL_CRIT_EXISTS	-125
DTL_ALT_UNKNOWN	-126
DTL_ALT_MISMATCH	-127
DTL_BUSY	-128
DTL_NAME_MISSING	-129
DTL_NAME_TOO_LONG	-130
DTL_NAME_EXISTS	-131
DTL_NOT_ALLOWED	-132
DTL_WRONG_METHOD	-133
DTL_WRONG_TOLERANCE	-134
DTL_FILE_UNKNOWN	-135
DTL_SCALE_CHANGE	-136
DTL_INTERNAL_ERROR	-137
DTL_WEAK_MASS_DISTR	-138
DTL_MEMORY_LEAK	-139
DTL_BUFFER_OVERRUN	-140
DTL_ASSERT_FAILED	-141

TCL error codes

In the event of a DTL_KERNEL_ERROR, a problem with the request has been detected in the TCL kernel. TCL reports the error to DTL as a positive number not to interfere with DTL error numbers. DTL records the error and it is passed on to the UNEDA caller as one numerical component in DTL_KERNEL_ERROR. The possible codes are:

TCL_INCONSISTENT

The call results in a previously consistent frame becoming inconsistent. The call has been rolled back, and the frame is in the same state as it was before the call.

TCL_INPUT_ERROR

An input parameter contains illegal data, for example, an index out of range or values not within given intervals.

TCL_TREE_ERROR

The structure of the specified input tree is not a valid tree according to the syntactic requirements.

TCL_ILLEGAL_NODE

An attempt to assign a value to an intermediate node in a tree. (Probabilities and weights are allowed but not values)

TCL_TOO_FEW_ALTS

The call contains too few alternatives. This should be prohibited in the user interface at an earlier point.

TCL_TOO_MANY_ALTS

The call contains too many alternatives. This should be prohibited in the user interface at an earlier point.

TCL_TOO_MANY_CONS

The call contains too many consequences. This should be prohibited in the user interface at an earlier point.

TCL_TOO_MANY_STMTS

The call contains too many statements. This should be prohibited in the user interface at an earlier point.

TCL_TOO_NARROW_STMT

The TCL layer could operate in a mode where, for reasons of speed and stability, intervals of very small size are not allowed. This excludes the use of pointwise statements.

TCL_ATTACHED

An attempt to delete a frame that is currently attached (loaded).

TCL_DETACHED

An attempt to access a frame that is currently detached (unloaded).

TCL_CORRUPTED

The frame or other system resources have been rendered corrupt, either by modifications outside of TCL or because of an internal error in TCL.

TCL_OUT_OF_MEMORY

The kernel has run out of memory. This is the result of allocating too little virtual memory to the application in which the TCL layer is hosted.

TCL_MEMORY_LEAK

Memory was not recycled at garbage collection.

TCL error numbers

TCL_INCONSISTENT	1
TCL_INPUT_ERROR	2
TCL_TREE_ERROR	3
TCL_ILLEGAL_NODE	4
TCL_TOO_MANY_CONS	5
TCL_TOO_MANY_ALTS	6
TCL_TOO_MANY_STMTS	7
TCL_TOO_NARROW_STMT	8
TCL_TOO_FEW_ALTS	9
TCL_CORRUPTED	10
TCL_ATTACHED	11
TCL_DETACHED	12
TCL_OUT_OF_MEMORY	13
TCL_MEMORY_LEAK	14

Mapping of DTL return codes

This is the mapping of DTL return codes to the error interpretation done by DTL_error2 and thus indirectly by all error checks above.

DTL return codes	**Interpretation**	**DTL_error2 value***
DTL_OK	Output result valid	0
DTL_DIFFERING_RANKS		
DTL_WEAK_MASS_DISTR		
DTL_SCALE_CHANGE		
DTL_USER_ABORT	Output result invalid	1
All other return codes	Error occurred	2
TCL return codes		
TCL_TOO_MANY_STMTS	Output result invalid	1
TCL_TOO_MANY_CONS		
All other return codes	Error occurred	2

***** NOTE: Only when the result value is 0 there exists a result from the call. Thus, only after an evaluation call resulting in the value 0 is the result cache filled and subsequent output calls such as belief mass will succeed.

