BAE SYSTEMS
INSPIRED WORK

ExternIF

Introduction

This document describes an ECOA® example of using the ECOA External Interface capability available to ECOA Application Software Components (ASCs).

This document presents information about the principal user generated artefacts required to create an "*ExternIF*" program using the ECOA. It is assumed that the reader is *thoroughly* 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. If not, then let me suggest working through some of the other examples/samples provided, starting with "*Hello World*" and working your way up to "*Pub Sub*".

The ECOA External Interface capability is to allow the creation of single executables containing both ECOA and non-ECOA code, where functions in the latter need to invoke ECOA Module Operations in the former. So first, some points of principal:

- An ECOA External Interface is declared as a link to an ECOA Event Module Operation;
- An External Interface API procedure is created from that declaration on an ECOA Module instance, and made public;
- The External Interface API can then be invoked from non-ECOA software procedures, causing the linked ECOA Event Module Operation to be queued and executed;
- Because the External Interface API invokes ECOA Event Operations using ECOA mechanisms (Operation queuing) the Inversion of Control Principal is not violated by the ASC.

Aims

This ECOA "*ExternIF*" example is intended to show how non-ECOA code functions can invoke ECOA Operations without breaking Inversion of Control (ref.[1]). It also illustrates one example of inter-process communication between ECOA and non-ECOA applications.

ECOA Features Exhibited

- The ECOA External Interface capability.
- Separation of functional behaviour implementation (in an ECOA ASC) from platform specific inter-process and interfacing code.

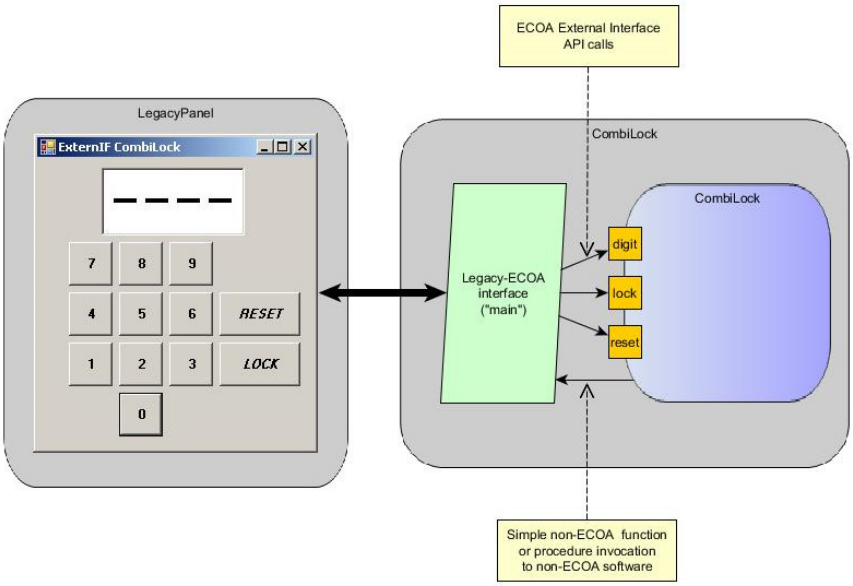
Design and Definition

ECOA Assembly Design and Definition

This ECOA "*ExternIF*" example system represents (very simply) a combination lock and a remote keypad & display. The original *CombiLock* program is to be replaced by one carrying an ECOA ASC implementing the combination lock algorithm. The ECOA ASC is re-usable in other systems, and the combination lock algorithm may be much more complex than that of the original legacy program it replaces.

The ECOA ASC is implemented with ECOA External Interfaces.

