A Concurrent Clean Syntax

Clean program

```
CleanProgram    =  {Module} ;
Module          =  DefModule        |   ImplModule ;
DefModule       =  'DEFINITION' 'MODULE' ModuleId ';' {Def}
                   |   'SYSTEM' 'MODULE' ModuleId ';' {Def} ;
ImplModule      =  ['IMPLEMENTATION'] 'MODULE' ModuleId ';' {Impl} ;
Def             =  Import     |   Type-Block    |   AbsType-Block
                   |   DefRule-Block    |   Macro-Block ;
Impl            =  Import     |   Type-Block    |   ImplRule-Block    |   Macro-Block ;
Import          =  'IMPORT' ModuleId {',' ModuleId} ';'
                   |   'FROM' ModuleId 'IMPORT' Symbol {',' Symbol} ';' ;
Type-Block      =  'TYPE' [TypeRule]+ ;
AbsType-Block   =  'ABSTYPE' [TypeGraph]+ ;
DefRule-Block   =  'RULE' [TypeRule]+ ;
ImplRule-Block  =  'RULE' [StrategyDef] [[TypeRule] Rule]+ ;
Macro-Block     =  'MACRO' [Rule]+ ;
StrategyDef     =  'STRATEGY' Strategy ';' ;
Strategy        =  'FUNCTIONAL' ;
```

Type definitions

```
TypeRule        =  '::' TypeAlt {'|' TypeGraph} ';'     |   '::' TypeAlt {'->' TypeGraph} ';' ;
AbsTypeRule     =  '::' TypeGraph ';' ;
TypeAlt         =  TypeGraph '->' TypeGraph ;
TypeGraph       =  Graph ;
```

Rule definitions

```
Rule            =  [RuleAlt]+ ;
RuleAlt         =  Graph '->' GuardedRhss {',' NodeDef} ';' ;
GuardedRhss     =  GuardedRhs ['->' GuardedRhss]    |   Rhs ;
GuardedRhs      =  Rhs [','] 'IF' Guards ;
Rhs             =  NodeId     |   [NodeId ':'] Node ;
Guards          =  Guards {'&&' Guards}     |   Guards {'||' Guards}
                   |   '(' Guards ')'           |   Rhs ;
Graph           =  [NodeId ':'] Node {',' NodeDef} ;
NodeDef         =  NodeId ':' Node              |   NodeIds ':' ListNode
                   |   NodeIds ':' ApplyNode    |   NodeIds ':' SimpleNode
                   |   NodeIds ':' [Annot] NodeId ;
NodeIds         =  '(' NodeId [',' NodeId]+ ')' ;
Node            =  TupleNode    |   ListNode    |   ApplyNode    |   SimpleNode ;
TupleNode       =  [AnnOrAttr] '(' UnBrackArg [',' UnBrackArg ]+ ')';
ListNode        =  [AnnOrAttr] '[ ]'
                   |   [AnnOrAttr] '[' UnBrackArg {',' UnBrackArg } ['|' UnBrackArg ] ']' ;
ApplyNode       =  [AnnOrAttr] NodeId [Arg]+ ;
SimpleNode      =  [AnnOrAttr] Symbol {Arg} ;
Arg             =  [AnnOrAttr] NodeId    |   [AnnOrAttr] [NodeId ':'] ArgNode ;
ArgNode         =  Symbol    |   '(' Node ')'    |   TupleNode    |   ListNode ;
UnBrackArg      =  [AnnOrAttr] NodeId    |   [AnnOrAttr] [NodeId ':'] Node ;
```

Annotations and attributes

```
AnnOrAttr       =  Annotation Attribute     |   Annotation    |   Attribute ;
Annotation      =  '{' StrictAnnot '}'    |   '{' ProcessAnnot '}'      |   StrictShorthand ;
StrictAnnot     =  '!' ;
ProcessAnnot    =  'I'      |   'P' ['AT' Graph] ;
StrictShorthand =  '!' ;
Attribute       =  UniqueAttribute ;
UniqueAttribute =  'UNQ' ;
```

Symbol and node identifiers

```
Symbol          =  SymbolId       |   BasicType    |   BasicValue   |   PredefinedType ;
BasicType       =  'INT'          |   REAL'        |   'CHAR'       |   'BOOL'
                   |   'STRING'   |   'FILE'       |   'PROCID'     |   'WORLD' ;
BasicValue      =  IntDenot  |  RealDenot |  CharDenot  |  BoolDenot  |  StringDenot ;
PredefinedType  =  '=>' ;
```

Identifiers and denotations

A dot (.) is used for concatenation of characters. Reserved words are not allowed as identifiers.

```
ModuleId        =  LowerCaseChar.{RestChar}    |   UpperCaseChar.{RestChar}
                   |   Class1Char.{RestChar} ;
NodeId          =  LowerCaseChar.{RestChar} ;
SymbolId        =  UpperCaseChar.{RestChar}    |   Class1Char.{RestChar} ;
LowerCaseChar   =  'a' | 'b' | ... | 'z' ;
UpperCaseChar   =  'A' | 'B' | ... | 'Z' ;
Digit           =  '0' | '1' | ... | '9' ;
OctDigit        =  '0' | '1' | ... | '7' ;
RestChar        =  LowerCaseChar |   Class1Char   |   Digit
                   |   UpperCaseChar |   CharDel      |   StringDel ;
CharDel         =  ''' ;
StringDel       =  '"' ;
IntDenot        =  [Sign].[Digit]+ ;
RealDenot       =  [Sign].[Digit]+.'.'.[Digit]+.[Exponent] ;
Exponent        =  'E'.[Sign].[Digit]+ ;
CharDenot       =  CharDel.Char.CharDel ;
Char            =  LowerCaseChar |   Class1Char   |   Digit
                   |   UpperCaseChar |   Class2Char   |   Special ;
BoolDenot          =   'TRUE'                     |   'FALSE' ;
StringDenot     =  StringDel.{Char}.StringDel ;
Sign            =  '+' | '-' ;
Class1Char      =  '@'    |   '#'  |   '$'  |   '%'    |   '^'    |   '&'
                   |   '?'  |   '*'     |   '-'    |   '+'  |   '/'    |   '='
                   |   '<'  |   '>'  |   '_'  |   '.'    |   '`'    |   '"'   ;
Class2Char      =  '|'    |   '!'     |   '('    |   ')'    |   '{'    |   '}'
                   |   '['    |   ']'     |   ':'    |   ';'    |   '~'  |   ','   ;
Special         =  '\n'   |   '\r'  |   '\f'  |   '\b'   |   '\t'  |   '\\'
                   |   '\'.CharDel         |   '\'.StringDel   |   '\'.OctDigit.OctDigit.OctDigit ;
```

Reserved words

'AT'	'FROM'	'PROCID'	'UNQ'
'ABSTYPE'	'FUNCTIONAL'	'REAL'	'->'
'BOOL'	'IF'	'RULE'	'=>'
'CHAR'	'IMPLEMENTATION'	'STRATEGY'	'&&'
'CODE'	'IMPORT'	'STRING'	'||'
'DEFINITION'	'INT'	'SYSTEM'	'<<'
'FALSE'	'MACRO'	'TRUE'	'>>'
'FILE'	'MODULE'	'TYPE'	'=='

B Delta Rules

General library entrance (delta)

```
DEFINITION MODULE delta;
IMPORT  deltaB,   deltaC,   deltaI,    deltaR,   deltaS,   deltaM;
```

Imports all library operations excluding deltaP and the IO-library.

Operations on booleans (deltaB)

```
SYSTEM MODULE deltaB;
RULE
::  NOT   !BOOL           ->  BOOL;      == boolean negation
::  AND   !BOOL   !BOOL   ->  BOOL;      == boolean and
::  OR    !BOOL   !BOOL   ->  BOOL;      == boolean or
::  =B    !BOOL   !BOOL   ->  BOOL;      == boolean equal
::  <>B   !BOOL   !BOOL   ->  BOOL;      == boolean not equal
::  BTOS          !BOOL   ->  STRING;    == BOOL to STRING conversion
```

Operations on characters (deltaC)

```
SYSTEM MODULE deltaC;
RULE
::  =C    !CHAR   !CHAR   ->  BOOL;      == character equal
::  <>C   !CHAR   !CHAR   ->  BOOL;      == character not equal
::  <C    !CHAR   !CHAR   ->  BOOL;      == character less than
::  >C    !CHAR   !CHAR   ->  BOOL;      == character greater than
::  <=C   !CHAR   !CHAR   ->  BOOL;      == character less than or equal
::  >=C   !CHAR   !CHAR   ->  BOOL;      == character greater than or equal
::  CTOI          !CHAR   ->  INT;       == CHAR to ASCII conversion
::  CTOS          !CHAR   ->  STRING;    == CHAR to STRING conversion
```

