BAE SYSTEMS
INSPIRED WORK

Hello World

Introduction

This document describes an ECOA® "Hello World" example.

"Hello World" has long been used by programmers as an example of "a minimal program" when discussing or teaching programming languages. "Hello World" is often used as first verification that systems and language processors are working correctly.

As a program, "Hello World" can be traced back to Brian Kernighan's work on the C, B and possibly BCPL programming languages (ref.[2]). It appears in Kernighan's *Programming in C: A Tutorial* (ref.[3]) as:

```
main( ) {
      printf("hello, world");
}
```

This document presents the principal user generated artefacts required to create a "Hello World" program using the ECOA. It is assumed that the reader is conversant with the ECOA Architecture Specification (ref.[1]) and the process of defining and declaring ECOA Assemblies, ASCs (components), Modules, and deployments in XML, and then using code generation to produce Module framework (stub) code units and ECOA Container and Platform code.

Aims

This ECOA "*Hello World*" example follows the tradition and is intended to represent a minimum ECOA software system, comprising a single ECOA ASC (component) (ref.[1]), with no provided or required ECOA Services (external interfaces).

ECOA Features Exhibited

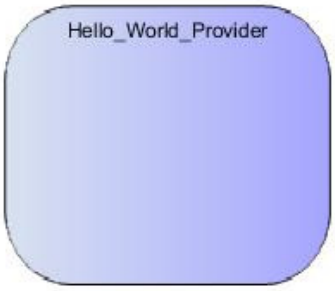
- A minimal ECOA software system "Assembly".
- Use of the ECOA runtime logging API.

Design and Definition

ECOA Assembly Design and Definition

This ECOA "*Hello World*" example Assembly comprises a single ECOA ASC, named "*Hello_World_Provider*".

Figure 1 ECOA "*Hello World*" Assembly Diagram



This ECOA Assembly is defined in an Initial Assembly XML file, and declared in a Final Assembly (or Implementation) XML file (which is practically identical). The Final Assembly XML for the ECOA "*Hello World*" Assembly is as follows (file *hw.impl.composite*):

```
<csa:composite xmlns:csa="http://docs.oasis-open.org/ns/opencsa/sca/200912"
     xmlns:ecoa-sca="http://www.ecoa.technology/sca-extenion-2.0" name="hw"
     targetNamespace="http://www.ecoa.technology">
     <csa:component name="Hello_World_Provider">
          <ecoa-sca:instance componentType="Hello_World_Provider">
               <ecoa-sca:implementation name="Hello_World_Provider"/>
          </ecoa-sca:instance>
     </csa:component>
</csa:composite>
```

The *Hello_World_Provider* ASC is defined in XML as follows (file *Hello_World_Provider.componentType*):

