SOFTWARE COMMUNICATIONS ARCHITECTURE SPECIFICATION

APPENDIX C: CORE FRAMEWORK INTERFACE DEFINITION LANGUAGE

20 August 2015
Version: 4.1

Prepared by:

Joint Tactical Networking Center (JTNC)
33000 Nixie Way
San Diego, CA 92147-5110

REVISION SUMMARY

Version	Revision	Date
Next <Draft>	Initial Draft Release	30 November 2010
Candidate Release	Initial Release	27 December 2011
4.0	ICWG Approved Release	28 February 2012
4.0.1	Incorporated transition to JTNC and applied SCA 4.0 Errata Sheet v1.0	01 October 2012
4.1<DRAFT>	Naming Proposal Changes, Backwards Compatibility Changes, Scalable Components Changes, Scalable Managers Changes, Device Registration Changes	31 December 2014
4.1	New :Process Collocation and Core Affinity Deployment Enhancement, Moved definition of CFPrimitiveTypes and CFPrimitiveSeqTypes to Appendix E Changes: Domain Late Registration, Allocation Properties, Domain Component Type Uniformity, Deployment Data, DomainManager and DeviceManager Instance Level Property Value **ICWG Approved**	20 August 2015

TABLE OF CONTENTS

APPENDIX C CORE FRAMEWORK IDL

C.1 SCOPE

The Core Framework (CF) interfaces are expressed in Interface Definition Language (IDL). Any IDL compiler for the target language of choice may compile the generated IDL.

The CF interfaces are contained in the CF module. The StandardEvent Module contains the standard event types to be passed via the event service.

C.2 CONFORMANCE

N/A

C.3 CONVENTIONS

N/A

C.4 NORMATIVE REFERENCES

N/A

C.5 INFORMATIVE REFERENCES

N/A

C.6 CONDENSED IDL

C.6.1 CF IDL

```
//Source file: CF.idl

#ifndef __CF_DEFINED
#define __CF_DEFINED

/* This file is provided to maintain backward compatibility with
   legacy systems that use CF.idl files. */


/* Base Elements */
#include "CFPrimitiveTypes.idl"
#include "CFPrimitiveSeqTypes.idl"
#include "CFCommonTypes.idl"

/* Specialized Information */
#include "CFSpecializedInfo.idl"

/* Base Application */
#include "CFComponentIdentifier.idl"
#include "CFControllableInterface.idl"
#include "CFLifeCycle.idl"
#include "CFPortAccessor.idl"
```

```
#include "CFPropertySet.idl"
#include "CFTestableInterface.idl"

/* Base Device */
#include "CFAggregateDevice.idl"
#include "CFCapacityManagement.idl"
#include "CFDeviceAttributes.idl"
#include "CFExecutableInterface.idl"
#include "CFLoadableInterface.idl"
#include "CFAdministratableInterface.idl"

/* Framework Control */
#include "CFApplicationManager.idl"
#include "CFDeploymentAttributes.idl"
#include "CFApplicationFactory.idl"
#include "CFComponentRegistry.idl"
#include "CFDomainInstallation.idl"
#include "CFDomainManager.idl"
#include "CFEventChannelRegistry.idl"
#include "CFReleasableManager.idl"

/* Framework Services */
#include "CFComponentFactory.idl"
#include "CFFile.idl"
#include "CFFileManager.idl"
#include "CFFileSystem.idl"

#endif
```

C.6.2 StandardEvent IDL

```
//Source file: StandardEvent.idl

#ifndef __STANDARDEVENT_DEFINED
#define __STANDARDEVENT_DEFINED

/* This file is provided to maintain backward compatibility with
   legacy systems that use StandardEvent.idl files. */

#include "SE_DomainEvent.idl"
#include "SE_StateEvent.idl"

#endif
```