Operations on integers (deltaI)

```
SYSTEM MODULE deltaI;
RULE
::  +         !INT   !INT   ->  INT;       == integer addition
::  -         !INT   !INT   ->  INT;       == integer subtraction
::  *         !INT   !INT   ->  INT;       == integer multiplication
::  /         !INT   !INT   ->  INT;       == integer division
::  %         !INT   !INT   ->  INT;       == integer modulus
::  ++        !INT          ->  INT;       == integer increment
::  --        !INT          ->  INT;       == integer decrement
::  =         !INT   !INT   ->  BOOL;      == integer equal
::  <>        !INT   !INT   ->  BOOL;      == integer not equal
::  <         !INT   !INT   ->  BOOL;      == integer less than
::  >         !INT   !INT   ->  BOOL;      == integer greater than
::  <=        !INT   !INT   ->  BOOL;      == integer less than or equal
::  >=        !INT   !INT   ->  BOOL;      == integer greater than or equal
::  AND%      !INT   !INT   ->  INT;       == integer bitwise and
::  OR%       !INT   !INT   ->  INT;       == integer bitwise or
::  NOT%      !INT          ->  INT;       == integer bitwise not
::  XOR%      !INT   !INT   ->  INT;       == integer exclusive-or
::  SHIFTL%   !INT   !INT   ->  INT;       == integer shift left
::  SHIFTR%   !INT   !INT   ->  INT;       == integer shift right
::  ITOC      !INT          ->  CHAR;      == INT to CHAR conversion
::  ITOR      !INT          ->  REAL;      == INT to REAL conversion
::  ITOS      !INT          ->  STRING;    == INT to STRING conversion
::  RANDOM                  -> INT;        == Random integer
```

Operations on reals (deltaR)

```
SYSTEM MODULE deltaR;
RULE
::  +R        !REAL   !REAL   ->  REAL;     == real addition
::  -R        !REAL   !REAL   ->  REAL;     == real subtraction
::  *R        !REAL   !REAL   ->  REAL;     == real multiplication
::  /R        !REAL   !REAL   ->  REAL;     == real division
::  =R        !REAL   !REAL   ->  BOOL;     == real equal
::  <>R       !REAL   !REAL   ->  BOOL;     == real not equal
::  <R        !REAL   !REAL   ->  BOOL;     == real less than
::  >R        !REAL   !REAL   ->  BOOL;     == real greater than
::  <=R       !REAL   !REAL   ->  BOOL;     == real less than or equal
::  >=R       !REAL   !REAL   ->  BOOL;     == real greater than or equal
::  SIN       !REAL           ->  REAL;     == sine
::  COS       !REAL           ->  REAL;     == cosine
::  TAN       !REAL           ->  REAL;     == tangent
::  ASIN      !REAL           ->  REAL;     == arc sine
::  ACOS      !REAL           ->  REAL;     == arc cosine
::  ATAN      !REAL           ->  REAL;     == arc tangent
::  LN        !REAL           ->  REAL;     == logarithm base e
::  LOG10     !REAL           ->  REAL;     == logarithm base 10
::  EXP       !REAL           ->  REAL;     == raise e to power
::  POW       !REAL   !REAL   ->  REAL;     == raise to power
::  SQRT      !REAL           ->  REAL;     == square root
::  ENTIER    !REAL           ->  INT;      == entier
::  RTOI      !REAL           ->  INT;      == round REAL to INT
::  RTOS      !REAL           ->  STRING;   == REAL to STRING conversion
```

Operations on strings (deltaS)

```
SYSTEM MODULE deltaS;
RULE
::  +S      !STRING  !STRING     ->  STRING;   == concatenate arg2 to arg1
::  =S      !STRING  !STRING     ->  BOOL;     == string equal
::  <>S     !STRING  !STRING     ->  BOOL;     == string not equal
::  <S      !STRING  !STRING     ->  BOOL;     == string less than
::  >S      !STRING  !STRING     ->  BOOL;     == string greater than
::  <=S     !STRING  !STRING     ->  BOOL;     == string less than or equal
::  >=S     !STRING  !STRING     ->  BOOL;     == string greater than or equal
::  INDEX   !STRING !INT         ->  CHAR;     == get the arg2nd char from arg1
::  SLICE   !STRING !INT !INT    ->  STRING;   == get substring arg2 .. arg3 from arg1
::  UPDATE  !STRING !CHAR !INT   -> STRING;    == replace arg3rd char in arg1 with arg3
::  LENGTH  !STRING              ->  INT;      == string length
```

String comparison is based on lexical order. String indices range from 0. Note that as a consequence LENGTH gives the number of characters in the string and not the index of the last character.

Miscellaneous operations (deltaM)

```
SYSTEM MODULE deltaM;
RULE
::  ABORT   !STRING  ->  x ;                   == stop reduction, print arg1
```

Processor location operations (deltaP)

```
SYSTEM MODULE deltaP;
RULE
::  CurrentP                  ->  PROCID;      == proc-id of the current processor
::  ITOP !INT                 ->  PROCID;      == convert an integer to a proc-id
::  RandomP                   ->  PROCID;      == generate a random proc-id
::  NeighbourP !INT !PROCID   ->  PROCID;      == proc-id of n'th neighbour
::  ChannelP x                ->  PROCID;      == returns the proc-id of arg1
```

Version 0.7 operations on files (deltaIO)

```
SYSTEM MODULE deltaIO;
RULE
::  FOpen !STRING !STRING  ->  FILE;   == Open a file in a mode. “r” is read only, “w” is write only
::  FReOpen !FILE          ->  FILE;   == Reopen a file that is already open
```

```
== Input operations
::  FGetC        !FILE          ->  (!CHAR, !FILE);
::  FGetS        !FILE          ->  (!STRING, !FILE);
::  IsEndOfFile  !FILE          ->  BOOL;

== Ouput operations
::  FPutC     !CHAR !FILE      ->  FILE;
::  FPutS     !STRING !FILE    ->  FILE;

== Predefined files
::  StdIn                      -> FILE;      == Standard input (keyboard)
::  StdOut                     -> FILE;      == Standard output (console)
::  StdIO                      -> FILE;      == Standard i/o (keyboard+console)
::  StdErr                     -> FILE;      == Standard error
```

C I/O Library

Operations on files (deltaFile)

```
SYSTEM MODULE deltaFile;
ABSTYPE
::  UNQ FILES;

MACRO            ==    File modes
    FReadText    ->  0;                 == read from a text file
    FWriteText   ->  1;                 == write to a text file
    FAppendText  ->  2;                 == append to an existing text file
    FReadData    ->  3;                 == read from a data file
    FWriteData   ->  4;                 == write to a data file
    FAppendData  ->  5;                 == append to an existing data file

                 ==    Seek modes
    FSeekSet     ->  0;                 == new position is the seek offset
    FSeekCur     ->  1;                 == relative to current position
    FSeekEnd     ->  2;                 == relative to end of file

RULE
::  OpenFiles   !WORLD                ->   (!FILES, !WORLD);
```

Opens the file system (FILES) containing all accessible files. The file system is used as argument for the functions that open and close files (deltaFile.dcl). Attempts to open the file system when it is already open will result in a run-time error.

```
::  CloseFiles  !FILES  !WORLD        ->   WORLD;
```

Closes the file system. It does not close all open files.

```
::  FOpen       !STRING !INT !FILES   ->  (!BOOL, !UNQ FILE, !FILES);
```

Opens a file in a certain mode (read, write or append, text or data). The boolean output parameter reports success or failure. An attempt to open a file that is already open will result in a run-time error.

```
::  FReOpen     !UNQ FILE !INT        ->  (!BOOL, !UNQ FILE);
```

Re-opens an open file in a possibly different mode. The boolean indicates whether the file was successfully closed before reopening.

```
::  FClose      !UNQ FILE !FILES      ->  (!BOOL, !FILES);
```

Closes a file such that it can be opened again with FOpen. The boolean indicates whether the file was succesfully closed.

```
::  FReadC      !UNQ FILE             ->  (!BOOL, !CHAR, !UNQ FILE);
```

Reads a character from a text file or a byte from a data file. The boolean parameter reports success or failure of the operation.

```
::  FReadI      !UNQ FILE             ->  (!BOOL, !INT, !UNQ FILE);
```

Reads an integer from a text file by skipping spaces, tabs and newlines and then reading digits, which may be preceded by a plus or minus sign. From a data file FReadI will just read four bytes (a Clean INT).

```
::  FReadR      !UNQ FILE             ->  (!BOOL, !REAL, !UNQ FILE);
```

Reads a real from a text file by skipping spaces, tabs and newlines and then reading a character representation of a real number. From a data file FReadR will just read eight bytes (a Clean REAL).

```
::  FReadS      !UNQ FILE !INT        ->  (!STRING, !UNQ FILE);
```

Reads n characters from a text or data file, which are returned as a STRING. If the file doesn't contain n characters the file will be read to the end of the file. An empty string is returned if no characters can be read.

```
::  FReadLine   !UNQ FILE             ->  (!STRING, !UNQ FILE);
```

Reads a line from a text file (including a newline character, except for the last line). FReadLine cannot be used on data files.

```
::  FWriteC     !CHAR !UNQ FILE       ->  UNQ FILE;
```

Writes a character to a text file. To a data file FWriteC writes one byte (a Clean CHAR).

