Chapter 1

Environment

Environment is a special type of dictionary holding parameters used by parts. As figure Figure 2.1 (page 6) shows, there are multiple levels and types of environment files playing part in program run.

Each level is an environment node that contains parameters for specific section in the project.

1.1 Environment node

Environment evaluation process walks the project tree from top to program location. In each node it looks for set of files that defines the environment parameters.
By default, environment is derived from two type of files as each node:

No.	*Name*	*Description*
1	.envpackage.xml	environment parameters for the package.
2	.envoverride.xml	personal overrides for package environment. It may include re-define parameters with override=True.

Table 1.1: Default environment definitions

Project can alter this default behavior by defining *.envconfig.xml* at its root with specific value for envnodes.

```
<environ>
  <envnodes>.envproject, .envpackage, .envoverride</envnodes>
</environ>
```

Listing 1.1: Example for project configuration environment file

1.2 Program Interface

Within programs there are three types of access points to the environment variables. To get projenv 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 environment variables

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

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

```
Environ(self, osenv=True, trace_env=None, logclass=None, logger=None)
```

No.	*Name*	*Description*	*Default Values*
1	osenv	If set, load os environ.	True
2	trace_env	List of enviornment variables to trace	None
3	logclass	If provided the string will be used for trace naming.	None
4	logger	If set to True and logclass=None, use Python getChild to set trace name.	None

Table 1.3: Environ signature arguments

Within derivative articles environment can be updated with environment variable as follows:

1.2.2 Updating environment variables

```
env.update_env([
  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)])
```

if input is set to True the variable update will be ignored if the variable is defined in parent environment. If variable is not defined in parent environment, it will be defined and set to value from the command.

if input is set to False update will overwrite variable value if variable exists, if variable is not defined it will define it.

override flags environment variable as changeable by derivative program articles.

1.2.3 Accessing environment variables

```
import projenv
env=projenv.Environ()
env.update_env([
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)])

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

In the fist case (*OUT_FILE* variable), direct access, KeyError exception may be raised, if variable name does not found.
In the second example (*RATE* variable), None value will be returned if not found.

Chapter 2

Environment Tree

Environment files are evaluated in hierarchically. The project tree and its packages are treated as nodes in a tree. Each node can be evaluated and have its own representation of the environment.

2.1 Single Project Environment Tree

At each node, environment is evaluated in the sequence or *envnodes* configuration parameter. By default this means:

1. First *.envpackage.xml*, if available, is read and set.

2. Next, *.envoverride.xml* overrides, if available, is read and set.

As shown in Listing: 2.1 (Page: 5), this behavior could be changed to support different environment's node structure. For example, to support legacy projects using older version of projenv, the following configuration .envconfig.xml can be used:

```
1 <environ>
2 <envnodes>.projectenv, packageenv, personalenv</envnodes>
3 </environ>
```

Listing 2.1: Example for project configuration environment file

The following figure shows a possible use of default configuration.

Figure 2.1 (page 6) 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.

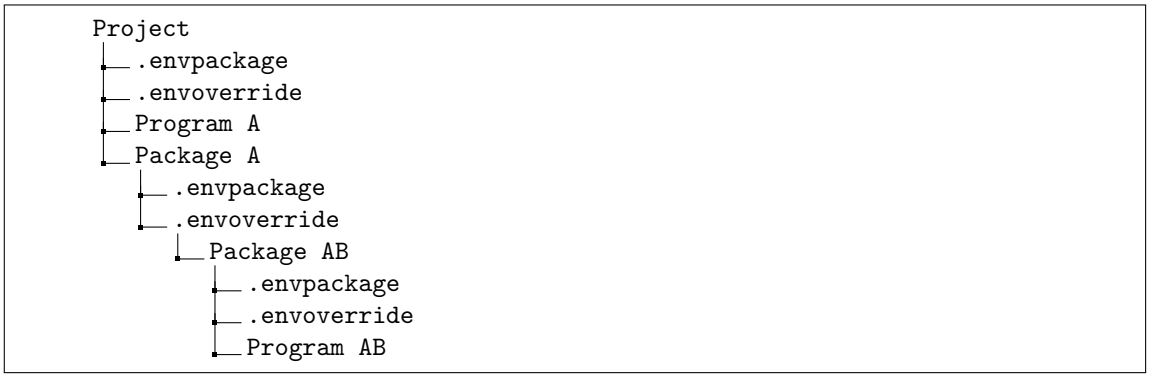

Figure 2.1: Environment tree example

