Eiffel ECF Primer

Anatomy of an Eiffel Project

Introduction

The ECF file is an XML file which defines a single Eiffel project, with one or more sub-projects called “targets”.

Projects and targets can include more than Eiffel code. They can include C/C++ code as either in-line (directly in Eiffel class method code) or external (C/C++ headers and .c or cpp or .cc files and so on).

The ECF file also contains specifications that control how the project is compiled, but what items (files and so on) participate in the compile process and the final result.

Tags

There are a number of XML <tags> used in the ECF file:

- <system>
    - <description>
    - <target>
        - <root>
        - <file_rule>
            - <include>
            - <exclude>
        - <options>
            - <assertions>
        - <setting>
        - <library>
        - <cluster>

System Tag

Defines an overall Eiffel project as a <system>. The UUID defines a unique ID to help prevent name-clashing when using libraries (see <library> below).

Description Tag

The <description> tag defines a one or multi-line text collection which describes the <system> or other tags (e.g. like <target>). Your best bet to working with <description> tags is to hack (manually edit) the ECF file itself.

Target Tag

ECF files have at least one <target> tag, but can have multiple root node <target> tags. Also, each <target> can have a child <target>, which uses the “extends” attribute. For example:

```
<target name="my eiffel testing demo tests" extends="my eiffel testing demo">
```

Here, the “my_eiffel_testing_demo_tests” is extending (inheriting) the “my_eiffel_testing_demo” target, making it the parent. This means that the child inherits all of the settings and so on of the parent (e.g. <library> and so on).

Root Tag

The <root> tag determines what the <target> root is. If it is set to “all_classes” then the target is a library and cannot be compiled to a binary executable. Otherwise, the <root> defines a root class and root procedure (method) on that class, which is used to “launch” the finalized binary. For example:

`<root all_classes="true"/>` is a library root for its <target>, whereas:

`<root class="APPLICATION" feature="make"/>` defines method “make” of the APPLICATION class as the root creation procedure.

File Rule Tag

File rules tell Eiffel Studio how to manage various file and folders within the <system>

Include Tag

What files and folders are <include> (included) in the <target> of the <system>. For example:

```
<file_rule>
    <include>/testing$</include>
</file_rule>
```

Here, the “testing” folder is being included, which countermands the <exclude> directive (see below).

Exclude Tag

What files and folders are <exclude> (excluded) from the <target> of the <system>? For example:

```
<file_rule>
    <exclude>/CVS$</exclude>
    <exclude>/EIFGENs$</exclude>
    <exclude>/\.git$</exclude>
    <exclude>/\.svn$</exclude>
    <exclude>/testing$</exclude>
</file_rule>
```

Here, folders are being excluded from the <target> of the <system>.

Options Tag

These are <options> that are being applied to the <system> <target>. For example:

```
<option warning="warning" manifest_array_type="mismatch_warning">
    <assertions precondition="true" postcondition="true" check="true" invariant="true" loop="true" supplier_precondition="true"/>
</option>
```

Here, we are handling warnings, manifest array types, and assertion handling.

Assertions Tag

See Options above. The <assertions> tells the compiler whether to pay attention to each type of Design-by-Contract assertin or not. Library assertions are generally not turned on unless you specifically turn then to “true” in a specific <library> (see below).

Setting Tag

Each <system> <target> has certain <setting> items that can be “set”. For example:

```
<setting name="console_application" value="true"/>
<setting name="total_order_on_reals" value="false"/>
<setting name="dead_code_removal" value="feature"/>
```

Library Tag

One of the most important parts of the ECF is telling the <system> <target> universe what libraries are available to the programmer for use in the <system>. For example:

```
<library name="base" location="$ISE_LIBRARY\library\base\base.ecf"/>
<library name="decimal" location="$ISE_LIBRARY\contrib\library\math\decimal\decimal\decimal.ecf"/>
<library name="diff" location="$ISE_LIBRARY\library\diff\diff.ecf"/>
<library name="time" location="$ISE_LIBRARY\library\time\time.ecf"/>
```

Here, we see use of Base, Time, Diff(erence), and Decimal libraries. The Base library has all of the basic classes needed by any Eiffel <system>, starting with class {ANY}. Time, Diff, and Decimal are included for what their names imply.

Cluster Tag

The <cluster> tag is for defining clusters of code. These do not have to be physical folders. They do not have to be a subdirectory under the physical

project directory. Each <cluster> has a name and a file-spec. Mostly, the name of the <cluster> will match the name of the folder, but this is not required.

Moreover, clusters and their root-folders do not have to be recursive in terms of including files from subfolders within the root folder of the cluster. This is done with the “recursive” attribute. For example:

```
<cluster name="src" location=".\src\" recursive="true"/>
```

Here, we see that “src” is the name of a <cluster> where the folder is located at a relative-path of “.\src\” and it includes all of the files in that directory, recursively.