:: FWriteI !INT !UNQ FILE -> UNQ FILE;
Writes an integer (its textual representation) to a text file. To a data file FWriteI writes four bytes (a Clean INT).

:: FWriteR !REAL !UNQ FILE -> UNQ FILE;
Writes a real (its textual representation) to a text file. To a data file FWriteR writes eight bytes (a Clean REAL).

:: FWriteS !STRING !UNQ FILE -> UNQ FILE;
Writes a string to a text or data file.

:: FEnd !UNQ FILE -> (!BOOL, !UNQ FILE);
Tests for end-of-file.

:: FError !UNQ FILE -> (!BOOL, !UNQ FILE);
Has an error occurred during previous file I/O operations?

:: FPosition !UNQ FILE -> (!INT, !UNQ FILE);
Returns the current position of the file pointer as an integer. This position can be used later on for the FSeek function.

:: FSeek !UNQ FILE !INT !INT -> (!BOOL, !UNQ FILE);
Move to a different position in the file, first integer argument is the offset, second argument is a seek mode (see above). True is returned if successful.

:: StdIO !FILES -> (!UNQ FILE, !FILES);
Open the 'Console' for reading and writing.

:: StdErr -> !UNQ FILE;
The StdErr file is a write-only file (the 'Errors' file) that need not be opened or closed.

```
:: SFOpen      !STRING !INT !FILES   -> (!BOOL, !FILE,!FILES);
:: SFReadC     !FILE                 -> (!BOOL, !CHAR, !FILE);
:: SFReadI     !FILE                 -> (!BOOL, !INT, !FILE);
:: SFReadR     !FILE                 -> (!BOOL, !REAL, !FILE);
:: SFReadS     !FILE !INT            -> (!STRING, !FILE);
:: SFReadLine  !FILE                 -> (!STRING, !FILE);
:: SFSeek      !FILE !INT !INT       -> (!BOOL, !FILE);
```

With SFOpen a file can be opened for reading more than once without closing it first. On a file opened by SFOpen only the operations beginning with SF can be used. The SF... operations work just like the corresponding F... operations.

:: FShare !UNQ FILE !FILES -> (!FILE, !FILES);
Change a file so that from now it can only be used with SF... operations.

```
:: SFEnd       !FILE                 -> BOOL;
:: SFPosition  !FILE                 -> INT;
```

The functions SFEnd and SFPosition work like FEnd and FPosition, but don't return a new file on which other operations can continue. They can be used for files opened with SFOpen or after FShare, and in guards for files opened with FOpen or FReOpen.

General operations on the IOState (deltaEventIO)

```
DEFINITION MODULE deltaEventIO;
IMPORT deltaIOSystem;
ABSTYPE
::  UNQ EVENTS;
== The event stream.
::  UNQ IOState UNQ s;
== The environment on which all event I/O functions operate.

TYPE
::  InitialIO s     ->  [=> s (=> (IOState s) (s, IOState s))];
== The I/O functions that will be evaluated before starting the interaction.

RULE
::  OpenEvents      !WORLD  ->      (!EVENTS, !WORLD);
```

Retrieves the event stream from the world. Attempts to retrieve the event stream from the world more than once without putting it back result in a run-time error.

:: CloseEvents !EVENTS !WORLD -> WORLD;
Puts the event stream back into the world.

```
::  StartIO    !(IOSystem s (IOState s))    !s   ![InitialIO s]    !EVENTS      ->  (!s, !EVENTS);
```
Starts a new interaction. The initial state of this interaction is composed of the program state (s) and the initial I/O state, which consists of the devices participating in the interaction (defined in the IOSystem argument) and the event stream (EVENTS). Of each device in the initial IOSystem only the first occurrence is taken into account. The program state must be UNQ. Before starting the interaction the InitialIO functions are evaluated. At the end of the interaction StartIO returns the final program state and the resulting event stream, from which the events that have been handled during the interaction have been removed.

```
::  NestIO    !(IOSystem t (IOState t))    !t   ![InitialIO t]    !(IOState s)   ->  (!t, !IOState s);
```
Starts a nested interaction. It replaces the current interaction (as specified by the IOState argument) with a completely new one (as specified by the IOSystem argument). It hides the devices of the current interaction (if any) and fills a new IOState with the devices that are specified in the IOSystem and with the event stream of the old IOState. The program state argument (t) serves as initial program state of the new interaction. Before starting the nested interaction the InitialIO functions are evaluated. NestIO returns the final program state of the new interaction and the original IOState, such that the original interaction re-appears.

```
::  QuitIO    !(IOState s)   ->  IOState s;
```
Closes all devices that are held in the IOState argument. The resulting IOState will cause StartIO or NestIO to terminate. QuitIO is the only function which terminates StartIO or NestIO.

```
::  ChangeIOState   ![=> (IOState s) (IOState s)]    (IOState s)    ->  IOState s;
```
Applies all functions in its first argument in consecutive order to the second (IOState) argument.

Definition of the I/O system (deltaIOSystem)

```
DEFINITION MODULE deltaIOSystem;
FROM deltaPicture IMPORT Picture, Rectangle, DrawFunction;
ABSTYPE
::  DialogInfo;
::  UNQ DialogState UNQ s UNQ io;

TYPE
::  IOSystem UNQ s UNQ io         ->  [DeviceSystem s io];
::  DeviceSystem UNQ s UNQ io     ->  TimerSystem     [TimerDef     s io]
                                  ->  MenuSystem      [MenuDef      s io]
                                  ->  WindowSystem    [WindowDef    s io]
                                  ->  DialogSystem    [DialogDef    s io]
```

The timer device responds only to timer events. A timer event occurs as soon as a certain TimerInterval has expired since the last time it was 'sampled'. A TimerInterval is defined as a number of ticks. A macro TicksPerSecond is defined in deltaTimer.dcl. The timer event causes one of the programmer defined TimerFunctions to be evaluated. The TimerState argument of a TimerFunction indicates how many times the TimerInterval has passed since the last timer event for that timer. When the TimerInterval of one of the timers is smaller than 1 the corresponding TimerFunction is called as often as possible.

```
::  TimerDef UNQ s UNQ io         ->  Timer TimerId SelectState TimerInterval (TimerFunction s io);
::  TimerId                       ->  INT;
::  TimerInterval                 ->  INT;
::  TimerFunction UNQ s UNQ io    ->  => TimerState (=> s (=> io (s, io)));
::  TimerState                    ->  INT;
```

The menu device consists of several PullDownMenus. PullDownMenus are logically grouped into a menu bar, in the same order as they are specified. They are selected by pressing the mouse on their MenuTitle in the menu bar. Menus contain MenuElements. The corresponding MenuFunctions are executed when these elements are selected.

```
::  MenuDef UNQ s UNQ io
    ->  PullDownMenu MenuId MenuTitle SelectState [MenuElement s io];
::  MenuElement UNQ s UNQ io
    ->  MenuItem MenuItemId ItemTitle KeyShortcut SelectState (MenuFunction s io)
    ->  CheckMenuItem MenuItemId ItemTitle KeyShortcut SelectState MarkState (MenuFunction s io)
    ->  SubMenuItem MenuId ItemTitle SelectState [MenuElement s io]
    ->  MenuItemGroup MenuItemGroupId [MenuElement s io]
    ->  MenuRadioItems MenuItemId [RadioElement s io]
    ->  MenuSeparator;
::  RadioElement UNQ s UNQ io
    ->  MenuRadioItem MenuItemId ItemTitle KeyShortcut SelectState (MenuFunction s io);
::  MenuFunction UNQ s UNQ io    ->  => s (=> io (s, io));
::  MenuId                       ->  INT;
```

```
::  MenuTitle                  ->  STRING;
::  MenuItemId                 ->  INT;
::  MenuItemGroupId            ->  INT;
::  KeyShortcut                ->  Key KeyCode | NoKey;
```

The window device consists of several ScrollWindows or FixedWindows. A ScrollWindow is defined by the following arguments:

- WindowId: the number by which the programmer refers to the window.
- WindowPos: the position of the upper-left corner of the window.
- WindowTitle: the title of the window.
- ScrollBarDefs: the horizontal and vertical scrollbars (in that order).
- PictureDomain: the range of the drawing operations in the window.
- MinimumWindowSize: the smallest dimensions of the window.
- InitialWindowSize: the initial dimensions of the window.
- UpdateFunction: the function to redraw parts (UpdateArea) of the window.
- A attribute list that may contain the following window attributes:
  - Activate: the way to respond to activation of the window.
  - Deactivate: the way to respond to de-activation of the window.
  - GoAway: the way to respond when the window is closed.
  - Keyboard: the way the window responds to keyboard input.
  - Mouse: the way the window responds to mouse events.
  - Cursor: the shape of the cursor (mouse pointer) inside the window.
  - StandByWindow: when this attribute is present the window will be a so-called stand-by window.

