START Job DESIGN Req #976 for EGB Date 29-Apr-85 9:37:29 Monitor: //, TOPS
File RM:<SYSTEM.SPEC>WORLDS..2, created: 24-Feb-85 17:33:47
printed: 29-Apr-85 9:41:02
Job parameters: Request created:29-Apr-85 9:37:27 Page limit:261 Forms:NORMAL
File parameters: Copy: 1 of 1 Spacing:SINGLE File format:ASCII Print mode:ASC

1. Basic Structure

A world consists of a directory named to indicate the activity involved. Nested inside this directory are a series of versions of the world, each numbered to indicate their place in the development sequence. Libraries that make up the program contents of the version of the world are nested directly inside the version. An example might be:

```
package Interface is
    package A_0_1_3 is
        Ada_Base : Universe.Library;
        Specs : Universe.Library;
        Main : Universe.Library;
        ...
    end A_0_1_3;
    ...
    package A_1_0_1 is
        Ada_Base : Universe.Library;
        Specs : Universe.Library;
        Main : Universe.Library;
        ...
    end A_1_0_1;
    ...
end Interface;
```

Naming for the version is then Interface.A_1_0_1.Main.Unit_Name.

Worlds will include other information to maintain histories, etc. Availability of nested libraries, etc. would change some of the structural elements, but not the basic structure.

2. Creating libraries.

Since worlds start out empty, one of the first operations to be undertaken will be to create a library. The initial library should include references to required LRM packages (this is target-specific). Creation of subsequent libraries will form a library chain. Creation of a library into a particular position causes it to have the linkages necessary to access the libraries preceding it in the chain and to introduce it into the visibility of the libraries that follow it in the sequence.

It is possible initially to limit library additions to be at the end of the sequence. Determination of library ordering can be done by auxiliary files or processing existing declarations, whichever seems more convenient.

3. Creating New Versions of a World

The simplest form of creating a new version is to simply copy it. References to other libraries in the version must work out correctly, but otherwise straightforward.

Differentials would allow the last N libraries in the chain to be copied while only references to the preceding M would be retained. References are appropriately qualified names in the context clause.

It will be useful to have the ability to partition libraries into two new libraries with appropriate closures, where possible. This could be specified by providing a set of units that should be put in their own library, an operation that can obviously fail. This should be implementable on top of the Directory.Move command, with appropriate demotions and closure calculations.

4. History

Time, date and user making modifications to a particular unit. This can be kept on a voluntary basis, i.e. user offers to explain a change. Would be good if it were recorded automatically, with user annotation to indicate purpose of modification. Could prompt for explanation of change at appropriate time. Problems here are appropriate time and, control over policy (some projects will want to require notes, others won't want to be prompted at all).

Genealogy of the versions of the world that have led to its current state. Time, date and user creating the new version. Copy/differential. Different targets/switches, etc.

Changes to switches, policies, etc.

This would be a good application for a database, but we should probably resist the temptation to build one.

5. Compilation Tools

Operations should be provided that use the currently available compilation tools with the switches and defaults specified for this world.