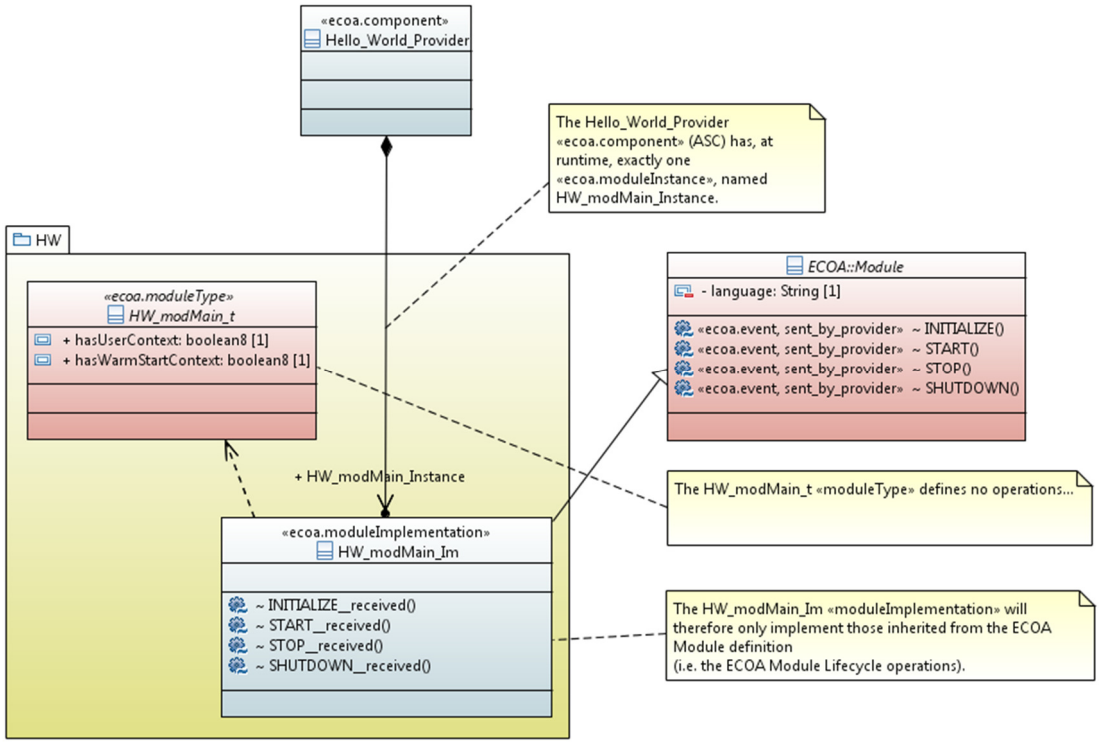
```
<componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912"
     xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:ecoa-
     sca="http://www.ecoa.technology/sca-extenion-2.0">
</componentType>
```

i.e. the ASC definition (the *<componentType>* XML element) has no content, as it provides nor references any ECOA Services, and it defines no ECOA Properties.

ECOA Module Design and Definition

The *Hello_World_Provider* ASC is composed of a single ECOA Module (Module Implementation *HW_modMain_Im* of Module Type *HW_modMain_t*) as illustrated in UML in Figure 2. As always in the ECOA, the Module Implementation implements the Module Lifecycle operations defined by the ECOA (as represented in UML by the abstract class *ECOA::Module*).

Figure 2 ECOA "*Hello World*" Module Design (as UML Class Diagram)



The ASC is declared in XML as follows (file *Hello_World_Provider.impl.xml*):

```
<componentImplementation xmlns="http://www.ecoa.technology/implementation-
     2.0" componentDefinition="Hello_World_Provider">
     <use library="ECOA" />
     <!-- ============================================== -->
     <!-- module type definition -->
     <moduleType name="HW_modMain_t" hasUserContext="false"
                                        hasWarmStartContext="false">
          <operations>
               <!-- None.  It's "hello world" after all... -->
          </operations>
     </moduleType>
     <!-- ============================================== -->
     <!-- module implementation definition -->
     <moduleImplementation  name="HW_modMain_Im"
                               moduleType="HW_modMain_t"
                               language="C" />
     <!-- ============================================== -->
     <!-- module instances -->
    <moduleInstance name="HW_modMain_Instance"
                    implementationName="HW_modMain_Im"
                    relativePriority="1">
    </moduleInstance>
     <!-- ============================================== -->
     <!-- module operation links -->
     <!-- None. -->
</componentImplementation>
```

Note that a naming convention has been adopted in this example whereby the Module is conceived as being declared within a code package named "*HW*" (for "Hello World"). This packaging is illustrated explicitly in the UML class diagram, and expressed in the XML (and subsequently in actual code) using the prefix "*HW_*".

From these definitions and declarations, a single functional code unit will be produced by the code generation process, implementing in code the single concrete class on the UML diagram (*HW_modMain_Im*), and named "*HW_modMain_Im.c*" (assuming the Module Implementation declaration has set the *language* property to "C").