```
<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 2.2: Example for project environment file

Listing 2.2 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 of enclosure is required; < *environ*> can not be in top level of the XML.

2.2 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 2.3: '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 2.4: 'Project B: /Users/me/projs/projb/.envoverride.xml'

Listings 2.4 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 must be set as full path for the installation of the included project. It is therefor best practice to populate real path only in .envoverride.xml and not in .envpackage.xml.

Chapter 3

Best Practices

So many options, so what should one do?

3.1 Naming Parameters

Project Prefix Prefix your parameters with an identifier. Specifically if your projects would need to cooperate (import their environment). In Listings 3.1 (page 10), we have all parameters us 'AC_' as prefix. We also define 'AC_PROJ_PREFIX' that can be used in program to construct parameter name.

Style We recommend following UNIX convention for environment variables. Use uppercase letters separated with underscore. We use this style in all of this document listings.

Drivers and Derivatives For the sake of this discussion we define three types of parameters:

1. *standalone is a parameter that is not dependent on another and is not used by another parameter.*
2. driver is a parameter that other parameters defined by it.
3. *derivative is a parameter that includes a driver in its definitions.*

A parameter can be both a driver and derivative.
Use drivers and derivative parameter definition in such a way that users may personalize the behavior of the system. For example, developers may want to change their own directory structure to fit their own tools.

3.2 .envproject

Dot (.) envproject, although not default in envnodes configuration, good practice to use. It is usually contains parameters that are good for the all projects. You can look at is as your standard parameters to all projects that you produce. In listings 3.1, locations are defined as derivatives of

AC_VAR_BASE. This is useful since users of this project can override that parameter to change to their own structure.

```
<environment>
<environ>
<var name='AC_PROJ_PREFIX' value='AC_' export='True' override='True'/>
<var name='AC_VAR_BASE' value='/var/accord/data/' override='True' export='True'/>
<var name='AC_ENV_NAME' value='.' override='True' export='True'/>
<var name='AC_VAR_LOC' value='${AC_VAR_BASE}${AC_ENV_NAME}/' override='True' export='True'/>
<var name='AC_LOG_LOC' value='${AC_VAR_LOC}/log/' override='True' export='True'/>
<var name='AC_REJ_LOC' value='${AC_VAR_LOC}/rej/' override='True' export='True'/>
<var name='AC_RUN_LOC' value='${AC_VAR_LOC}/run/' override='True' export='True'/>
<var name='AC_IN_LOC' value='${AC_VAR_LOC}/in/' override='True' export='True'/>
<var name='AC_OUT_LOC' value='${AC_VAR_LOC}/out/' override='True' export='True'/>
</environ>
</environment>
```

Listing 3.1: '.envproject.xml example'

3.3 .envpackage

Dot envpackage includes definitions for that are specific to the project or the package. Usually this is kept for things like RPC_PORT or maybe MAIL_SEND_SMTP.

3.4 .envoverride

Dot envoverride provides means to personalize an environment. Users can override .envpackage or .envproject parameters. you may want to exclude *envoverride from your code repository (e.g., add envoverride.xml to .gitignore). Otherwise, users may override each other personalizations.*

Chapter 4

Installation, validation and example program

How to install, validate installation and use the package?

4.1 Installation

To install run following command: pip install projenv

4.2 Validation

Add instruction to run test.py and check unit test cases

4.3 Example

See example of the program using projenv on Github https://github.com/Acrisel/projenv/blob/master/environ/example/examp