Call sequence trace (log file)

UNEDA contains the ability to create a log file (the call sequence trace log, cst_log). This log file contains all the API calls to UNEDA and enables the possibility to trace how an application works from the outside. It can be configured to log only the calls or alternatively also the results of the calls. It is enabled by storing a file "call_seq.log" in the home directory of the application calling DTL. The first line of text in the file controls the trace level and is shown in parenthesis below. Running under MS Windows, the text must be encoded in ANSI (not UTF-8).

```
  Level 0 (no file or no text): no log file written
  Level 1 ("call_seq.log"):     input data + execution status
  Level 2 (“call_seq_ext.log”): level 1 + output data
  Level 3 (“call_seq_dmp.log”): level 2 + dump TCL core on error
```

For level 2, replacing the first line with “call_seq_exx.log” also turns the error trace on. Similarly, “call_seq_exy.log” turns the error trace on but not the call sequence trace. For level 3, a TCL core dump will be written to the folder “dump” in the home directory of the application calling DTL. If the folder does not exist, it will be created. The filename of the dump will be “XXXX0123ABCD99.DTL” where “XXXX” is the acronym of the function that triggered the error dump (see below), “0123ABCD” is a random number in hex format to allow for several dumps resulting from the same problem, and “99” is the TCL error code in decimal format that triggered the dump to be written (see the list of TCL error numbers above). The core dump can subsequently be read by DTL_read_file since it is in the standard UNEDA format. [The dump functionality is not yet implemented.]

API function acronyms

All API functions that alter the contents in SML or ask for an evaluation of the contents have an acronym that will show up in the cst_log file (if it is enabled) in case of runtime error or single thread violation, or in the system trace file (if cst_log is not enabled).

	System functions
INIT	DTL_init
EXIT	DTL_exit
	File functions
FREAD	DTL_read_frame
FRDDT	DTL_read_ddt_frame
FWRT	DTL_write_frame
	Frame functions
PSF	DTL_new_PS_flat_frame
PST	DTL_new_PS_tree_frame
PMF	DTL_new_DM_flat_frame
PMT	DTL_new_DM_tree_frame
PMT	DTL_new_SM_tree_frame
PMF	DTL_new_PM_flat_frame
PMT	DTL_new_PM_tree_frame
PMCT	DTL_new_PM_crit_tree
LPMC	DTL_load_PM_crit
UPMC	DTL_unload_PM_crit
DPMC	DTL_delete_PM_crit
DISP	DTL_dispose_frame
LOAD	DTL_load_frame
UNL	DTL_unload_frame
	Weight functions
AWS	DTL_add_W_statement
CWS	DTL_change_W_statement
RWS	DTL_replace_W_statement
DWS	DTL_delete_W_statement
AWM	DTL_add_W_mid_statement
DWM	DTL_delete_W_mid_statement
SWB	DTL_set_W_box
SWMB	DTL_set_W_mbox

SWMB	DTL_set_W_mbox1
RWMB	DTL_remove_W_mbox
GWH	DTL_get_W_hull
RSTW	DTL_reset_W_base
	Probability functions
APS	DTL_add_P_statement
CPS	DTL_change_P_statement
RPS	DTL_replace_P_statement
DPS	DTL_delete_P_statement
APM	DTL_add_P_mid_statement
DPM	DTL_delete_P_mid_statement
SPB	DTL_set_P_box
SPMB	DTL_set_P_mbox
SPMB	DTL_set_P_mbox1
RPMB	DTL_remove_P_mbox
GPH	DTL_get_P_hull
RSTP	DTL_reset_P_base
	Value functions
AVS	DTL_add_V_statement
CVS	DTL_change_V_statement
RVS	DTL_replace_V_statement
DVS	DTL_delete_V_statement
AVM	DTL_add_V_mid_statement
DVM	DTL_delete_V_mid_statement
SVB	DTL_set_V_box
SVMB	DTL_set_V_mbox
SVMB	DTL_set_V_mbox
SVM	DTL_set_V_modal
RVMB	DTL_remove_V_mbox
GVH	DTL_get_V_hull
GVM	DTL_get_V_modal
CVMOD	DTL_check_V_modality
RSTV	DTL_reset_V_base
	Evaluation functions
EVAL	DTL_evaluate_frame
EVAL	DTL_evaluate_full
OMEGA	DTL_evaluate_omega
OMEGA1	DTL_evaluate_omega1/2
COMP	DTL_compare_alternatives
DMASS	DTL_delta_mass
RANK	DTL_rank_alternatives
DAISY	DTL_daisy_chain/1/2
DAISY	DTL_pie_chart/1/2
AVERS	DTL_get_aversion_value
EVARP	DTL_evaluate_rpf
EVARP	DTL_eval_basic_rpf
GDOM	DTL_get_dominance
GDOMX	DTL_get_dominance_matrix
GDOMX	DTL_get_dominance_nt_matrix
GDOMX	DTL_get_dominance_rank
GCDOMX	DTL_get_cardinal_dominance_matrix
TOW	DTL_get_W_tornado
TOWA	DTL_get_W_tornado_alt
TOP	DTL_get_P_tornado

