Appendix: Additional Module Definitions

Module MeasuringSlicer

```
DEFINITIONMeasuringSlicer;

   IMPORT Slicer, SlicerOPT, SlicerOPS;

   TYPE
      Slice = POINTER TO SliceDesc;
      SliceDesc = RECORD (Slicer.SliceDesc)
         PROCEDURE (s: Slice) BuildClassHierarchy;
         PROCEDURE (s: Slice) Compile (mod: ARRAY OF CHAR; VAR done: BOOLEAN);
         PROCEDURE (s: Slice) CompleteComputation;
         PROCEDURE (s: Slice) ControlFlow;
         PROCEDURE (s: Slice) DataFlow;
         PROCEDURE (s: Slice) MayAlias (o1, o2, proc: SlicerOPT.Object): BOOLEAN;
         PROCEDURE (s: Slice) SliceProc (node: SlicerOPT.Node; interprocedural: BOOLEAN);
         PROCEDURE (s: Slice) SliceProcForObj (proc: SlicerOPT.Node; obj: SlicerOPT.Object);
         PROCEDURE (s: Slice) SliceStat (node: SlicerOPT.Node; interprocedural: BOOLEAN);
         PROCEDURE (s: Slice) Statistics;
      END ;

   PROCEDURE InitSlice (s: Slice);
   PROCEDUREInstallMeasuringSlicer;
   PROCEDURE SliceFactoryMethod (): Slicer.Slice;

ENDMeasuringSlicer.
```

Module SlicerFE

```
DEFINITIONSlicerFE;

   IMPORT TextFrames, Display, Texts, Slicer, PopupElems, SlicerOPT, SlicerOPS;

   CONST
      unexpectedSituation = 99;
      version = "Oberon Slicing Tool V1.0 (CS)";
      (* options *)
      withDDElems = 1;
      withParInfoElems = 2; withActualParElems = 3;
      withCallingElems = 4; withCalledAtElems = 5;
      withDynTypeElems = 6; withAliasElems = 7;
      withParameterSummary = 8; withReachingEnd = 9;
```

```
    withPosition = 10; interprocedural = 11;
    defaultOptions = {withDDElems, withParInfoElems, withActualParElems,
      withCallingElems, withCalledAtElems, withDynTypeElems, withAliasElems,
      withPosition, interprocedural};
    (* id of SelectMessage *)
    on = 1; off = 2; toggle = 3;

  TYPE
    AliasElem = POINTER TO RECORD (PopupElems.ElemDesc)
    END ;
    DynTypeElem = POINTER TO RECORD (PopupElems.ElemDesc)
    END ;
    Frame = POINTER TO FrameDesc;
    FrameDesc = RECORD (TextFrames.FrameDesc)
      slice: Slicer.Slice;
      options: SET;
    END ;
    SelectMessage = RECORD (Texts.ElemMsg)
      frame: Frame;
      id: INTEGER;
      name: ARRAY 128 OF CHAR;
    END ;
    SliceMsg = RECORD (Display.FrameMsg)
      slice: Slicer.Slice;
      op: INTEGER;
    END ;

  VAR
    forcePersistence: BOOLEAN;
    recording: BOOLEAN;

  PROCEDURE AllocPopup;
  PROCEDURE ControlFlow;
  PROCEDURE DataFlow;
  PROCEDURE FindNode;
  PROCEDURE FindProc;
  PROCEDURE Flush;
  PROCEDURE Handle (f: Display.Frame; VAR msg: Display.FrameMsg);
  PROCEDURE InspectSlice;
  PROCEDURE MakePersistent;
  PROCEDURE NewFrame (slice: Slicer.Slice; T: Texts.Text; pos: LONGINT): Frame;
  PROCEDURE NotifyDisplay (s: Slicer.Slice; op: INTEGER);
  PROCEDURE Open;
  PROCEDURE OpenCallHierarchyViewer;
  PROCEDURE Playback;
  PROCEDURE ReconstructSource;
  PROCEDURE ResetDataFlowInfo;
  PROCEDURE SetAliases;
  PROCEDURE SetArrayExpansionLimit;
  PROCEDURE SetDynamicTypes;
  PROCEDURE SetForcePersistence;
  PROCEDURE SetMark;
  PROCEDURE SetOption;
  PROCEDURE SetRecording;
  PROCEDURE ShowOptions;
  PROCEDURE ShowPosition;
  PROCEDURE Statistics;

END SlicerFE.
