Migration of Old Library Module Usage

Uwe Kastens

University of Paderborn
D-33098 Paderborn
FRG

$Revision: 1.6 $

Table of Contents

This section helps to migrate applications of previous module library version to the current one.

Users who want to continue to use library modules of a previous Eli version, please contact your Eli system manager.

`ModLib3_8`
: Migration of Eli Version 3.8 modules

`ModLib3_6`
: Migration of Eli Version 3.6 modules

1 Migration of Eli Version 3.8 modules

This section gives a quick reference to modifications that have been made in library modules of Eli version 3.8. Users who are still using modules of Eli version 3.6 first see Chapter 2 [ModLib3_6], page 7.

In all library modules that provide LIDO specifications the symbol roles are now specified to be `CLASS SYMBOL`s. This classification avoids accidental name coincidences between grammar symbols and roles provided by modules.

As a consequence such a symbol may not occur as a name of a tree grammar symbol. It rather has to be inherited by a tree grammar symbol:

```
RULE: Program ::= Source END;
SYMBOL Program INHERITS RootScope END;
```

Constructs like

```
RULE: RootScope ::= Source END;
```

will cause error messages. Hence, the grammar symbols have to be renamed consistently in such cases.

We assume that some names like `IdDef` or `IdUse` have been used in this now illegal way. In order to simplify migration those module roles are now renamed `IdDefScope` and `IdUseEnv`. Hence, specifications which use `IdDef` or `IdUse` as grammar symbols may continue to do so. Only

```
SYMBOL IdDef INHERITS IdDefScope END;
SYMBOL IdUse INHERITS IdUseEnv END;
```

is to be added.

The following table contains a list symbol roles that are provided by modules of Eli version 3.8 and have been changed in the current version. All these adaptions refer to modules of the `Name` library.

The following library modules have been modified:

`BitSet`
: Use `$/Adt/BitSet.fw` instead of instantiation. (see Section "Bit Sets of Arbitrary Length" in *Abstract data types to be used in specifications*)

`DynSpace`
: Use `$/Adt/DynSpace.fw` instead of instantiation. (see Section "Dynamic Storage Allocation" in *Abstract data types to be used in specifications*)

`LeafPtg`
: Use `$/Output/LeafPtg.fw` instead of instantiation. (see Section "PTG Output for Leaf Nodes" in *Tasks related to generating output*)

`Indent`
: Use `$/Output/Indent.fw` instead of instantiation. (see Section "Indentation" in *Tasks related to generating output*)

`List`
: The module uses `obstack` directly instead of using module `DynSpace`.

`OutStr`
: Use `$/Output/OutStr.fw` instead of instantiation. (see Section "Output String Conversion" in *Tasks related to generating output*)

`Name Analysis`
: There are minor modifications in all name analysis modules. See table of entity modifications below, and (see Section "Basic Scope Rules" in *Name analysis*

according to scope rules) (see Section “Scopes Being Properties of Objects” in *Name analysis according to scope rules*) (see Section “Inheritance of Scopes” in *Name analysis according to scope rules*)

`PreDefId` Instead of instantiating the module `PreDefId` once for each predefined identifier, all predefined identifiers are be described in one file. Its name is supplied as `referto` parameter. (see Section “Predefined Identifiers” in *Name analysis according to scope rules*)

`PropLib` Use `$/Prop/PropLib.fw` instead of instantiation. (see Section “Some Useful PDL Specifications” in *Association of properties to definitions*)

`Type CType BuType`
These modules are still usable. But they are not maintained anymore. It is recommended to use the module `Typing` (See Section “Typed Entities” in *Type Analysis Reference Manual*,) instead. The functionality of `CType` and `BuType`, where the C-like name analysis scheme is also imposed upon type analysis is not supported by `Typing`. Modules having corresponding facilities are not yet available.

The following entities of library modules have been modified:

`AnyScope.GotScopes`
Replaced: use `RootScope.GotScopeProp` (see Section “Scopes Being Properties of Objects” in *Name analysis according to scope rules*)

`AnyScope.GotScopesDefs`
Replaced: use `RootScope.GotScopeProp` (see Section “Scopes Being Properties of Objects” in *Name analysis according to scope rules*)

`IdDef` Renamed to `IdDefScope`. (see Section “Basic Scope Rules” in *Name analysis according to scope rules*)

`IdDefScopeProp`
Deleted: The scope property is now associated to a key in the context of the role `RangeScopeProp`. (see Section “Scopes Being Properties of Objects” in *Name analysis according to scope rules*)

`IdGetScopeProp`
Deleted: Use the `GetScope` function explicitly if necessary. (see Section “Scopes Being Properties of Objects” in *Name analysis according to scope rules*)

`IdUse` Renamed to `IdUseEnv`. (see Section “Basic Scope Rules” in *Name analysis according to scope rule*)

`IdUseScope`
modified: The default scope to be used can not be changed (see Section “Basic Scope Rules” in *Name analysis according to scope rule*); use `IdUseScopeProp` if necessary (see Section “Scopes Being Properties of Objects” in *Name analysis according to scope rule*).

