

Cedar Style Sheet

No common suffix on the names of DEFINITIONS modules

```
-- FILE: CedarSample.mesa
-- Last edited by Mitchell, April 23, 1980 12:05 PM
```

Standard prelude for a module

Capitalize the first letter of module procedure, signal, or type names

```
CedarSample: DEFINITIONS =
BEGIN
upperLimit: INTEGER = 32; -- name the upper limit of an interval type
IntervalType: TYPE = [0 .. upperLimit]; -- preferred form for interval type
Sample: TYPE = REF SampleRec;
SampleRec: TYPE = RECORD[val: Value, next: Sample];
SampleSet: TYPE = REF SampleSetRec;
SampleSetRec: TYPE = RECORD[head: Sample, count: Value];
```

Using a type is better than using anonymous type constructor.

Capitalize the first letter of each imbedded word of a multi-word name

```
Problem: ERROR [reason: ErrorCode];
ErrorCode: TYPE = {damagedSample, callingError, programError};
NewSample: PROCEDURE [val: Value] RETURNS [Sample];
NewSampleSet: PROCEDURE RETURNS [nilSet: SampleSet];
RemoveSample: PROCEDURE [toBeRemoved: Sample, from: SampleSet]
    RETURNS [inSet: BOOLEAN, setMinus: SampleSet];
-- ERRORS: Problem[damagedSample]
-- SIGNALS: ResumableCondition, SomeSignal
emptySet: SampleSet = NIL;
END.
```

Use a small set of ERRORS with an error code parameter.

attach stylized comments to procedures that generate ERRORS or SIGNALS.

```
This is where a global description of the module goes
CHANGE LOG
Changed by: YourName: DateTime
DescriptionOfChange
```

Standard postlude for a module includes a history log of (non-trivial) changes to the module

PROGRAM module name is normally formed by suffixing interface name with "Impl". Alternatively, name may be totally different than interface name

```
-- FILE: CedarSampleImpl.mesa
-- Last edited by Mitchell, April 23, 1980 3:19 PM
DIRECTORY
SomeInterface USING [SomeProc, SomeType],
CedarSample;
```

OK to use unnamed OPEN of interface if DIRECTORY entry has a USING list

NO names in same scope differing only by letter case distinctions except a value with same name as its type but with lowercase first letter

```
CedarSampleImpl PROGRAM IMPORTS SI: SomeInterface
EXPORTS CedarSample =
BEGIN OPEN CedarSample;
Problem: ERROR[reason: ErrorCode] = CODE;
ResumableCondition: PUBLIC SIGNAL = CODE;
SomeSignal: PUBLIC SIGNAL = CODE;
NewSample: PUBLIC PROCEDURE [val: Value] RETURNS [Sample] =
BEGIN
sample: Sample;
x: SI.SomeType = 0;
IF val = 0 THEN x := SomeProc(x);
RETURN [sample];
END;
```

OK to OPEN interface that module implements

Qualify identifiers from interfaces

```
IF val = 0 THEN x := SomeProc(x);
RETURN [sample];
END;
```

Keyword constructors, argument lists and extractors preferred for multiple component constructors

```
NewSampleSet: PUBLIC PROCEDURE RETURNS [nilSet: SampleSet] =
BEGIN
nilSet _ AllocatesSampleSet[allocFault => ERROR Problem[programError]]
nilSet [head: NIL, count: 0];
END;
```

Only let signals that are part of abstraction escape out of it.

Use REF ANY instead of variant records and discriminate

```
Bug: ERROR = CODE;
RemoveSample: PUBLIC PROCEDURE [toBeRemoved: Sample, from: SampleSet]
    RETURNS [inSet: BOOLEAN, setMinus: SampleSet] =
BEGIN OPEN SI;
IF . . . THEN ERROR Problem[damagedSample];
WITH refAnyVar SELECT FROM
r: REF REAL => r^ - r^ + 1.0;
i: REF INT => { . SomeProc[y]; . . };
b: REF BOOLEAN => SIGNAL ResumableCondition;
ENDCASE => ERROR Bug;
```

OK to OPEN an interface in a local scope where it is heavily used

Calls on single-argument procedures don't have to use keyword notation

```
AllocFault: ERROR CODE; -- error for local use in this module
AllocateSampleSet : PROCEDURE RETURNS [unitedSet: SampleSet] =
BEGIN
. . .
END;
```

Only raise SIGNALS using SIGNAL, and ERROR using ERROR

NO using ENDCASE to handle a single remaining case: use as "OTHERWISE" or to generate an ERROR

```
CHANGE LOG
Changed by: YourName: DateTime
DescriptionOfChange
```

OK to have locally defined ERRORS and SIGNALS