A FixedWindow has a fixed size, which is defined by its PictureDomain. Therefore it has no scroll bars and no size parameters. When the PictureDomain of a FixedWindow is or becomes greater than one of the screen's dimensions it becomes a ScrollWindow.

```
::  WindowDef UNQ s UNQ io
    -> ScrollWindow  WindowId WindowPos WindowTitle ScrollBarDef ScrollBarDef PictureDomain
                     MinimumWindowSize InitialWindowSize (UpdateFunction s) [WindowAttribute s io]
    -> FixedWindow   WindowId WindowPos WindowTitle PictureDomain (UpdateFunction s)
                                                                          [WindowAttribute s io];

::  WindowId                 ->  INT;
::  WindowPos                ->  (!INT, !INT);
::  WindowTitle              ->  STRING;
::  ScrollBarDef             ->  ScrollBar ThumbValue ScrollValue;
::  ThumbValue               ->  Thumb  INT;
::  ScrollValue              ->  Scroll INT;
::  MinimumWindowSize        ->  (!INT, !INT);
::  InitialWindowSize        ->  (!INT, !INT);
::  UpdateFunction UNQ s     ->  => UpdateArea (=> s (s, [DrawFunction]));
::  UpdateArea               ->  [Rectangle];

::  WindowAttribute UNQ s UNQ io
    ->  Activate      (WindowFunction s io)
    ->  Deactivate    (WindowFunction s io)
    ->  GoAway        (WindowFunction s io)
    ->  Mouse         SelectState (MouseFunction s io)
    ->  Keyboard      SelectState (KeyboardFunction s io)
    ->  Cursor        CursorShape;
    ->  StandByWindow;

::  WindowFunction      UNQ s UNQ io   ->  => s (=> io (s, io));
::  MouseFunction       UNQ s UNQ io   ->  => MouseState (=> s (=> io (s, io)));
::  KeyboardFunction    UNQ s UNQ io   ->  => KeyboardState (=> s (=> io (s, io)));

::  CursorShape     ->  StandardCursor  |  BusyCursor     |  IBeamCursor    |  CrossCursor
                        |  FatCrossCursor  |  ArrowCursor  |  HiddenCursor;
```

The dialog device: Use the Open(Modal)Dialog function (deltaDialog.icl) to open dialogs during the interaction. PropertyDialogs are special dialogs that can only be modeless. They are used to change certain properties (default values, preferences etc.) of a program. They have two predefined buttons: the Set and the Reset button. A CommandDialog can be any kind of dialog. A PropertyDialog is defined by the following attributes:

- DialogId: a number by which the programmer can refer to the dialog.
- DialogTitle: The title of the dialog.
- A list of attributes that may contain the following dialog attributes:
  - DialogPos: The position of the dialog on the screen.
  - DialogSize: The size of the dialog.
  - DialogMargin: The horizontal and vertical margins between the borders and the items.
  - ItemSpace: The horizontal and vertical space between the items of the dialog.

- StandByDialog: When this attribute is present the dialog will be a so-called stand-by dialog.

When none of these attributes is specified the dialog is centered on the screen, a size is chosen such that all items fit in the dialog and save default margins and item spaces are chosen.
- SetFunction/ ResetFunction: The button function for the set/reset button.
- A list of DialogItems: Other items such as CheckBoxes, Control's etc..

A CommandDialog also has an id, a title, a position, a size and a list of DialogItems. Furthermore it has the following attribute:
- DialogItemId: The item id of the default button.

In the AboutDialog information about the application (version, authors etc.) can be presented. It may also contain a button that should provide a help facility. The first AboutDialog that is encountered in the initial dialog device becomes the AboutDialog of the application. Attempts to open AboutDialogs with Open(Modal)Dialog are ignored. The AboutDialog will be accessible by the user during the interaction in a system-dependent way.

```
::  DialogDef UNQ s UNQ io
    -> PropertyDialog    DialogId DialogTitle [DialogAttribute] (SetFunction s io) (ResetFunction s io)
                                                                                    [DialogItem s io]
    -> CommandDialog     DialogId DialogTitle [DialogAttribute] DialogItemId [DialogItem s io]
    -> AboutDialog       ApplicationName PictureDomain [DrawFunction] (AboutHelpDef s io);

::  DialogId            ->  INT;
::  DialogTitle         ->  STRING;
::  DialogAttribute     ->  DialogPos       Measure Measure
                        ->  DialogSize      Measure Measure
                        ->  DialogMargin    Measure Measure
                        ->  ItemSpace       Measure Measure;
                        ->  StandByDialog;
::  Measure             ->  MM REAL  |  Inch REAL  |  Pixel INT;
::  ApplicationName     ->  STRING;
::  AboutHelpDef UNQ s UNQ io
    -> AboutHelp ItemTitle (AboutHelpFunction s io)
    -> NoHelp;
::  AboutHelpFunction UNQ s UNQ io      ->  => s (=> io (s,io));
```

A DialogItem can be a final button (DialogButton), a final button with a user-defined look (DialogIconButton), an unchangeable piece of text (StaticText), a changeable piece of text (DynamicText), an editable text field (EditText), a popup menu (DialogPopUp), a group of RadioButtons, a group of CheckBoxes, or a user-defined Control. The ItemPos specifies the position of the item relative to the other items. When the ItemPos is Left, Center resp. Right the item is placed left-aligned, centered resp. right-aligned, beneath all other items.

```
::  DialogItem UNQ s UNQ io
    -> DialogButton DialogItemId ItemPos ItemTitle SelectState (ButtonFunction s io)
    -> DialogIconButton DialogItemId ItemPos PictureDomain IconLook SelectState
                                                                  (ButtonFunction s io)
    -> StaticText DialogItemId ItemPos STRING
    -> DynamicText DialogItemId ItemPos TextWidth STRING
    -> EditText  DialogItemId ItemPos TextWidth NrEditLines STRING
    -> DialogPopUp DialogItemId ItemPos SelectState DialogItemId [RadioItemDef s io]
    -> RadioButtons DialogItemId ItemPos RowsOrColumns DialogItemId [RadioItemDef s io]
    -> CheckBoxes DialogItemId ItemPos RowsOrColumns [CheckBoxDef s io]
    -> Control DialogItemId ItemPos PictureDomain SelectState ControlState ControlLook
                                                          ControlFeel (DialogFunction s io);

::  DialogItemId        ->  INT;
::  ItemPos             ->  Left | Center | Right | RightTo DialogItemId | Below DialogItemId |
                            XOffset DialogItemId Measure | YOffset DialogItemId Measure |
                            XY Measure Measure | ItemBox INT INT INT INT;
::  IconLook            ->  => SelectState [DrawFunction];
::  TextWidth           ->  Measure;
::  NrEditLines         ->  INT;
::  RowsOrColumns       ->  Rows INT  |  Columns INT;

::  RadioItemDef UNQ s UNQ io
    -> RadioItem        DialogItemId ItemTitle SelectState (DialogFunction s io);
::  CheckBoxDef UNQ s UNQ io
    -> CheckBox         DialogItemId ItemTitle SelectState MarkState (DialogFunction s io);
```

Attributes of a user-defined Control: The ControlState can be a boolean, an integer, a real or a string. The look of the control is defined by the DrawFunctions (see deltaPicture.dcl) returned by the ControlLook function. The ControlFeel defines the way to respond to mouse clicks inside the control's picture domain.

```
::  ControlState  ->  BoolCS BOOL     |     IntCS INT       |     RealCS REAL     |    StringCS STRING
                  |   ListCS [ControlState]                 |     PairCS ControlState ControlState;
```

```
::  ControlLook   ->  => SelectState (=> ControlState [DrawFunction]);
::  ControlFeel   ->  => MouseState (=> ControlState (ControlState,[DrawFunction]));
```

Types of the several dialog item functions:

```
::  SetFunction      UNQ s UNQ io   ->  ButtonFunction s io;
::  ResetFunction    UNQ s UNQ io   ->  ButtonFunction s io;
::  ButtonFunction   UNQ s UNQ io   ->  => DialogInfo (=> s (=> io (s,io)));
::  DialogFunction   UNQ s UNQ io   ->  => DialogInfo (=> (DialogState s io) (DialogState s io));
```

A notice is a simple, modal dialog containing only text and final buttons. It can be used to inform the user bout unusual or dangerous situations. A notice is defined by the following attributes:
- A list of STRINGs: Each string is a line of the message of the notice.
- A NoticeButtonDef: The default button of the notice.
- A list of NoticeButtonDefs: The other buttons of the notice.

```
::  NoticeDef          -> Notice [STRING] NoticeButtonDef [NoticeButtonDef];
::  NoticeButtonDef    -> NoticeButton NoticeButtonId ItemTitle;
::  NoticeButtonId     -> INT;
```

Keyboard input: a window may respond to keyboard events. Each such event causes the KeyboardFunction to be evaluated. For certain special keys constants of type KeyCode are provided in deltaSystem.dcl.

```
::  KeyboardState   ->  (!KeyCode, !KeyState, !Modifiers);
::  KeyCode         ->  CHAR;
::  KeyState        ->  KeyUp   |   KeyDown   |   KeyStillDown;
```