```

Module ParInfoElems

```
DEFINITION ParInfoElems;

   IMPORT Texts, Slicer, SlicerOPT, SlicerOPS;

   CONST
      version = "Oberon Slicing Tool V1.0 (CS)";

   TYPE
      ActualParElem = POINTER TO ActualParElemDesc;
      ActualParElemDesc = RECORD (Texts.ElemDesc) END ;
      Elem = POINTER TO ElemDesc;
      ElemDesc = RECORD (Texts.ElemDesc)
      END ;

   PROCEDURE Alloc;
   PROCEDURE AllocActualPar;
   PROCEDURE Handle (e: Texts.Elem; VAR msg: Texts.ElemMsg);
   PROCEDURE HandleActualPar (e: Texts.Elem; VAR msg: Texts.ElemMsg);
   PROCEDURE NewActualParElem (varpar, used, defined: BOOLEAN): ActualParElem;
   PROCEDURE NewElem (slice: Slicer.Slice; proc: SlicerOPT.Object; parIn: SlicerOPT.Node;
      sel, abstract: BOOLEAN; col: INTEGER): Elem;
   PROCEDURE Track (e: Elem; VAR msg: Texts.ElemMsg);

END ParInfoElems.
```

Module SlicerOPT

```
DEFINITION SlicerOPT;

   IMPORT SlicerOPS;

   CONST
      (* kinds of parameter usages *)
      parIn = 0; parOut = 1; parUsed = 2; parDefined = 3; parAlwaysDefined = 4; isPar = 5;
      (* kinds of dependences *)
      (* parIn, parOut *) CD = 2; DD = 3; transDD = 4; call = 5; dynCall = 6;
      (* flags for hash table entries *)
      empty = 0; filled = 1; deleted = 2; mustAssign = 3; defOfAlias = 4;
      (* flags for ProcInfo *)
      useDefComputed = 0; solved = 1;
      (* nodes classes *)
      Nfpar = 29; NcallSite = 30; Ndyncall = 31; NloopExit = 32; NprocExit = 33;
      Nhalt = 34;
      (* node subclasses *)
      (* Nfpar *)
      inPar = 0; outPar = 1; additionalInPar = 2; additionalOutPar = 3;
      (* Nvarpar *)
      additionalPar = 1;
      (* Ncall *)
      normal = 0; statMeth = 1; superCall = 2;
      dynMethAllKnown = 3; dynMethNotAllKnown = 4;
      procVarAllKnown = 5; procVarNotAllKnown = 6;
```

```
   MaxConstLen = 256;
   unexpectedSituation = 99;

TYPE
   Access = POINTER TO AccessDesc;
   AccessDesc = RECORD
      obj: Object;
      node: Node;
   END ;
   AccessArr = POINTER TO ARRAY OF Access;
   AccessIterator = RECORD
      PROCEDURE (VAR it: AccessIterator) First (): Access;
      PROCEDURE (VAR it: AccessIterator) Next (): Access;
   END ;
   AliasIterator = RECORD
      PROCEDURE (VAR it: AliasIterator) First (): Object;
      PROCEDURE (VAR it: AliasIterator) GetSelection (): BOOLEAN;
      PROCEDURE (VAR it: AliasIterator) Next (): Object;
      PROCEDURE (VAR it: AliasIterator) NextSelected (): Object;
      PROCEDURE (VAR it: AliasIterator) SetSelection (sel: BOOLEAN);
   END ;
   CallIterator = RECORD
      end-: BOOLEAN;
      PROCEDURE (VAR it: CallIterator) AdditionalAPars (): Node;
      PROCEDURE (VAR it: CallIterator) Advance;
      PROCEDURE (VAR it: CallIterator) CallNode (): Node;
      PROCEDURE (VAR it: CallIterator) ProcObj (): Object;
      PROCEDURE (VAR it: CallIterator) Reset;
   END ;
   ChoiceIterator = RECORD
      node-: Node;
      PROCEDURE (VAR it: ChoiceIterator) First (): Node;
      PROCEDURE (VAR it: ChoiceIterator) GetSelection (): BOOLEAN;
      PROCEDURE (VAR it: ChoiceIterator) Next (): Node;
      PROCEDURE (VAR it: ChoiceIterator) NextSelected (): Node;
      PROCEDURE (VAR it: ChoiceIterator) SetSelection (sel: BOOLEAN);
   END ;
   Const = POINTER TO ConstDesc;
   ConstDesc = RECORD
      ext: ConstExt;
      intval, intval2: LONGINT;