TMCP	DTL_get_MCP_tornado
TOV	DTL_get_V_tornado
TMCV	DTL_get_MCV_tornado
BTP	DTL_get_BTP_tornado
BTV	DTL_get_BTV_tornado
CINF	DTL_get_cons_influence
	Belief mass functions
AMASS	DTL_get_mass_above
BMASS	DTL_get_mass_below
RMASS	DTL_get_mass_range
SMASS	DTL_get_support_mass
SMASL	DTL_get_support_lower
SMASU	DTL_get_support_upper

CONFIGURATION

The package can run as a server-side process or as a client. Moreover, it can run on Windows, Unix/Linux, and macOS operating systems with only minor modifications. There are also several configuration parameters that control how the package is built during compile time. The default values are indicated with asterisks (*).

UNEDA-DTL configuration options

CALC_SKEW

Enables the evaluation of skew-normal distributions.

Module: DTLeval.c

Values: OFF = Do not allow the evaluation skew-normal distributions
ON = Allow the evaluation of skew-normal distributions *

V_MODAL_RANGE

Enables the check that only physically sound distributions are allowed.

Module: DTLvbase.c

Values: OFF = Allow non-physical (non-modal) value distributions *
ON = Do not allow non-physical value distributions

UNEDA test options

The following is a list of configuration options for testing. All testing options reside in the respective module files. Make sure the options are off if the application has no console window.

V_CRIT0, WARN_MIDPT, WARN_MIDPT_EXT, WARN_MC

Enables printing file content errors to the console. Undefine if there is no console window or no desire to see the messages.

Module: DTLfile.c

Values: Defined = Print file content errors to console *
        Not defined = Do not print file content errors to console

WARN_MIDPT, WARN_MIDPT_EXT, WARN_VSCALE

Enables printing file content errors to the console. Undefine if there is no console window.

Module: DTLfile2.c

Values: Defined = Print file content errors to console *
        Not defined = Do not print file content errors to console

TRACE_BT

Activates tracing binary tree tornado calculations.

Module: DTLtornado.c

Values: Defined = Print binary tree tornado calculations to console
        Not defined = Do not print binary tree calculations to console *

TRACE_MP

Activates tracing tornado mass calculations.

Module: DTLtornado.c

Values: Defined = Print tornado mass calculations to console
        Not defined = Do not print tornado mass calculations to console *

UNEDA-TCL configuration options

NO_ZERO_INTERVALS

Activates blocking intervals of width zero in the bases.

Module: TCLpbase.c, TCLvbase.c

Values: Defined = Block intervals of width zero
        Not defined = Do not block intervals of width zero *

TRACE_MP

Activates tracing mass point generation (centroid).

Module: TCLpbase.c

```
Values: Defined = Print mass point generation
        Not defined = Do not print mass point generation *
```

TRACE_MOMENTS, TRACE_MOMENTS_MC

Activates tracing generated moments.

Module: TCLmoments.c

```
Values: Defined = Print generated moments to console
        Not defined = Do not print generated moments to console *
```

TRACE_MOMCALC, TRACE_MOMCALC_MC

Activates tracing moment calculations.

Module: TCLmoments.c

```
Values: Defined = Print calculation of moments to console
        Not defined = Do not print calculation of moments to console *
```

V_SNAP, V_SNAP_HALF

Controls whether non-physical mean values are corrected or not (or halfway).