Mouse input: windows and controls may respond to mouse events. Each such event causes the MouseFunction to be evaluated.

```
::  MouseState      ->  (!MousePosition, !ButtonState, !Modifiers);
::  MousePosition   ->  (!INT, !INT);
::  ButtonState     ->  ButtonUp   |   ButtonDown  |   ButtonDoubleDown  |
                        ButtonTripleDown   |   ButtonStillDown;
```

For each modifier or meta-key (Shift, Option (Alternate), Command, Control) a boolean in Modifiers indicates whether it was pressed (TRUE) or not (FALSE). On keyboards that have no Command key both the third and the fourth boolean in Modifiers become TRUE when Control is pressed.

```
::  Modifiers       ->  (!BOOL, !BOOL, !BOOL, !BOOL);
```

Other common types:

```
::  ItemTitle       ->  STRING;
::  SelectState     ->  Able   |   Unable;
::  MarkState       ->  Mark   |   NoMark;
::  PictureDomain   ->  Rectangle;
```

Operations on the timer device (deltaTimer)

```
DEFINITION MODULE deltaTimer;
IMPORT deltaIOSystem, deltaEventIO;
MACRO
    TicksPerSecond      ->  … ;     == system dependent

TYPE
::  CurrentTime  ->  (!INT, !INT, !INT);
    (hours (0-23), minutes (0-59), seconds (0-59))
::  CurrentDate  ->  (!INT, !INT, !INT, !INT);
    (year, month (1-12), day (1-31), day of week (1-7, Sunday=1, Saturday=7))

RULE
::  OpenTimer    !(TimerDef s io)   !(IOState s)   ->  IOState s;
::  CloseTimer   !TimerId           !(IOState s)   ->  IOState s;
    Open (install) a new timer, close (remove) an existing timer.

::  EnableTimer             !TimerId                                 !(IOState s)   ->  IOState s;
::  DisableTimer            !TimerId                                 !(IOState s)   ->  IOState s;
::  ChangeTimerFunction     !TimerId   !(TimerFunction s (IOState s)) !(IOState s)  ->  IOState s;
::  SetTimerInterval        !TimerId   !TimerInterval                !(IOState s)   ->  IOState s;
::  GetTimerBlinkInterval   !(IOState s)   ->  (!TimerInterval, !IOState s);
    Enable, disable, change the TimerFunction and TimerInterval of a Timer. GetTimerBlinkInterval re-
```

turns the number of ticks that should pass between a blink of the cursor (also called caret time).

```
:: Wait   !TimerInterval  x      -> x;
:: UWait  !TimerInterval  UNQ x  -> UNQ x;
```
Wait/UWait delay the evaluation of their second argument for a certain TimerInterval.

```
:: GetCurrentTime  !(IOState s)  ->  (!CurrentTime,  !IOState s);
:: GetCurrentDate  !(IOState s)  ->  (!CurrentDate,  !IOState s);
```
GetCurrentTime / GetCurrentDate return the current time and date.

Operations on menus (deltaMenu)

DEFINITION MODULE **deltaMenu**;
IMPORT deltaIOSystem, deltaEventIO;

Menu operations on unknown MenuIds/MenuItemIds are ignored.

```
RULE
:: EnableMenuSystem       !(IOState s)  ->  IOState s;
:: DisableMenuSystem      !(IOState s)  ->  IOState s;
```
Enable and disable the complete menu system. When the menu system is enabled the previously enabled menus and menu items will become selectable again. Operations on a disabled menu system take effect when the menu system is re-enabled.

```
:: EnableMenus   ![MenuId]   !(IOState s)  ->  IOState s;
:: DisableMenus  ![MenuId]   !(IOState s)  ->  IOState s;
```
Enable, disable menus. Disabling a menu causes its contents to be unselectable. Enabling a disabled menu which contents was partially selectable, causes the previously selectable items to become selectable again.

```
:: InsertMenuItems    !MenuItemGroupId !INT ![MenuElement s (IOState s)]
                                                   !(IOState s)  ->  IOState s;
:: AppendMenuItems    !MenuItemGroupId !INT ![MenuElement s (IOState s)]
                                                   !(IOState s)  ->  IOState s;
:: RemoveMenuItems         ![MenuItemId]           !(IOState s)  ->  IOState s;
:: RemoveMenuGroupItems    !MenuItemGroupId ![INT] !(IOState s)  ->  IOState s;
```
Addition, removal of MenuItems in MenuItemGroups. InsertMenuItems inserts menu items before the item with the specified index, AppendMenuItems inserts them after that item. Items are numbered starting from one. Indices smaller than one or greater than the number of elements cause the elements to be inserted before the first resp. after the last item in the group. Only (Check)MenuItems and MenuSeparators are added to MenuItemGroups. RemoveMenuItems only works on items that are in a MenuItemGroup. RemoveMenuGroupItems removes the items with the specified indices (counting from one) from the specified MenuItemGroup.

```
:: SelectMenuRadioItem  !MenuItemId  !(IOState s)  ->  IOState s;
```
Select a MenuRadioItem: the mark will move from the currently selected item in the group to the item with the specified id.

```
:: EnableMenuItems    ![MenuItemId]                !(IOState s)  ->  IOState s;
:: DisableMenuItems   ![MenuItemId]                !(IOState s)  ->  IOState s;
:: MarkMenuItems      ![MenuItemId]                !(IOState s)  ->  IOState s;
:: UnmarkMenuItems    ![MenuItemId]                !(IOState s)  ->  IOState s;
:: ChangeMenuItemTitles  ![(MenuItemId, STRING)]   !(IOState s)  ->  IOState s;
:: ChangeMenuItemFunctions    ![(MenuItemId, MenuFunction s (IOState s))]
                                                   !(IOState s)  ->  IOState s;
```
Enable, disable, mark, unmark, change titles and functions of MenuElements.

Operations on windows (deltaWindow)

DEFINITION MODULE **deltaWindow**;
IMPORT deltaIOSystem, deltaEventIO, deltaPicture;

Functions applied on non-existing windows or unknown WindowIds are ignored. Functions that operate on the active (or frontmost) window have the same effect as those operating on a (list of) WindowId(s) but are (slightly) faster.

```
RULE
:: OpenWindows       ![WindowDef s (IOState s)]  !(IOState s)  ->  IOState s;
```
The windows are opened in the same order as specified in the list of WindowDefinitions. If one of these windows has the same WindowId as a window that is already open, then this window is not opened. Each new window is always the active window and is placed in front of all existing win-

dows.

```
:: CloseWindows           ![WindowId] !(IOState s)    ->  IOState s;
```
The windows are closed in the same order as specified in the list.

```
:: CloseActiveWindow      !(IOState s)  ->  IOState s;
```
The active window is closed.

```
:: GetActiveWindow        !(IOState s)  ->  (!BOOL, !WindowId, !IOState s);
```
Returns the id of the active window. If there is no active window the boolean will be FALSE.

```
:: ActivateWindow         !WindowId    !(IOState s)  ->  IOState s;
```
Activates the indicated window. No effect, if the window is already active.

```
:: ChangeUpdateFunction      !WindowId    !(UpdateFunction s) !(IOState s)    ->  IOState s;
:: ChangeActiveUpdateFunction             !(UpdateFunction s) !(IOState s)    ->  IOState s;
```
Change the update function of the indicated window.

```
:: ChangeWindowTitle      !WindowId    !WindowTitle !(IOState s)    ->  IOState s;
:: ChangeActiveWindowTitle             !WindowTitle !(IOState s)    ->  IOState s;
```
Change the window title of the indicated window.

```
:: ChangeWindowCursor  !WindowId !CursorShape     !(IOState s)   ->  IOState s;
:: ChangeActiveWindowCursor    !CursorShape       !(IOState s)   ->  IOState s;
```
Change the local cursor shape of a window. When the mouse pointer moves over the content region of the window the cursor will get the indicated shape.

```
TYPE
:: ScrollBarChange
   -> ChangeThumbs     INT    INT        == set new horizontal and vertical thumb values
   |  ChangeScrolls    INT    INT        == set new horizontal and vertical scroll values
   |  ChangeHThumb     INT               == set new horizontal thumb value
   |  ChangeVThumb     INT               == set new vertical thumb value
   |  ChangeHScroll    INT               == set new horizontal scroll value
   |  ChangeVScroll    INT               == set new vertical scroll value
   |  ChangeHBar       INT    INT        == set new horizontal thumb and scroll values
   |  ChangeVBar       INT    INT;       == set new vertical thumb and scroll values

RULE
:: ChangeScrollBar !WindowId !ScrollBarChange !s !(IOState s) -> (!s, !IOState s);
:: ChangeActiveScrollBar !ScrollBarChange !s !(IOState s)    ->  (!s, !IOState s);
```
Change the scrollbar(s) of a window according to the ScrollBarChange.

```
:: ChangePictureDomain !WindowId  !PictureDomain !s !(IOState s)  ->  (!s, !IOState s);
:: ChangeActivePictureDomain      !PictureDomain !s !(IOState s)  ->  (!s, !IOState s);
```
Change the PictureDomain of the indicated window. The settings of the scrollbars are automatically adjusted. Windows will be resized in case the size of the new PictureDomain is smaller than the current size of the window.

```
:: DrawInWindow !WindowId   ![DrawFunction] !(IOState s)    ->  IOState s;
:: DrawInActiveWindow       ![DrawFunction] !(IOState s)    ->  IOState s;
```
Apply the list of DrawFunctions (see deltaPicture.dcl) to the Picture of the window in the given order.

```
:: DrawInWindowFrame !WindowId   !(UpdateFunction s) !s !(IOState s)    ->  (!s, !IOState s);
:: DrawInActiveWindowFrame       !(UpdateFunction s) !s !(IOState s)    ->  (!s,!IOState s);
```
The UpdateFunction has a list of visible rectangles as parameter. Using this list it is possible to return a list of drawing functions that only draw in the visible part of the window.

```
:: WindowGetFrame !WindowId   !(IOState s)   ->  (!PictureDomain, !IOState s);
:: ActiveWindowGetFrame       !(IOState s)   ->  (!PictureDomain, !IOState s);
```
Return the visible part of the Picture of the indicated window in terms of the PictureDomain. Returns ((0, 0), (0, 0)), if the indicated window does not exist.

```
:: EnableKeyboard      !WindowId    !(IOState s)   ->  IOState s;
:: DisableKeyboard     !WindowId    !(IOState s)   ->  IOState s;
:: EnableActiveKeyboard             !(IOState s)   ->  IOState s;
:: DisableActiveKeyboard            !(IOState s)   ->  IOState s;
:: ChangeKeyboardFunction   !WindowId !(KeyboardFunction s (IOState s))
                                                        !(IOState s)   ->  IOState s;
:: ChangeActiveKeyboardFunction   !(KeyboardFunction s (IOState s))  !(IOState s)   ->  IOState s;
```
Enable, disable, change KeyboardFunction of a window.

```
:: EnableMouse      !WindowId     !(IOState s)   ->  IOState s;
:: DisableMouse     !WindowId     !(IOState s)   ->  IOState s;
```

```
::  EnableActiveMouse              !(IOState s)   ->  IOState s;
::  DisableActiveMouse             !(IOState s)   ->  IOState s;
::  ChangeMouseFunction !WindowId      !(MouseFunction s (IOState s)) !(IOState s)   ->  IOState s;
::  ChangeActiveMouseFunction          !(MouseFunction s (IOState s)) !(IOState s)   ->  IOState s;
```