      setval: SET;
      realval: LONGREAL;
      id: LONGINT;
   END ;
   ConstExt = POINTER TO SlicerOPS.String;
   Definitions = POINTER TO ARRAY OF VarDef;
   Dependences = POINTER TO DependencesDesc;
   DependencesDesc = RECORD
      cds, dds, parIns, parOuts, transdds, dyncalls: NodeArr;
   END ;
   HashTable = RECORD
      size, count: LONGINT;
      PROCEDURE (VAR h: HashTable) Found (o: Object; n: Node;
         VAR pos: LONGINT): BOOLEAN;
      PROCEDURE (VAR h: HashTable) Free;
      PROCEDURE (VAR h: HashTable) Init (size: LONGINT);
      PROCEDURE (VAR h: HashTable) Insert (o: Object; n: Node; flags: SET);
      PROCEDURE (VAR h: HashTable) Reset;
```

```
    PROCEDURE (VAR ht: HashTable) SetIterator (VAR it: HashTableIterator);
END ;
HashTableIterator = RECORD
    end-: BOOLEAN;
    pos-: LONGINT;
    PROCEDURE (VAR it: HashTableIterator) CurFlags (): SET;
    PROCEDURE (VAR it: HashTableIterator) CurMustAssign (): BOOLEAN;
    PROCEDURE (VAR it: HashTableIterator) CurNode (): Node;
    PROCEDURE (VAR it: HashTableIterator) CurObj (): Object;
    PROCEDURE (VAR it: HashTableIterator) Next;
    PROCEDURE (VAR it: HashTableIterator) SetPos (pos: LONGINT);
END ;
Node = POINTER TO NodeDesc;
NodeDesc = RECORD
    left, right, link: Node;
    class, subcl: SHORTINT;
    readonly: BOOLEAN;
    mark: SHORTINT;
    typ: Struct;
    obj: Object;
    conval: Const;
    procInfo:ProcInfo;
    usedObjs, definedObjs: ObjArr;
    dependences:Dependences;
    gen, kill, in, choice: SetArr;
    aliases: ObjArr;
    enabledAliases:SetArr;
    id: LONGINT;
    PROCEDURE (node: Node) SetAliasIterator (VAR it: AliasIterator);
    PROCEDURE (node: Node) SetChoiceIterator (VAR it: ChoiceIterator);
    PROCEDURE (node: Node) SetDependenceIterator (kind: SHORTINT;
        VAR it: NodeIterator);
END ;
NodeArr = POINTER TO ARRAY OF Node;
NodeIterator = RECORD
    PROCEDURE (VAR it: NodeIterator) First (): Node;
    PROCEDURE (VAR it: NodeIterator) Next (): Node;
END ;
Nodes = POINTER TO NodesDesc;
NodesDesc = RECORD
    using, defining: NodeArr;
    nofUsing, nofDefining: INTEGER;
    PROCEDURE (nodes: Nodes) SetNodeIterator (using: BOOLEAN; VAR it: NodeIterator);
END ;
ObjArr = POINTER TO ARRAY OF Object;
Object = POINTER TO ObjDesc;
ObjDesc = RECORD
    left, right, link, scope: Object;
    name: SlicerOPS.Name;
    leaf: BOOLEAN;
    mode, mnolev, vis: SHORTINT;
    typ: Struct;
    conval: Const;
    adr, linkadr: LONGINT;
    nodes: Nodes;
    procInfo:ProcInfo;
    assignedToProcVar:BOOLEAN;
    mark, level: SHORTINT;
    mod: Object;
```

```
    expanded: BOOLEAN;
    components: ObjArr;
    containedIn:Object;
    id: LONGINT;
  END ;
  ObjectIterator = RECORD
    PROCEDURE (VAR it: ObjectIterator) First (): Object;
    PROCEDURE (VAR it: ObjectIterator) IsIn (obj: Object): BOOLEAN;
    PROCEDURE (VAR it: ObjectIterator) Next (): Object;
  END ;
  Objects = POINTER TO ObjectsDesc;
  ObjectsDesc = RECORD
    used, defined: ObjArr;
    nofUsed, nofDefined: INTEGER;
    PROCEDURE (objs: Objects) SetObjectIterator (used: BOOLEAN; VAR it: ObjectIterator);
  END ;
  ProcInfo = POINTER TO ProcInfoDesc;
  ProcInfoDesc = RECORD
    fpars, callSites: Node;
    calls: NodeArr;
    accesses: AccessArr;
    procExit, enter: Node;
    procObj: Object;