`InheritScopeProp`
Renamed and modified: use `InheritScope`. (see Section “Inheritance of Scopes” in *Name analysis according to scope rule*)

`RangeScopeProp`

Modified: A computation of `THIS.ScopeKey` for the key to which the scope is to be associated has to be provided, instead of `THIS.Scope`. (see Section "Scopes Being Properties of Objects" in *Name analysis according to scope rule*)

2 Migration of Eli Version 3.6 modules

The following table is a quick reference for migration of module usage from module library upto Eli version 3.6 to modules of this library. For each module name of the old library a reference to the documentation of the corresponding module of this library is given. In many cases it is sufficient to modify the instantiation command. Others require modifications of identifiers in `.lido` specifications, or in the way they are used.

Note: The modules of the library of Eli version 3.6 which are mentioned below are no longer contained in the Eli distribution. Users who want to continue to use them, please contact your Eli system manager.

`AdaptOil`	use Section "Operator Identification" in *Type analysis task*,
`Bool`	removed
`Chain`	use Section "C-like Basic Scope Rules" in *Name analysis according to scope rule*,
`ChainPtg`	removed, see PTG documentation
`CmdLineIncl`	now in Section "Input Processing" in *Tasks related to input processing*, library, See Section "Command Line Arguments for Included File" in *Tasks related to input processing*.
`CoordMap`	now in Section "Input Processing" in *Tasks related to input processing*, library, See Section "Command Line Arguments for Included File" in *Tasks related to input processing*.
`CurrTok`	now in Section "Input Processing" in *Tasks related to input processing*, library, See Section "Accessing the Current Token" in *Tasks related to input processing*.
`DefPt`	use Section "Set a Property at the First Object Occurrence" in *Association of properties to definitions*, in Section "Property Library" in *Association of properties to definitions*, library
`DynSpace`	now in Section "Abstract Data Types" in *Abstract data types to be used in specifications*, library, See Section "Dynamic Storage Allocation" in *Abstract data types to be used in specifications*.
`Field`	use Section "Scope Properties Algol-like" in *Name analysis according to scope rules*, or Section "Scope Properties C-like" in *Name analysis according to scope rules*.
`Fwd`	use Section "Associate Kinds to Objects" in *Association of properties to definitions*, or Section "Associate Sets of Kinds to Objects" in *Association of properties to definitions*, in Section "Property Library" in *Association of properties to definitions*, library
`GenName`	use Section "Generating Optional Identifiers" in *Solutions of common problems*.
`GChain`	use Section "C-like Basic Scope Rules Computed Bottom-Up" in *Name analysis according to scope rules*.

`InclLido`	use Section "Insert a File into the Input Stream" in *Tasks related to input processing*, in Section "Input Processing" in *Tasks related to input processing*, library
`Include`	use Section "Insert a File into the Input Stream" in *Tasks related to input processing*, in Section "Input Processing" in *Tasks related to input processing*, library
`Indent`	now in Section "Generating Output" in *Tasks related to generating output*, library, See Section "Indentation" in *Tasks related to input processing*.
`IntSet`	
`IntToKey`	use Section "Mapping Integral Values To Other Types" in *Abstract data types to be used in specifications*, in Section "Abstract Data Types" in *Abstract data types to be used in specifications*, library
`IntToPtr`	use Section "Mapping Integral Values To Other Types" in *Abstract data types to be used in specifications*, in Section "Abstract Data Types" in *Abstract data types to be used in specifications*, library now in Section "Abstract Data Types" in *Abstract data types to be used in specifications*, library, See Section "Bit Sets of Integer Size" in *Abstract data types to be used in specifications*.
`KindSet`	unchanged, See Section "Associate Sets of Kinds to Objects" in *Association of properties to definitions*.
`LeafPtg`	now in Section "Generating Output" in *Tasks related to generating outpu*, library, See Section "PTG Output for Leaf Nodes" in *Tasks related to generating output*.
`List`	use Section "Lists in LIDO Specifications" in *Abstract data types to be used in specifications*, in Section "Abstract Data Types" in *Abstract data types to be used in specifications*, library
`Message`	use Section "String Concatenation" in *Solutions of common problems*, to compose message texts and call the `message`. function directly, as described in Section "Error Reports" in *Solutions of common problems*.
`Nest`	use Section "Algol-like Basic Scope Rules" in *Name analysis according to scope rules*.
`NoKeyMsg`	removed, See Section "Basic Scope Rules" in *Name analysis according to scope rules*.
`OccCnt`	unchanged, See Section "Count Occurrences of Objects" in *Association of properties to definitions*.
`Once`	use Section "Determine First Object Occurrence" in *Association of properties to definitions*.
`OutStr`	now in Section "Generating Output" in *Tasks related to generating output*, library, See Section "Output String Conversion" in *Tasks related to generating output*.
`PreDef`	use `PreDefine` and `PreDefId`, See Section "Predefined Identifiers" in *Name analysis according to scope rules*.

`PtrList`	use Section “Linear Lists of Any Type” in *Abstract data types to be used in specifications*, in Section “Abstract Data Types” in *Abstract data types to be used in specifications*, library
`Stack`	now in Section “Abstract Data Types” in *Abstract data types to be used in specifications*, library, See Section “Stacks of Any Type” in *Abstract data types to be used in specifications*.
`Unique`	now in Section “Property Library” in *Association of properties to definitions*, library, See Section “Check for Unique Object Occurrences” in *Association of properties to definitions*.

Index