Enable, disable, change MouseFunction of a window.

Operations on dialogs (deltaDialog)

DEFINITION MODULE **deltaDialog**;
IMPORT deltaIOSystem, deltaEventIO;

Functions applied on non-existent dialogs or unknown id's are ignored.

```
TYPE
::  DialogChange s -> => (DialogState s (IOState s)) (DialogState s (IOState s));
```

RULE
OpenDialog opens a PropertyDialog or CommandDialog as a modeless dialog.

```
::  OpenDialog !(DialogDef s (IOState s)) !(IOState s) -> IOState s;
```

OpenModalDialog opens a CommandDialog as a modal dialog. The function terminates when the dialog is closed (by means of Close(Active)Dialog. Attempts to open property dialogs with this function are ignored.

```
::  OpenModalDialog !(DialogDef s (IOState s)) !s !(IOState s) -> (!s, !IOState s);
```

Close(Active)Dialog closes the indicated dialog.

```
::  CloseDialog !DialogId !(IOState s) -> IOState s;
::  CloseActiveDialog !(IOState s) -> IOState s;
```

OpenNotice opens a notice and returns the id of the selected notice button.

```
::  OpenNotice !NoticeDef !(IOState s) -> (!NoticeButtonId, !IOState s);
```

A Beep is the simplest kind of notice.

```
::  Beep !(IOState s) -> IOState s;
```

With the following functions the state of dialog items can be changed. They have no effect when an id is specified of an item for which the state change is invalid.

```
::  EnableDialogItems    ![DialogItemId] !(DialogState s (IOState s))    ->  DialogState s (IOState s);
::  DisableDialogItems   ![DialogItemId] !(DialogState s (IOState s))    ->  DialogState s (IOState s);
::  MarkCheckBoxes       ![DialogItemId] !(DialogState s (IOState s))    ->  DialogState s (IOState s);
::  UnmarkCheckBoxes       ![DialogItemId] !(DialogState s (IOState s))
                                                                         ->  DialogState s (IOState s);
::  SelectDialogRadioItem  !DialogItemId !(DialogState s (IOState s))
                                                                         ->  DialogState s (IOState s);
::  ChangeEditText         !DialogItemId !STRING !(DialogState s (IOState s))
                                                                         ->  DialogState s (IOState s);
::  ChangeDynamicText      !DialogItemId !STRING !(DialogState s (IOState s))
                                                                         ->  DialogState s (IOState s);
::  ChangeIconLook         !DialogItemId !IconLook !(DialogState s (IOState s))
                                                                         ->  DialogState s (IOState s);
```

Functions to change state, look and feel (behaviour) of Controls. When the id is not the id of a Control the functions have no effect.

```
::  ChangeControlState     !DialogItemId !ControlState !(DialogState s (IOState s))
                                                                         ->  DialogState s (IOState s);
::  ChangeControlLook      !DialogItemId !ControlLook !(DialogState s (IOState s))
                                                                         ->  DialogState s (IOState s);
::  ChangeControlFeel      !DialogItemId !ControlFeel !(DialogState s (IOState s))
                                                                         ->  DialogState s (IOState s);
```

Functions to change the functions related to dialog items.

```
::  ChangeDialogFunction   !DialogItemId !(DialogFunction s (IOState s)) !(DialogState s (IOState s))
                                                                         ->  DialogState s (IOState s);
::  ChangeButtonFunction   !DialogItemId !(ButtonFunction s (IOState s)) !(DialogState s (IOState s))
                                                                         ->  DialogState s (IOState s);
```

```
::  ChangeSetFunction      !(SetFunction s (IOState s)) !(DialogState s (IOState s))
                                                         ->  DialogState s (IOState s);
::  ChangeResetFunction    !(ResetFunction s (IOState s)) !(DialogState s (IOState s))
                                                         ->  DialogState s (IOState s);
```

Get(Active)DialogInfo returns the current settings (DialogInfo) of the indicated dialog. The boolean indicates whether the indicated dialog is open. When it is FALSE a dummy DialogInfo is returned.

```
::  GetDialogInfo       !DialogId  !(IOState s)   ->  (!BOOL, !DialogInfo, !IOState s);
::  GetActiveDialogInfo            !(IOState s)   ->  (!BOOL, !DialogInfo, !IOState s);
```

DialogStateGetDialogInfo returns the DialogInfo corresponding to a given DialogState.

```
::  DialogStateGetDialogInfo !(DialogState s (IOState s)) -> (!DialogInfo, !DialogState s (IOState s));
```

The following functions return the current settings of certain dialog items (those that can be changed by the user). When the corresponding item does not exist a run-time error occurs. The id passed to GetSelectedRadioItemId must be the id of a DialogPopUp or a group of RadioButtons. The function CheckBoxesMarked returns the settings of a group of CheckBoxes. The id passed to it must be the id of such a group.

```
::  GetEditText              !DialogItemId   !DialogInfo   ->   STRING;
::  GetSelectedRadioItemId   !DialogItemId   !DialogInfo   ->   DialogItemId;
::  CheckBoxesMarked         !DialogItemId   !DialogInfo   ->   [(DialogItemId, BOOL)];
::  CheckBoxMarked           !DialogItemId   !DialogInfo   ->   BOOL;
::  GetControlState          !DialogItemId   !DialogInfo   ->   ControlState;
```

ChangeDialog can be used to modify open dialogs.

```
::  ChangeDialog     !DialogId ![DialogChange s] !(IOState s)   ->  IOState s;
```

The file selector dialogs (deltaFileSelect)

DEFINITION MODULE **deltaFileSelect**;
IMPORT deltaFile, deltaEventIO;

With the functions defined in this module standard file selector dialogs can be opened, which provide a user-friendly way to select input or output files. The lay-out of these dialogs depends on the operating system.

RULE

```
::  SelectInputFile !FILES !(IOState s)      -> (!BOOL, !STRING, !FILES, !IOState s);
```

Opens a dialog in which the user can traverse the file system to indicate an existing file. The boolean result indicates whether the user has pressed the Open button (TRUE) or the Cancel button (FALSE). The STRING result is the complete path name of the selected file. If the Cancel button has been pressed, an empty string will be returned.

```
::  SelectOutputFile !STRING !STRING !FILES !(IOState s)
                                       ->  (!BOOL, !STRING, !FILES, !IOState s);
```

Opens a dialog in which the user can specify the name of a file that should be created in a certain directory. The first parameter is the prompt of the dialog (default: "Save As:"). The second parameter is the default file name. The boolean result indicates whether the user has pressed the Save button (TRUE) or the Cancel button (FALSE). The STRING result is the complete path name of the selected file. If the Cancel button has been pressed, an empty string will be returned. When a file with the indicated name already exists in the indicated directory a confirm dialog will be opened after pressing Save.