    in, out: SetArr;
    objs: Objects;
    definitionsHT:HashTable;
    varDefs:Definitions;
    flags: SET;
    id: LONGINT;
    PROCEDURE (procInfo: ProcInfo) AddCall (call, callee: Node; VAR callSite: Node);
    PROCEDURE (procInfo: ProcInfo) AddFPar (node: Node);
    PROCEDURE (procInfo: ProcInfo) InsertObj (obj: Object; used: BOOLEAN);
    PROCEDURE (procInfo: ProcInfo) RegisterAccess (obj: Object; node: Node);
    PROCEDURE (procInfo: ProcInfo) SetAccessIterator (VAR it: AccessIterator);
    PROCEDURE (procInfo: ProcInfo) SetCallIterator (VAR it: NodeIterator);
  END ;
  SetArr = POINTER TO ARRAY OF SET;
  Struct = POINTER TO StrDesc;
  StrDesc = RECORD
    form, comp, mno, extlev: SHORTINT;
    ref, sysflag: INTEGER;
    n, size, tdadr, offset, txtpos: LONGINT;
    BaseTyp: Struct;
    link, strobj, mod: Object;
    extensions:StructArr;
    fields: ObjArr;
    id: LONGINT;
    mark: SHORTINT;
  END ;
  StructArr = POINTER TO ARRAY OF Struct;
  StructIterator = RECORD
    PROCEDURE (VAR it: StructIterator) First (): Struct;
    PROCEDURE (VAR it: StructIterator) Next (): Struct;
  END ;
  VarDef = RECORD
    mustAssign-, mayAssign-: SetArr;
  END ;
  WorkProcObject = PROCEDURE (o: Object);
```

```
VAR
   GlbMod: ARRAY 31 OF Object;
   ModFromRepository: PROCEDURE (name: ARRAY OF CHAR; key: LONGINT): Object;
   SYSimported:BOOLEAN;
   booltyp: Struct;
   bytetyp:Struct;
   chartyp: Struct;
   currentModule:Object;
   forwards: ObjArr;
   inttyp:Struct;
   linttyp:Struct;
   lrltyp: Struct;
   niltyp:Struct;
   nofForwards:LONGINT;
   nofGmod: SHORTINT;
   notyp:Struct;
   realtyp: Struct;
   settyp:Struct;
   sinttyp:Struct;
   stringtyp:Struct;
   syslink: Object;
   sysptrtyp:Struct;
   topScope: Object;
   undftyp:Struct;
   universe:Object;

PROCEDURE AppendNode (VAR head: Node; node: Node);
PROCEDURE Close;
PROCEDURE CloseScope;
PROCEDURE ExistsDependence (from, to: Node; kind: SHORTINT): BOOLEAN;
PROCEDURE Export (VAR modName: SlicerOPS.Name; VAR newSF: BOOLEAN;
   VAR key: LONGINT);
PROCEDURE Find (VAR res: Object);
PROCEDURE FindField (VAR name: SlicerOPS.Name; typ: Struct; VAR res: Object);
PROCEDURE FindImport (mod: Object; VAR res: Object);
PROCEDURE FindMethod (name: ARRAY OF CHAR; typ: Struct): Object;
PROCEDURE FindOverriddenMethod (name: ARRAY OF CHAR; typ: Struct): Object;
PROCEDURE FirstNode (n: Node; class: SHORTINT; subclasses: SET): Node;
PROCEDURE GetAliasForRealName (realName: ARRAY OF CHAR;
   VAR alias: ARRAY OF CHAR);
PROCEDURE GetRealNameForAlias (alias: ARRAY OF CHAR;
   VAR realName: ARRAY OF CHAR);
PROCEDURE Import (VAR aliasName, impName, selfName: SlicerOPS.Name);
PROCEDURE IndexOfNode (nodeArr: NodeArr; node: Node): LONGINT;
PROCEDURE IndexOfObject (objArr: ObjArr; obj: Object): LONGINT;
PROCEDURE Init;
PROCEDURE Insert (VAR name: SlicerOPS.Name; VAR obj: Object);
PROCEDURE InsertAlias (at: Node; alias: Object);
PROCEDURE InsertDependence (from, to: Node; kind: SHORTINT);
PROCEDURE InsertDyncall (from, to: Node);
PROCEDURE InsertFwdDecl (proc: Object);
PROCEDURE InsertNew (VAR varDefs: Definitions; obj: Object; size: LONGINT): LONGINT;
PROCEDURE InsertNode (VAR nodeArr: NodeArr; node: Node);
PROCEDURE InsertObject (VAR objArr: ObjArr; obj: Object);
PROCEDURE InsertStruct (VAR arr: StructArr; str: Struct);