C.7 CORE FRAMEWORK IDL

C.7.1 Base Elements

C.7.1.1 CFPrimitiveTypes IDL

See Appendix E

C.7.1.2 CFPrimitiveSeqTypes IDL

See Appendix E

C.7.1.2.1 CF BooleanSeq

See Appendix E

C.7.1.2.2 CF CharSeq

See Appendix E

C.7.1.2.3 CF DoubleSeq

See Appendix E

C.7.1.2.4 CF FloatSeq

See Appendix E

C.7.1.2.5 CF LongSeq

See Appendix E

C.7.1.2.6 CF LongLongSeq

See Appendix E

C.7.1.2.7 CF OctetSeq

See Appendix E

C.7.1.2.8 CF ShortSeq

See Appendix E

C.7.1.2.9 CF StringSeq

See Appendix E

C.7.1.2.10 CF ULongSeq

See Appendix E

C.7.1.2.11 CF ULongLongSeq

See Appendix E

C.7.1.2.12 CF UShortSeq

See Appendix E

C.7.1.3 CFCommonTypes IDL

```
//Source file: CFCommonTypes.idl

#ifndef __CFCOMMONTYPES_DEFINED
#define __CFCOMMONTYPES_DEFINED

#include "CFPrimitiveTypes.idl"
#include "CFPrimitiveSeqTypes.idl"
#include "CFProperties.idl"

module CF {

   /* This type is an unbounded sequence of octets. */
   typedef CF::OctetSeq OctetSequence;

   /* This type defines a sequence of strings. */
   typedef sequence <string> StringSequence;
```

```
   /* This enum is used to pass error number information in various
      exceptions. Those exceptions starting with "CF_E" map to the POSIX
      definitions.
      The "CF_" has been added to the POSIX exceptions to avoid namespace
      conflicts.  CF_NOTSET is not defined in the POSIX specification.
      CF_NOTSET is an SCA specific value that is applicable for any
      exception when the method specific or standard POSIX error values
      are not appropriate. */
   enum ErrorNumberType {
      CF_NOTSET,
      CF_E2BIG,
      CF_EACCES,
      CF_EAGAIN,
      CF_EBADF,
      CF_EBADMSG,
      CF_EBUSY,
      CF_ECANCELED,
      CF_ECHILD,
      CF_EDEADLK,
      CF_EDOM,
      CF_EEXIST,
      CF_EFAULT,
      CF_EFBIG,
      CF_EINPROGRESS,
      CF_EINTR,
      CF_EINVAL,
      CF_EIO,
      CF_EISDIR,
      CF_EMFILE,
      CF_EMLINK,
      CF_EMSGSIZE,
      CF_ENAMETOOLONG,
      CF_ENFILE,
      CF_ENODEV,
      CF_ENOENT,
      CF_ENOEXEC,
      CF_ENOLCK,
      CF_ENOMEM,
      CF_ENOSPC,
      CF_ENOSYS,
      CF_ENOTDIR,
      CF_ENOTEMPTY,
      CF_ENOTSUP,
      CF_ENOTTY,
      CF_ENXIO,
      CF_EPERM,
      CF_EPIPE,
      CF_ERANGE,
      CF_EROFS,
      CF_ESPIPE,
      CF_ESRCH,
      CF_ETIMEDOUT,
      CF_EXDEV
   };

   /* This exception indicates an invalid file name was passed
```

```
      to a file service operation. The message provides information
      describing why the filename was invalid. */
   exception InvalidFileName {
      CF::ErrorNumberType errorNumber;
      string msg;
   };

   /* This exception indicates an invalid object reference error. */
   exception InvalidObjectReference {
      string msg;
   };

   /* This structure defines a port. */
   struct PortAccessType {
      string portName;
      Object portReference;
   };

   /* This type defines an name/value sequence of PortAccessType
      structures. */
   typedef sequence <PortAccessType> Ports;

   /* This enumeration defines the basic component types. */
   enum ComponentEnumType {
      APPLICATION_COMPONENT,
      MANAGEABLE_APPLICATION_COMPONENT,
      DEVICE_COMPONENT,
      LOADABLE_DEVICE_COMPONENT,
      EXECUTABLE_DEVICE_COMPONENT,
      MANAGEABLE_SERVICE_COMPONENT,
      SERVICE_COMPONENT,
      DEVICE_MANAGER_COMPONENT,
      DOMAIN_MANAGER_COMPONENT,
      APPLICATION_MANAGER_COMPONENT,
      APPLICATION_FACTORY_COMPONENT,
      APPLICATION_COMPONENT_FACTORY_COMPONENT,
      PLATFORM_COMPONENT_FACTORY_COMPONENT
   };

   /* This structure defines the basic elements of a component. */
   struct ComponentType {
      string identifier;
      string profile;
      CF::ComponentEnumType type;
      Object componentObject;
      CF::Ports providesPorts;
      CF::Properties specializedInfo;
   };

   /* This type defines an unbounded sequence of objects. */
   typedef sequence <Object> ObjectSequence;

};
#endif
```

C.7.1.4 CFPlatformTypes IDL

```
//Source file: CFPlatformTypes.idl
#ifndef __CFPLATFORMTYPES_DEFINED
#define __CFPLATFORMTYPES_DEFINED

#include "CFCommonTypes.idl"

module CF {
   /* This structure associates a component with the device
      upon which the component is executing. */
   struct DeviceAssignmentType {
      string componentId;
      string assignedDeviceId;
   };

   /* The sequence provides an unbounded sequence of 0..n of
      DeviceAssignmentType. */
   typedef sequence <DeviceAssignmentType> DeviceAssignmentSequence;

   /* This exception indicates an invalid component profile error. */
   exception InvalidProfile {
   };

   /* This sequence defines a sequence of ComponentType structures */
   typedef sequence <CF::ComponentType> Components;

   /* This exception indicates that the device is not capable of
      the behavior being attempted due to the state the device is in.
      An example of such behavior is allocateCapacity. */
   exception InvalidState {
      string msg;
   };

};
#endif
```