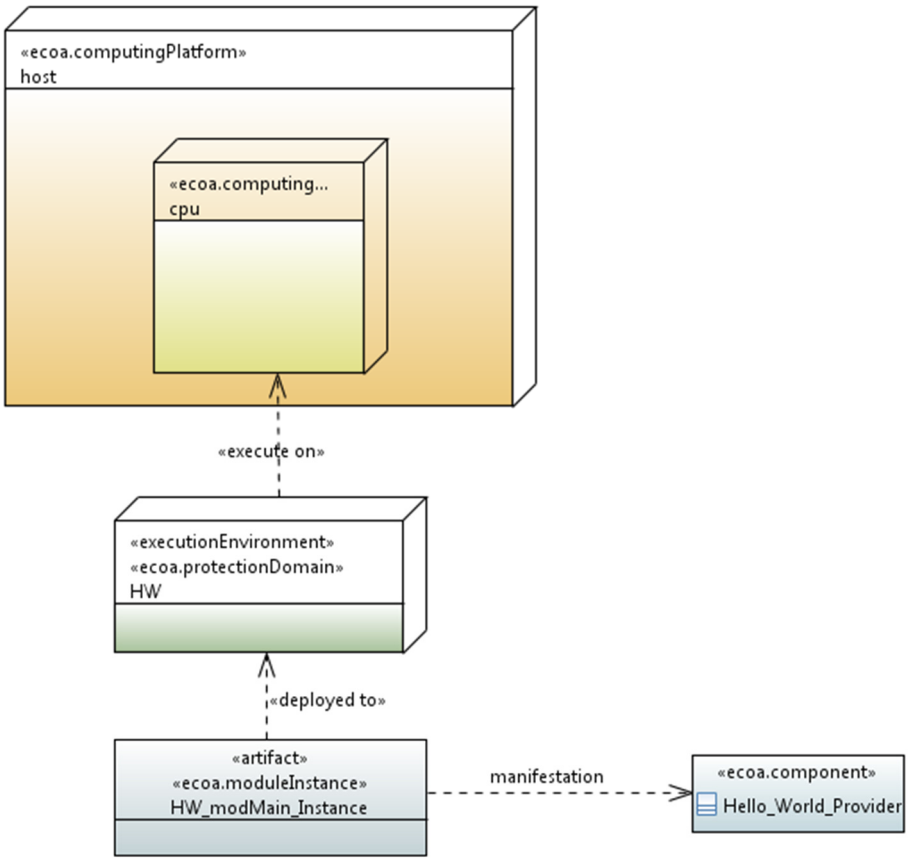
ECOA Deployment Definition

The ECOA "*Hello World*" Assembly is deployed (that is, the declared Module Instances are allocated to ECOA Protection Domains, which are themselves allocated to computing nodes) by the following XML (file *hw.deployment.xml*):

```
<deployment xmlns="http://www.ecoa.technology/deployment-2.0"
     finalAssembly="hw " logicalSystem="hostbased">
     <protectionDomain name="HW">
          <executeOn computingNode="cpu" computingPlatform="host"/>
          <deployedModuleInstance componentName="Hello_World_Provider"
               moduleInstanceName="HW_modMain_Instance"
               modulePriority="50"/>
  </protectionDomain>
  <platformConfiguration faultHandlerNotificationMaxNumber="8"
                                                  computingPlatform="host"/>
</deployment>
```

Thus in this case, the single Module Instance (*HW_modMain_Instance*), which is the runtime manifestation of the *Hello_World_Provider* ASC, is deployed into an ECOA Protection Domain *HW* executing on the Computing Node *cpu* which is part of the (possibly multi-Node) ECOA Computing Platform *host*.

Figure 3 ECOA "*Hello World*" Deployment



Implementation

The ECOA "*Hello World*" is sufficiently trivial – by design! – that the only software code that needs to be added to the code generated framework is to print out the actual message.

