Chapter 1

Environment

Environment is a special type of dictionary holding parameters used by parts. As figure Figure 1.1 (page 3) shows, there are multiple levels and types of environments playing part in program run. Environment evaluation process walks the project tree from top to program location. In each node (package branch), there are three files defining nodes environment:

1.1 Definitions

No.	*Name*	*Description*
1	.projectenv	read-only project level environment variables; also identifies project root location.
2	projectenv	addon environment for given read-only environment.
3	personalenv	personal overrides for project or package environment. It may include only overrides.
4	.packageenv	read-only package level environment variables.
5	packageenv	addon environment for given read-only environment in local or upper environments.

Table 1.1: Environment Definitions

1.2 Program Interface

Within programs there are three types of access points to environment variables. To get environ dictionary, program can perform the following command:

1. Loading environment variables from project structure
2. Updating environment variable in program
3. Accessing environment variables

1.2.1 Loading environmentvariables

```
import environ
env=environ.Environ()
```

When Program evaluates environment, it starts with root location going down the tree up to an including package environment of Program location.
Environ __init__ has the following signature:

Environ(env=None, path=None, envtree=True, osenv=True, syntax='XML', config=None*dict*)

No.	*Name*	*Description*	*Default Values*
1	env	Direct environment list to override environment tree.	None
2	path	Path to project location or to package within.	Current working directory
3	envtree	If set, load enviroenment files from directory tree.	True
4	osenv	If set, load os environ.	True
5	syntax	Expected structure of environment files.	XML
6	config	dictionary overriding default names for environment files.	'.projectenv':'.projectenv'
			'packageenv':'packageenv'
			'personalenv':'personalenv'
			'envtag':'environ'

Table 1.3: Environ signature arguments

In addition to file name overrides, configuration dictionary can provide syntax used in these files and the environment root tag under which environment variables are defined.
Within derivative articles environment can be updated with environment variable as follows:

1.2.2 Updating environment variables

```
env.update([
  EnvVar(name='REJ_ALLOWED',cast='integer',value=0,input=True),
  EnvVar(name='OUT_FILE',value='${VAR_LOC}/summary.csv',cast='path', input=True),
  EnvVar(name='RATE',override='True',cast='integer',value=5,input=True)])
```

input flag will set the variable with the value of parent environment, if defined.

override flags environment variable as changeable by derivative program articles.

1.2.3 Accessing environment variables

```
ofile=env.environ['OUT_FILE']
rate=env.get('RATE')
```

In the first case, direct access, KeyError exception may be sent if variable name does not exist.
In the second example, None value will be returned if not found.

1.3 Single Project Environment Tree

At each node, environment is evaluated in the following sequence:

1. First .projectenv, if available, is read and set.
2. Next packageenv, if available, is read and set.
3. Finally, personalenv overrides, if available, is read and set.

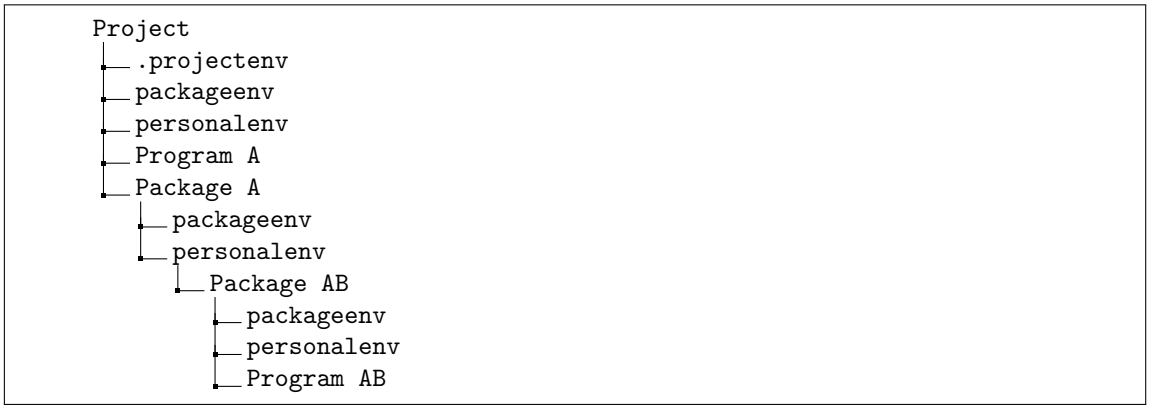

Figure 1.1: Environment tree example

Figure 1.1 (page 3) shows example environment tree in a project.
When the above command is engaged in Program A, it would include environment setting of Project and Package A locations. Program AB will include Program A, Package A and Package AB accordingly.

```
<environment>
  <environ>
    <var name='AC_WS_LOC' value='${HOME}/sand/myproject' export='True'/>
    <var name='AC_ENV_NAME' value='test' export='True'/>
    <var name='AC_VAR_BASE' value='${HOME}/var/data/' export='True'/>
    <var name='AC_LOG_LEVEL' value='DEBUG' export='True'/>
    <var name='AC_LOG_STDOUT' value='True' override='True' export='True' cast='boolean'/>
    <var name='AC_LOG_STDOUT_LEVEL' value='INFO' override='True' export='True'/>
    <var name='AC_LOG_STDERR' value='True' override='True' export='True' cast='boolean'/>
    <var name='AC_LOG_STDERR_LEVEL' value='CRITICAL' override='True' export='True'/>
  </environ>
</environment>
```

Listing 1.1: Example for project environment file

Listing 1.1 shows example of an environment file. Core environment is tagged under < *environ*>. Environ mechanism would look for this tag. Once found, it would evaluate its content as environment directive.

Note: < *environment*> tag is to provide enclosure to environ. Environ mechanism is not depending on its existent per se. However, some kind on enclosure is required; < *environ*> can not be in top level of the XML.

1.4 Multiple Project Environment Tree

At each import, environment is evaluated in the following sequence:

1. First get the node representation of imported path.
2. Evaluate it recursively (loading imports).
3. Finally, insert the resulted imported map instead of the import directive (flat).

```
<environment>
  <environ>
    <var name='FILE_LOC' value='/Users/me/tmp/' export='True'/>
    <var name='FILE_NAME' value='aname' export='True'/>
    <var name='FILE_PATH' value='${FILE_LOC}${FILE_NAME}' export='True'/>
  </environ>
</environment>
```

Listing 1.2: 'Project A: /Users/me/projs/proja/.projectenv.xml

```
<environment>
  <environ>
    <import name='proja' path='/Users/me/projs/proja/.projectenv.xml'/>
    <var name='FILE_NAME' value='bname' export='True'/>
  </environ>
</environment>
```

Listing 1.3: 'Project B: /Users/me/projs/projb/.projectenv.xml'

Listings 1.3 shows import project directive within project B's environment. In project B's context, FILE_PATH will result with the value */Users/me/tmp/bname*.

Recursive inclusion of environments (recursive import statement) would cause evaluation of environment variables to be loaded recursively. Consideration is given to overrides in post import environments.

Note: import path can only include environment variables that are in the OS level pre-eveluation.