PROCEDURE InsertUseDef (node: Node; obj: Object; used: BOOLEAN);
PROCEDURE IsExtended (typ: Struct): BOOLEAN;
PROCEDURE IsGlobal (obj: Object): BOOLEAN;
PROCEDURE IsIntermediate (obj, proc: Object): BOOLEAN;
```

```
   PROCEDURE IsLocal (obj, proc: Object): BOOLEAN;
   PROCEDURE IsOverridden (typ: Struct; name: ARRAY OF CHAR): BOOLEAN;
   PROCEDURE Lookup (scope: Object; name: ARRAY OF CHAR; VAR obj: Object);
   PROCEDURE MatchingParameterLists (par1, par2: Object; ret1, ret2: Struct): BOOLEAN;
   PROCEDURE NestingLevel (obj: Object): SHORTINT;
   PROCEDURE NewConst (): Const;
   PROCEDURE NewExt (): ConstExt;
   PROCEDURE NewNode (class: SHORTINT): Node;
   PROCEDURE NewObj (): Object;
   PROCEDURE NewProcInfo (): ProcInfo;
   PROCEDURE NewStr (form, comp: SHORTINT): Struct;
   PROCEDURE OpenScope (level: SHORTINT; owner: Object);
   PROCEDURE ProcObj (call: Node): Object;
   PROCEDURE SameType (t1, t2: Struct): BOOLEAN;
   PROCEDURE SetCallIterator (node: Node; VAR it: CallIterator);
   PROCEDURE SetNodeIterator (nodes: NodeArr; VAR it: NodeIterator);
   PROCEDURE SetObjectIterator (objs: ObjArr; VAR it: ObjectIterator);
   PROCEDURE SetStructIterator (structs: StructArr; VAR it: StructIterator);
   PROCEDURE SetsClear (s: SetArr);
   PROCEDURE SetsCopy (s1, s2: SetArr);
   PROCEDURE SetsDifference (s1, s2, s3: SetArr);
   PROCEDURE SetsEmpty (s: SetArr): BOOLEAN;
   PROCEDURE SetsEqual (s1, s2: SetArr): BOOLEAN;
   PROCEDURE SetsExcl (s: SetArr; x: LONGINT);
   PROCEDURE SetsFill (s: SetArr);
   PROCEDURE SetsIn (s: SetArr; x: LONGINT): BOOLEAN;
   PROCEDURE SetsIncl (s: SetArr; x: LONGINT);
   PROCEDURE SetsIntersection (s1, s2, s3: SetArr);
   PROCEDURE SetsNew (VAR s: SetArr; size: LONGINT);
   PROCEDURE SetsPrint (s: SetArr; VAR ht: HashTable);
   PROCEDURE SetsUnion (s1, s2, s3: SetArr);
   PROCEDURE Statistics;
   PROCEDURE ThisAdditionalPar (call: Node; obj: Object): Node;
   PROCEDURE ThisFPar (procInfo: ProcInfo; obj: Object): Node;
   PROCEDURE ThisFPar2 (procInfo: ProcInfo; kinds: SET; obj: Object): Node;
   PROCEDURE ThisVar (name: ARRAY OF CHAR; proc: Node): Object;
   PROCEDURE ThisVarDef (varDefs: Definitions; obj: Object): LONGINT;
   PROCEDURE TraverseSymbolTable (scope: Object; proc: WorkProcObject);

END SlicerOPT.
```

Module SlicerAuxiliaries

```
DEFINITIONSlicerAuxiliaries;

   IMPORT SlicerOPT, SlicerOPS;

   TYPE
      CSGNode = POINTER TO CSGNodeDesc;
      Fixups = POINTER TO FixupsDesc;
      FixupsDesc = RECORD
         n: INTEGER;
         arr: SlicerOPT.NodeArr;
         PROCEDURE (f: Fixups) Fixup (to: SlicerOPT.Node);
         PROCEDURE (f: Fixups) Init;
```

```
      PROCEDURE (f: Fixups) Insert (n: SlicerOPT.Node);
      PROCEDURE (f: Fixups) SetIterator (VAR it: FixupsIterator);
    END ;
    FixupsIterator = RECORD
      PROCEDURE (VAR it: FixupsIterator) First (): SlicerOPT.Node;
      PROCEDURE (VAR it: FixupsIterator) Next (): SlicerOPT.Node;
    END ;

  PROCEDURE FindProc (root: CSGNode; proc: SlicerOPT.Node): CSGNode;
  PROCEDURE InsertProc (VAR root: CSGNode; proc: SlicerOPT.Node);
  PROCEDURE RemoveProc (VAR root: CSGNode; proc: SlicerOPT.Node): CSGNode;
  PROCEDURE ShowECSG (root: CSGNode);
  PROCEDURE UpdateExistingProc (VAR root: CSGNode; caller, callee: SlicerOPT.Node);

ENDSlicerAuxiliaries.
```