Predefined Controls (deltaControls)

DEFINITION MODULE **deltaControls**;
IMPORT deltaIOSystem, deltaEventIO;

General scrolling list and slider bar definition. These predefined dialog items are implemented entirely in Concurrent Clean as a user-defined Control.

TYPE

```
::  NrVisible          -> INT;
::  SliderDirection    -> Horizontal | Vertical;
::  SliderPos          -> INT;
::  SliderMax          -> INT;
```

RULE
A ScrollingList is defined by the following attributes:
- Id, ItemPos and SelectState (like other dialog items).
- The minimum width of the scrolling list (Measure).
  This attribute is important only when ChangeScrollingList is used. Use a minimum width of zero when the scrolling list is not changed.
- The number of items that is visible in the list (NrVisible).
- The item that is initially selected (ItemTitle).
- The list of items ([ItemTitle]).
- A DialogFunction that is called whenever a new item is selected.

The function ScrollingList returns a DialogItem (a Control) that can be used in any dialog definition.

```
:: ScrollingList !DialogItemId !ItemPos !Measure !SelectState !NrVisible !ItemTitle
      ![ItemTitle] !(DialogFunction s (IOState s))   -> DialogItem s (IOState s);
```

With ChangeScrollingList the items in the scrolling list can be changed. Its arguments are the id of the scrolling list, the newly selected item and the new list of items. When the id is not the id of a ScrollingList a run-time error is generated.

```
:: ChangeScrollingList !DialogItemId !ItemTitle ![ItemTitle]
      !(DialogState s (IOState s))   -> DialogState s (IOState s);
```

GetScrollingListItem returns the currently selected item in the scrolling list with the indicated id from the DialogInfo parameter. When the id is not the id of a ScrollingList a run-time error is generated.

```
:: GetScrollingListItem !DialogItemId !DialogInfo -> ItemTitle;
```

A SliderBar is defined by the following attributes:
- Id, ItemPos and SelectState, like other DialogItems.
- SliderDirection: Horizontal or Vertical.
- SliderPos: the initial position of the slider. This position is always adjusted between 0 and SliderMax.
- SliderMax: the slider can take on positions between 0 and SliderMax.
- A DialogFunction that is called whenever the slider moves.

```
:: SliderBar !DialogItemId !ItemPos !SelectState !SliderDirection !SliderPos
      !SliderMax !(DialogFunction s (IOState s))   -> DialogItem s (IOState s);
```

ChangeSliderBar moves the slider of the indicated bar to the new position. The position is always adjusted between 0 and SliderMax.

```
:: ChangeSliderBar !DialogItemId !SliderPos !(DialogState s (IOState s))
   -> DialogState s (IOState s);
```

GetSliderPosition returns the current slider position of the slider bar with the indicated id from the DialogInfo parameter. When the id is not the id of a SliderBar a run-time error is generated.

```
:: GetSliderPosition !DialogItemId !DialogInfo   -> SliderPos;
```

Global I/O operations (deltaIOState)

DEFINITION MODULE **deltaIOState**;
IMPORT deltaIOSystem, deltaEventIO;
RULE

```
:: SetGlobalCursor     !CursorShape    !(IOState s)   -> IOState s;
```
Sets the shape of the cursor (mouse pointer) globally. This shape overrules the local cursor shapes of windows. Attempts to set the global cursor to HiddenCursor are ignored

```
:: ResetCursor                         !(IOState s)   -> IOState s;
```
Undoes the effect of SetGlobalCursor. It resets the cursor to the standard shape outside windows with a local cursor shape and to the local shape inside such windows.

```
:: ObscureCursor                       !(IOState s)   -> IOState s;
```
Hides the cursor until the next time that the mouse is moved.

```
:: SetDoubleDownDistance   !INT   !(IOState s)   -> IOState s;
```
Set the maximum distance (in pixels) between two mouse clicks such that they will be treated as a ButtonDoubleDown or ButtonTripleDown. Values smaller than zero will be adjusted to zero.

Operations on pictures (deltaPicture)

```
DEFINITION MODULE deltaPicture;
IMPORT deltaFont;
ABSTYPE
::  UNQ Picture;

TYPE
::  DrawFunction        ->  => Picture Picture;
```

The predefined figures that can be drawn:

```
::  Point               ->  (!INT, !INT);
::  Line                ->  (!Point, !Point);
::  Curve               ->  (!Oval, !INT, !INT);
::  Rectangle           ->  (!Point, !Point);
::  RoundRectangle      ->  (!Rectangle, !INT, !INT);
::  Oval                ->  Rectangle;
::  Circle              ->  (!Point, !INT);
::  Wedge               ->  (!Oval, !INT, !INT);
::  Polygon             ->  (!Point, !PolygonShape);
::  PolygonShape        ->  [Vector];
::  Vector              ->  (!INT, !INT);
```

The pen attributes influence the way figures are drawn. The PenMode also influences the way text is drawn. The Not... modes do not work when text is drawn. When the PenMode is a Not... mode text is drawn in OrMode.

```
::  PenSize     ->  (!INT, !INT);
::  PenMode     ->  CopyMode      |  OrMode        |  XorMode     |
                    ClearMode     |  NotCopyMode   |  NotOrMode   |
                    NotXorMode    |  NotClearMode  |  HiliteMode;
::  PenPattern  ->  BlackPattern  |  DkGreyPattern |  GreyPattern |
                    LtGreyPattern |  WhitePattern;
```

The predefined colours:

```
::  Colour ->  RGB REAL REAL REAL          |
               BlackColour  |  WhiteColour  |  BlueColour    |  CyanColour  |
               RedColour    |  GreenColour  |  YellowColour  |  MagentaColour;

MACRO
    MinRGB  ->  0.0;
    MaxRGB  ->  1.0;

RULE
```

Rules setting the attributes of a Picture:

SetPenSize (w, h) sets the PenSize to w pixels wide and h pixels high.
SetPenMode sets the drawing mode of the pen.
SetPenPattern sets the pattern of the pen.
SetPenNormal sets the PenSize to (1,1), the PenMode to CopyMode and the PenPattern to BlackPattern.

```
::  SetPenSize      !PenSize     !Picture  ->  Picture;
::  SetPenMode      !PenMode     !Picture  ->  Picture;
::  SetPenPattern   !PenPattern  !Picture  ->  Picture;
::  SetPenNormal                 !Picture  ->  Picture;
```

Colour: There are two types of Colours: RGB colours and basic colours. An RGB colour defines the amount of red (r), green (g) and blue (b) in a certain colour by the tuple (r,g,b). These are REAL values and each of them must be between MinRGB and MaxRGB (0.0 and 1.0). The colour black is defined by (MinRGB, MinRGB, MinRGB) and white by (MaxRGB, MaxRGB, MaxRGB). Given a RGB colour, all amounts are adjusted between MinRGB and MaxRGB.

SetPenColour sets the colour of the pen.
SetBackColour sets the background colour.

```
::  SetPenColour    !Colour !Picture  ->  Picture;
::  SetBackColour   !Colour !Picture  ->  Picture;
```

Fonts: The initial font of a Picture is the default font (see deltaFont.dcl).

SetFont sets a new complete Font in the Picture.

SetFontName sets a new font without changing the style or size.
SetFontStyle sets a new style without changing font or size.
SetFontSize sets a new size without changing font or style. The size is always adjusted between MinFontSize and MaxFontSize (see deltaFont.dcl).

```
:: SetFont          !Font         !Picture  ->  Picture;
:: SetFontName      !FontName     !Picture  ->  Picture;
:: SetFontStyle     ![FontStyle]  !Picture  ->  Picture;
:: SetFontSize      !FontSize     !Picture  ->  Picture;
```

PictureCharWidth (PictureStringWidth) yield the width of the given CHAR (STRING) given the current font of the Picture. PictureFontMetrics yields the FontInfo of the current font.

```
:: PictureCharWidth     !CHAR    !Picture  ->  (!INT,      !Picture);
:: PictureStringWidth   !STRING  !Picture  ->  (!INT,      !Picture);
:: PictureFontMetrics            !Picture  ->  (!FontInfo, !Picture);
```

Absolute and relative pen move operations without drawing.

```
:: MovePenTo  !Point   !Picture  ->  Picture;
:: MovePen    !Vector  !Picture  ->  Picture;
```

Absolute and relative pen move operations with drawing.

```
:: LinePenTo  !Point   !Picture  ->  Picture;
:: LinePen    !Vector  !Picture  ->  Picture;
```

DrawChar (DrawString) draws a character (string) in the current font. The baseline of the characters is the y co-ordinate of the pen. The new position of the pen is directly after the (last) character (of the string).

```
:: DrawChar    !CHAR    !Picture  ->  Picture;
:: DrawString  !STRING  !Picture  ->  Picture;
```

All following rules do not change the position of the pen after drawing.