In accordance with the ECOA definition, and as illustrated in Figure 2 by the association to the `ECOA::Module` interface class, the Module Implementation is code generated with four stub Module Operation functions, one each for the four ECOA Module Lifecycle operations (`INITIALIZE`, `START`, `STOP`, and `SHUTDOWN`). For the ECOA "*Hello World*" example, we want an output to be made when the whole Assembly starts operating, which is to say, when the `START` operation of the one and only Module Instance (`HM_modMain_Instance`) is called. That `START` operation is implemented by the (C) code function `HW_modMain_Im__START_received` in the (C) code unit `HW_modMain_Im.c`:

```
void HW_modMain_Im__START__received
     (HW_modMain_Im__context* context)
{
     ECOA__log msg;
     //
     msg.current_size = sprintf( msg.data. "  Hello ECOA World!\007" );
     HW_modMain_Im_container__log_info( context, msg );
}
```

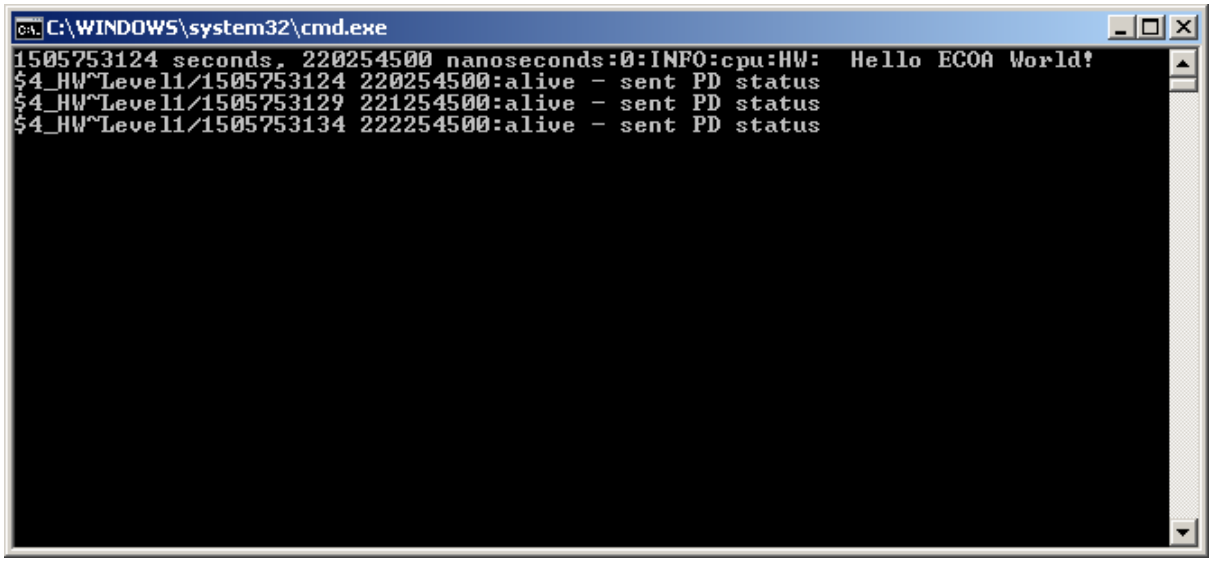
This function uses the ECOA logging API (*log_info*) to output the required message to the user. The message is constructed (as variable *msg* of type *ECOA__log*) by using sprint to copy the message text ("*Hello ECOA World*") into the *msg* variable's *data* field, and set its *current_length* field. *msg* is then passed to the ECOA *log_info* API. The octal character code on the end of the text message ('\007') makes the output device ring the bell when the message is output…

No other code is required, i.e. the other three code generated Module Lifecycle operation stubs remain empty.

Program Output

When the ECOA "*Hello World*" Assembly is built and run, an output similar to Figure 4 should be achieved. The text message is output to the system console, followed by other ECOA Platform logging messages (such as the 10 second periodic "alive" message in the example shown):

Figure 4 ECOA "*Hello World*" in Execution



References

1	European Component Oriented Architecture (ECOA®) Collaboration Programme: Architecture Specification (Parts 1 to 11) "ECOA" is a registered trade mark.
2	"Hello, World!" Program *https://en.wikipedia.org/wiki/Hello_world_program*
3	"Programming in C: A Tutorial" Prof. Brian Kernighan Bell Laboratories internal memorandum, 1974