Module: TCLmoments.c

```
Values: Defined = correct non-physical mean values (in full or halfway) *
        Not defined = Do not correct non-physical mean values
```

DEVELOPER'S TEST INTERFACE

In addition to the UNEDA Application Programmer's Interface (API), there is also a DTL Developer's Test Interface (DTI). The DTI consists of several calls that are intended for the development and testing of applications rather than being used when the product is finished. The interface provides access to internal data in DTL for inspection and for tallying the calling application. The calls of the DTI come in two categories: i) Fully developed access calls complete with error handling and logging. They return results in call parameters as normal UNEDA calls; and ii) Simpler calls with basic error handling. The latter use console output to return information. In addition, there are also some standard UNEDA calls that can accept parameters only intended for development and testing. There is further a get-to-know package for new callers where call stack, parameter transfer back and forth, and basic logging (files and folders) are being exercised. This latter package is called by a new user with the personal aid of someone familiar with UNEDA and will thus not require the same level of documentation.

Complete access calls

The complete access calls contain the same level of input checks and error handling as ordinary UNEDA calls. They also use the same calling conventions and share result codes with standard calls (see the section on return codes above). The complete access calls are of three types: general calls, base calls, and moment calls.

General DTI functions

Call syntax: DTI_node2crit(int node)
Call syntax: DTI_crit2node(int crit)

```
Return information:
OK    - index number > 0
ERROR - frame not loaded
        wrong frame type
        criterion unknown
        0 if node is not an end/final node
```

Call semantics: These calls convert between node numbers and criteria numbers in a weight tree. All nodes have a node number, but only end/final nodes have a criteria number. Only in the case of a one-level tree do these numbers coincide. While DTI_crit2node will always yield a node number, DTI_node2crit will return 0 if the node supplied is an intermediate node in the tree and thus does not contain a criterion. These calls do not appear in the call sequence log. NOTE: DTI_crit2node is not yet implemented.

Base DTI functions

Call syntax: DTI_W_node_parents(int node1, int node2)
Call syntax: DTI_P_node_parents(int crit, int alt, int node1, int node2)

```
Return information:
OK    -  0 = same parent
        +1 = different parents
ERROR - -1 = frame not loaded
        -1 = wrong frame type
        -1 = criterion unknown
        -2 = input error
```

Call semantics: This call checks whether two weight or probability nodes have the same parent or not. Returns 0 if the same parent, +1 if different parents, and <0 if don't know due to error.

Moment calculus DTI functions

The moment calculus in UNEDA is mainly taking place down in TCL where the moment generation and moment arithmetic functions reside. Although the DTL calls are adequate for displaying the necessary information and results to users, a developer might at times find it useful to gain access to the inner workings of the moment calculus.

Call syntax: DTI_get_mass_moments(int crit, double *rm1, double *cm2, double *cm3)
Call syntax: DTI_get_psi_moments(int crit, int alt, double *rm1, double *cm2)

Call syntax: DTI_get_bn_params(int crit, double *loc, double *scale, double *alpha)
Call syntax: DTI_get_support_mid(int crit, double *cdf)
Call syntax: DTI_dtl_from_bn_cdf(int crit, double cdf_bn, double *cdf_dtl)
Call syntax: DTI_get_lo_inflexion(int crit, double *lo_lim, double *lo_ifx, double *lo_ifk, double *lo_cdf)
Call syntax: DTI_get_up_inflexion(int crit, double *up_lim, double *up_ifx, double *up_ifk, double *up_cdf)

```
Return information:
OK    -
ERROR - frame not loaded
        criterion unknown
        alternative unknown
        input error
        output error
```

Call semantics: The calls relate to the documentation on the B-normal method and will be explained in oral sessions on request.

Basic access calls

The basic access calls contain a simpler level of input checks and error handling than ordinary UNEDA calls. They also use simpler calling conventions, most often no result codes, and do not appear in the call sequence log. The basic access calls are of two types: general calls and base calls. Both types print to the console window which has to be defined in the application.

Basic general DTI functions

Call syntax: void DTI_list_all_frames()

This function outputs a list of all frames loaded in UNEDA to the console. Assume that four frames are created in frame numbers 3, 9, 22, and 30. Frame number 22 is loaded and contains 14 criteria of which criteria 8, 11, and 13 are shadow criteria. The output format is then the following:

```
Frame 3  exists
Frame 9  exists
Frame 22 exists
Frame 30 exists
Frame 22 is loaded
 Crit 1  exists
 Crit 2  exists
 Crit 3  exists
 Crit 6  exists
 Crit 7  exists
 Crit 9  exists
 Crit 10 exists
```

Call semantics: The existing frames are shown in increasing order followed by the loaded frame (if any) and a display of which criteria exist in the loaded frame if it is of the type PM-frame.