Draw(C)Point draws the pixel (in the given colour) in the Picture.
Draw(C)Line draws the line (in the given colour) in the Picture.
Draw(C)Curve draws the curve (in the given colour) in the Picture. A Curve is a part of an Oval o starting from angle a up to angle b (both in degrees modulo 360): (o, a, b). Angles are always taken counterclockwise, starting from 3 o'clock.

```
:: DrawPoint   !Point            !Picture  ->  Picture;
:: DrawLine    !Line             !Picture  ->  Picture;
:: DrawCurve   !Curve            !Picture  ->  Picture;
:: DrawCPoint  !Point  !Colour   !Picture  ->  Picture;
:: DrawCLine   !Line   !Colour   !Picture  ->  Picture;
:: DrawCCurve  !Curve  !Colour   !Picture  ->  Picture;
```

A Rectangle is defined by two diagonal corner points.

DrawRectangle draws the edges of the rectangle.
FillRectangle draws the edges and interior of the rectangle.
EraseRectangle erases the edges and interior of the rectangle.
InvertRectangle inverts the edges and interior of the rectangle.
Copy/MoveRectangleTo copies/moves the contents of the rectangle to a new top-left corner.
Copy/MoveRectangleTo copies/moves the contents of the rectangle over the given vector.

```
:: DrawRectangle     !Rectangle          !Picture  ->  Picture;
:: FillRectangle     !Rectangle          !Picture  ->  Picture;
:: EraseRectangle    !Rectangle          !Picture  ->  Picture;
:: InvertRectangle   !Rectangle          !Picture  ->  Picture;

:: CopyRectangleTo   !Rectangle !Point   !Picture  ->  Picture;
:: CopyRectangle     !Rectangle !Vector  !Picture  ->  Picture;
:: MoveRectangleTo   !Rectangle !Point   !Picture  ->  Picture;
:: MoveRectangle     !Rectangle !Vector  !Picture  ->  Picture;
```

Rounded corner rectangles: a RoundRectangle with enclosing Rectangle r and corner curvatures x and y is defined by the tuple (r, x, y). x and y define the horizontal and vertical diameter of the corner curves. They are always adjusted between 0 and the width (height) of r.

```
:: DrawRoundRectangle  !RoundRectangle  !Picture  ->  Picture;
:: FillRoundRectangle  !RoundRectangle  !Picture  ->  Picture;
```

```
::  EraseRoundRectangle  !RoundRectangle   !Picture   ->  Picture;
::  InvertRoundRectangle !RoundRectangle   !Picture   ->  Picture;
```

An Oval is defined by its enclosing Rectangle.

```
::  DrawOval     !Oval  !Picture   ->  Picture;
::  FillOval     !Oval  !Picture   ->  Picture;
::  EraseOval    !Oval  !Picture   ->  Picture;
::  InvertOval   !Oval  !Picture   ->  Picture;
```

A Circle with centre c (a Point) and radius r (an INT) is defined by the tuple (c, r).

```
::  DrawCircle    !Circle    !Picture   ->  Picture;
::  FillCircle    !Circle    !Picture   ->  Picture;
::  EraseCircle   !Circle    !Picture   ->  Picture;
::  InvertCircle  !Circle    !Picture   ->  Picture;
```

A Wedge is a part of an Oval o starting from angle a up to angle b (both in degrees modulo 360): (o, a, b). Angles are always taken counterclockwise, starting from 3 o'clock.

```
::  DrawWedge      !Wedge   !Picture   ->  Picture;
::  FillWedge      !Wedge   !Picture   ->  Picture;
::  EraseWedge     !Wedge   !Picture   ->  Picture;
::  InvertWedge    !Wedge   !Picture   ->  Picture;
```

A Polygon is a figure drawn by a number of lines without taking the pen of the Picture, starting from and ending at some Point p. The PolygonShape is a list of Vectors ($[v_1,...,v_n]$) that defines how the Polygon is drawn: (p, $[v_1,...,v_n]$).

ScalePolygon scales the polygon. Non-positive scale factors are allowed.
MovePolygonTo changes the starting point into the given Point and
MovePolygon moves the starting point by the given Vector.

```
::  ScalePolygon     !INT       !Polygon  ->  Polygon;
::  MovePolygonTo    !Point     !Polygon  ->  Polygon;
::  MovePolygon      !Vector    !Polygon  ->  Polygon;

::  DrawPolygon      !Polygon   !Picture  ->  Picture;
::  FillPolygon      !Polygon   !Picture  ->  Picture;
::  ErasePolygon     !Polygon   !Picture  ->  Picture;
::  InvertPolygon    !Polygon   !Picture  ->  Picture;
```

Operations on fonts (deltaFont)

```
DEFINITION MODULE deltaFont;
ABSTYPE
::  Font;

TYPE
::  FontName    ->  STRING;
::  FontStyle   ->  STRING;
::  FontSize    ->  INT;
::  FontInfo    ->  (!INT, !INT, !INT, !INT);

MACRO
    MinFontSize     ->  … ;        == system dependent
    MaxFontSize     ->  … ;        == system dependent

RULE
```

SelectFont creates the font as specified by the name, the stylistic variations and size. The size is always adjusted between MinFontSize and MaxFontSize. The boolean result is TRUE if the font is available and need not be scaled. In case the font is not available, the default font is chosen in the specified style and size.

```
::  SelectFont    !FontName ![FontStyle] !FontSize   ->  (!BOOL, !Font);
```

DefaultFont returns the default font, specified by name, style and size.

```
::  DefaultFont   ->  (!FontName, ![FontStyle], !FontSize);
```

FontNames returns the FontNames of all available fonts.
FontStyles returns the available FontStyles for a certain font.
FontSizes returns all FontSizes of a font that are available without scaling.

In case the font is not available, the styles or sizes of the default font are returned.

```
:: FontNames                 -> [FontName];
:: FontStyles  !FontName     -> [FontStyle];
:: FontSizes   !FontName     -> [FontSize];
```

FontCharWidth(s) and FontStringWidth(s) return the width(s) in terms of pixels of given character(s) or string(s) in a certain Font.

```
:: FontCharWidth      !CHAR      !Font  -> INT;
:: FontCharWidths     ![CHAR]    !Font  -> [INT];
:: FontStringWidth    !STRING    !Font  -> INT;
:: FontStringWidths   ![STRING]  !Font  -> [INT];
```

FontMetrics yields the FontInfo in terms of pixels of a given Font. The FontInfo is a four-tuple (ascent, descent, maxwidth, leading), which defines the font metrics:
- ascent is the maximum height of a character measured from the base line.
- descent is the maximum depth of a character measured from the base line.
- maxwidth is the width of the widest character.
- leading is the vertical distance between two lines of the same font.

The full height of a line is the sum of the ascent, descent and leading.

```
:: FontMetrics  !Font  -> FontInfo;
```

System dependent constants and functions (deltaSystem)

DEFINITION MODULE **deltaSystem**;
MACRO
Keyboard constants (of type KeyCode):

```
UpKey         -> ...;  == Arrow up
DownKey       -> ...;  == Arrow down
LeftKey       -> ...;  == Arrow left
RightKey      -> ...;  == Arrow right
PgUpKey       -> ...;  == Page up
PgDownKey     -> ...;  == Page down
BeginKey      -> ...;  == Begin of text
EndKey        -> ...;  == End of text
BackSpKey     -> ...;  == Backspace
DelKey        -> ...;  == Delete
TabKey        -> ...;  == Tab
ReturnKey     -> ...;  == Return
EnterKey      -> ...;  == Enter
EscapeKey     -> ...;  == Escape
HelpKey       -> ...;  == Help
```

Separator between directory- and filenames in a pathname (of type CHAR):

```
DirSeparator  -> ...;
```

Constants to check which of the meta keys (modifiers) is down (of type Modifiers):

```
ShiftOnly     -> (TRUE,FALSE,FALSE,FALSE);
OptionOnly    -> (FALSE,TRUE,FALSE,FALSE);
CommandOnly   -> ...;  == depends on whether the keyboard of the system
ControlOnly   -> ...;  == has a Command key.
```

The minimum and maximum sizes of a picture domain (of type INT):

```
MinPictureDomain  -> ...;
MaxPictureDomain  -> ...;
```

RULE
The functions HomePath and ApplicationPath prefix the pathname given to them with the full pathname of the 'home' and 'application' directory. The 'home' directory is the home directory of the user (on single-user systems this is the current working directory). The 'application' directory is the directory in which the application resides. One should note that users may not have write access to the 'application' directory.

```
:: HomePath         !STRING  -> STRING;
:: ApplicationPath  !STRING  -> STRING;
```

Functions to retrieve the horizontal and vertical screen resolution.

```
::  MMToHorPixels     !REAL   ->  INT;
::  MMToVerPixels     !REAL   ->  INT;
::  InchToHorPixels   !REAL   ->  INT;
::  InchToVerPixels   !REAL   ->  INT;
```

Functions that return the maximum width and height for the indicated type of window such that the window will fit on the screen. For FixedWindows these are the maximum width and height of the PictureDomain such that the window does not become a ScrollWindow.

```
::  MaxScrollWindowSize  ->  (!INT, !INT);
::  MaxFixedWindowSize   ->  (!INT, !INT);
```