Figure 1 ECOA "*ExternIF*" 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 "*ExternIF*" Assembly is as follows (file *CombiLock.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="
     CombiLock" targetNamespace="http://www.ecoa.technology">
     <csa:component name="CombiLock">
          <ecoa-sca:instance componentType="CombiLock">
               <ecoa-sca:implementation name="CombiLock"/>
          </ecoa-sca:instance>
     </csa:component>
</csa:composite>
```

As you see above, only the ECOA ASC declaration appears in the Assembly file. The non-ECOA parts do not.

I will spare much detail and repetition in this description of material presented with other examples, and, relying on your understanding by now of ECOA XML description principals, I am sure you will realise that 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 *CombiLock* ASC is composed of a single ECOA Module defined (as a Module Type) and declared (as a Module Implementation and Instance) following the normal ECOA principals, and having the relevant Event Operations to which the External Interfaces will be mapped. That is, from Figure 1, Event Operations *digit*, *lock* and *reset*:

```
<!-- module CombiLock_modMain_t type definition -->
<moduleType name="CombiLock_modMain_t" hasUserContext="true"
                                       hasWarmStartContext="false">
  <operations>
    <eventReceived name="digit">
      <input name="ch" type="char8"/>
    </eventReceived>
    <eventReceived name="lock"/>
    <eventReceived name="reset"/>
  </operations>
</moduleType>
```

The External Interfaces themselves are declared using *<eventLink>* elements, linking to the Event Operations defined:

```
<!-- Definition of module operation links -->
<eventLink>
  <senders>
    <external operationName="digit" language="C"/>
  </senders>
  <receivers>
    <moduleInstance operationName="digit" instanceName="CombiLock_modMain_Inst"/>
  </receivers>
</eventLink>
<eventLink>
  <senders>
    <external operationName="lock" language="C"/>
  </senders>
  <receivers>
    <moduleInstance operationName="lock" instanceName="CombiLock_modMain_Inst"/>
  </receivers>
</eventLink>
<eventLink>
  <senders>
    <external operationName="reset" language="C"/>
  </senders>
  <receivers>
    <moduleInstance operationName="reset" instanceName="CombiLock_modMain_Inst"/>
  </receivers>
</eventLink>
```

Implementation

The three ECOA Operations *digit*, *lock* and *reset* become, in C code, the ECOA External Interface functions:

```
void CombiLock_digit(const ECOA__char8 ch)
void CombiLock_lock();
void CombiLock_reset();
```

The *Legacy-ECOA interface code* (see Figure 1) is a simple loop that receives control codes (packaged as UDP inter-process messages) from the keypad panel application and makes the appropriate operation call (via those ECOA External Interfaces defined) on the ECOA ASC:

```
for(;;){
  if(( bytes = datagramSocketReceive( pnlSock, pkt )) >= 1 ){
    switch(pkt->buffer[0]){
        case 'R': case 'r':
        CombiLock__reset(); // Invoke the ECOA Module Operation
        break;
      case 'L': case 'l':
        CombiLock__lock(); // Invoke the ECOA Module Operation
        break;
      case '0': case '1': case '2': case '3': case '4':
      case '5': case '6': case '7': case '8': case '9':
        CombiLock__digit(pkt->buffer[0]); // Invoke the ECOA Module Operation
        break;
      default:
        break;
    }
  }
}
```

The loop above is implemented in a function called *panelListener*, which forms a piece of active (i.e. in-parallel) non-ECOA code that must be initiated from the "*main()*" procedure:

```
int main()
{
  pthread_t thrdInf;
  //
  posix_apos_binding__Initialise();

  // Initialise/start all non-ECOA active content (legacy app. and interface)
  pthread_create( &thrdInf, NULL, panelListener, NULL );

  // Continue with "normal" ECOA initialisation...
  pdCombiLock_PD_Controller__Initialise();
    :
}
```

At runtime then we can imagine the two active elements operating in parallel within the one (now extended) Protection Domain, the *panelListener* (*Legacy-ECOA interface*) code, and the ECOA Software Platform (*pdCombiLock_PD_Controller*) and the hosted ASC (*CombiLock*).

In addition of course, the ECOA "*ExternIF*" example needs the keypad panel application (see Figure 1). In the present case, this is provided by a Microsoft Visual Basic program ("*Panel*") which displays a simple key pad and four-digit display. It transmits key presses as UDP inter-process messages, and displays up to four-digit numbers received in like manner.