Call syntax: void DTI_tree_structure(int crit)

Call semantics: When there is a tree structure in either a weight base or a probability base, the structure can be shown by this function. For the weight tree, criterion 0 should be supplied in ‘crit’ and for a probability tree, its criterion number should be supplied. Assume a weight tree with 24 criteria and 36 nodes. Then the output could look like the following:

Criteria tree node numbers

```
 1--- 2
 |    3
 4
 5
 6--- 7
 |    8
 9---10
 |   11---12
 |        13
14---15
 |   16
17---18
 |   19---20
 |        21
22---23
 |   24
25---26
 |   27
28---29---30
 |    |   31
 |    |   32
 |   33
34---35
     36
```

Real criteria numbers

```
0--- 1
|    2
3
4
0--- 5
|    6
0--- 7
|    0--- 8
|         9
0---10
|   11
0---12
|    0---13
|        14
0---15
|   16
0---17
|   18
0--- 0---19
|    |   20
|    |   21
|   22
0---23
    24
```

First, the tree is shown with the node numbers in a structured format, and then the tree is again shown but this time with criteria numbers. In the latter case, a 0 denotes an intermediate node that has no criterion attached to it. In this way, the mapping between node numbers and criteria numbers is easily viewed for testing purposes. Note that nodes on the same level appear vertically from the first node at the level and nodes one level down appear to the right with the first one pointed to by dash markers.

Basic W/P/V-base DTI functions

For each of the base types, weight (W), probability (P), and value (V), there are functions to show the contents of the base in three ways. In the descriptions below, replace X with {W,P,V} as appropriate.

Call syntax: void DTI_show_W_base()
Call syntax: void DTI_show_P_base(int crit)
Call syntax: void DTI_show_V_base(int crit)

Call semantics: DTI_show_X_base shows the statements entered into UNEDA with DTL_add_X_statement for X∈{W,P,V}.

Call syntax: void DTI_show_W_box()
Call syntax: void DTI_show_P_box (int crit)
Call syntax: void DTI_show_V_box (int crit)

Call semantics: DTI_show_X_box shows the statements entered into UNEDA with DTL_set_X_box for X∈{W,P,V}.

Call syntax: void DTI_show_W_mbox()
Call syntax: void DTI_show_P_mbox (int crit)
Call syntax: void DTI_show_V_mbox (int crit)

Call semantics: DTI_show_X_mbox shows the statements entered into UNEDA with DTL_add_X_mid_statement and DTL_set_X_mbox for X∈{W,P,V}.

Undocumented DTI functions

There are a number of DTI calls that are intended for special situations involving users with special knowledge of the internals of DTL. These will be described when the needs arise for them.

Call syntax: DTI_set_folder(char *folder, int style)
Call syntax: DTI_reset_folder()
Call syntax: DTI_get_folder(char *folder, unsigned *c_size, int style)
Call syntax: DTI_get_folder16(char *folder, unsigned *c_size, bool LE)

Call syntax: DTI_get_API_type(char* typestrg, unsigned c_size)
Call syntax: DTI_split_DM_frame(int ufnbr)
Call syntax: DTI_is_tree(int crit)
Call syntax: DTI_pure_W_tree()
Call syntax: DTI_crit_exists(int crit)

Call syntax: DTI_real_W_crit(int node)
Call syntax: DTI_nbr_W_midpoints()

```
Call syntax: DTI_real_V_crit(int crit, int alt, int node)

Call syntax: DTI_set_AV_crit_scale(int crit, double v_min, double v_max)
Call syntax: DTI_reset_AV_crit_scale(int crit)
Call syntax: DTI_AV_scale_ratio(int c_from, int c_to, int mode, double *ratio)
Call syntax: DTI_check_AV_values(int crit, int type, int count, ...)
Call syntax: DTI_is_AV_default_scale(int crit)

Call syntax: DTI_get_support_mass(int crit, double belief_level, double *lobo, double *upbo)
Call syntax: DTI_pure_W_tree()
Call syntax: DTI_real_W_crit(int node)
Call syntax: DTI_nbr_W_midpoints()
```