```
Private lockSock As New Sockets.UdpClient()

Private Sub sendToLock(s As String)
    Dim msg As Byte() = System.Text.Encoding.ASCII.GetBytes(s)
    lockSock.Send(msg, msg.Length, lockAddr)
End Sub

Private Sub Button0_Click(sender As System.Object, e As System.EventArgs) Handles
                                                                    Button0.Click
    sendToLock("0")
End Sub

Private Sub Button1_Click(sender As System.Object, e As System.EventArgs) Handles
                                                                    Button1.Click
    sendToLock("1")
      : etc.
```

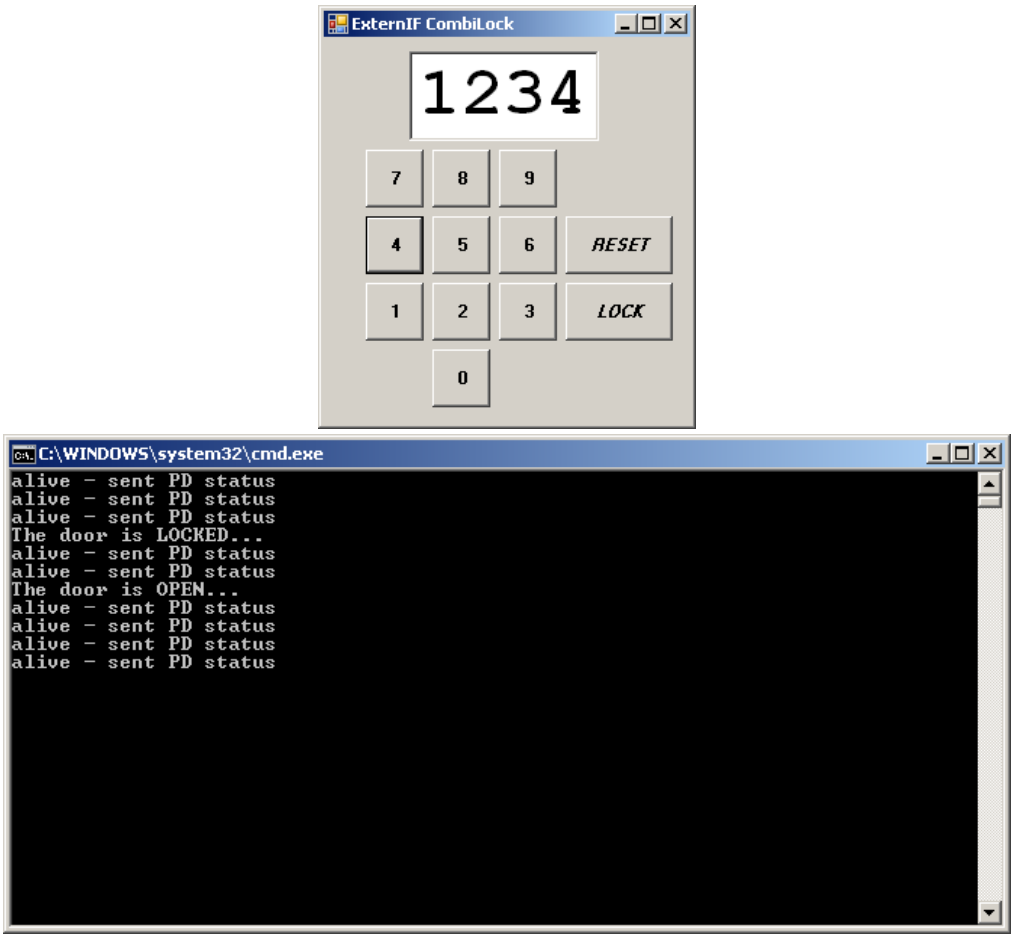
Program Output

The ExternIF Sample comprises two executables:

- the legacy "Panel" keypad simulator, implemented in Visual Basic;
- the "pdCombiLock" ECOA Protection Domain executable.

When the ECOA "*ExternIF*" Assembly is built and run, an output similar to Figure 2 should be achieved. The text message is output to the system console, prefixed by miscellaneous logging data (time stamp, logging type, etc.), and interleaved with any other ECOA Platform logging messages (such as the 10 second periodic "alive" message in the example shown):

Figure 2 ECOA "*ExternIF*" in Execution



References

1	European Component Oriented Architecture (ECOA®) Collaboration Programme: Architecture Specification (Parts 1 to 11) "ECOA" is a registered trade mark.