C.7.1.5 CFSpecializedInfoIDL

```
//Source file: CFSpecializedInfo.idl

#ifndef __CFSPECIALIZEDINFO_DEFINED
#define __CFSPECIALIZEDINFO_DEFINED

#include "CFCommonTypes.idl"
#include "CFFileSystem.idl"
#include "CFPlatformTypes.idl"

module CF {

   /* This enumeration defines the basic actions that may be taken against an
      allocation property. */
   enum PropertyActionType {
      CF_EQ,
      CF_NE,
      CF_GT,
      CF_GE,
```

```
      CF_LT,
      CF_LE,
      CF_EXTERNAL
   };

   /* This enumeration defines the basic data types of an allocation property. */
   enum PropertyType {
      CF_BOOLEAN,
      CF_CHAR,
      CF_DOUBLE,
      CF_FLOAT,
      CF_SHORT,
      CF_LONG,
      CF_OBJREF,
      CF_OCTET,
      CF_STRING,
      CF_USHORT,
      CF_ULONG
   };

   /* This string constant is the identifier for the allocation property
      specialized info entry. */
   const string ALLOCATION_PROPS_ID = "ALLOCATION_PROPS";

   /* This structure defines the specialized type for
      the allocation properties associated with a component. The id attribute
      indicates the kind of value and type. The id can
      be an integer string or a unique alphanumeric identifier.
      The value attribute can be any static IDL type or basic type. */
   struct AllocationPropertyType {
      string id;
      CF::StringSequence values;
      CF::PropertyActionType action;
      CF::PropertyType type;
   };

   /* This sequence defines a list of AllocationPropertyType structures. */
   typedef sequence <AllocationPropertyType> AllocationProperties;

   /* This string constant is the identifier for a DeviceManagerComponent string
      identifier type value within a BasePlatformComponent ComponentType's
      specializedInfo. */
   const string DEVICE_MANAGER_ID = "DEVICE_MANAGER_ID";

   /* This string constant is the identifier for a ManagerInfo type within a
      ComponentType's specializedInfo. */
   const string MANAGER_INFO_ID = "MANAGER_INFO";

   /* This string constant is the identifier for ExecutableInterface::ExecutionID_Type
      Value within a ComponentType's specializedInfo. */
   const string EXECUTION_ID = "EXECUTION_ID";

    /* This string constant is the identifier for SPD implementation id  string
       value within a ComponentType's specializedInfo, which is the implementation used
       for the creation of the component. */
    const string IMPLEMENTATION_ID = "IMPLEMENTATION_ID";
```

```
    /* This string constant is the identifier for the device identifier string value
       within a ComponentType' specializedInfo field, which is the device that deployed
       the component. */
    const string TARGET_DEVICE_ID = "TARGET_DEVICE";

    /* This string constant is the identifier for the CF::UsesDeviceAssignmentSequence
       value within a ComponentType' specializedInfo, which denotes the devices used
       by component. */
    const string USES_DEVICE_ID = "USES_DEVICE";

    /* This string constant is the identifier for the CF::Components type value within a
       ComponentType' specializedInfo field. */
    const string COMPONENTS_ID = "COMPONENTS";

    /* This structure associates a component's profile uses device identifier with the
       assigned device identifier. */
    struct UsesDeviceAssignmentType
    {
       string usesDeviceId;
       string assignedDeviceId;
    };

    /* The sequence provides an unbounded sequence of UseDeviceAssignmentType
       elements. */
    typedef sequence <UsesDeviceAssignmentType> UsesDeviceAssignmentSeq;

   /* This structure defines the specialized type for
      the a manager component. */
   struct ManagerInfo {
      CF::FileSystem fileSys;
      CF::Components deployedComponents;
   };
};
#endif
```

C.7.1.6 CFProperties IDL

```
//Source file: CFProperties.idl

#ifndef __CFPROPERTIES_DEFINED
#define __CFPROPERTIES_DEFINED

module CF {

   /* This type is an IDL struct type which can be used to hold any
      basic type or static IDL type. */
   struct DataType {
      /* This attribute indicates the kind of value and type. The id can
         be an integer string or a unique alphanumeric identifier. */
      string id;
      /* This attribute can be any static IDL type or basic
         type. */
      any value;
   };
```

```
   /* This type is an IDL unbounded sequence of CF DataType(s),
      which can be used in defining a sequence of name and value pairs. */
   typedef sequence <DataType> Properties;

   /* This exception indicates a set of properties unknown by the component. */
   exception UnknownProperties {
      CF::Properties invalidProperties;
   };

};
#endif
```

C.7.2 Base Application

C.7.2.1 CFComponentIdentifer IDL

```
//Source file: CFComponentIdentifier.idl

#ifndef __CFCOMPONENTIDENTIFIER_DEFINED
#define __CFCOMPONENTIDENTIFIER_DEFINED

module CF {

   /* This interface provides an identifier attribute for
      a component. */
   interface ComponentIdentifier {
      /* This readonly identifier attribute contains the instance-unique
         identifier for a component. */
      readonly attribute string identifier;
   };
};
#endif
```

C.7.2.2 CFControllableInterface IDL

```
//Source file: CFControllableInterface.idl

#ifndef __CFCONTROLLABLEINTERFACE_DEFINED
#define __CFCONTROLLABLEINTERFACE_DEFINED

#include "CFCommonTypes.idl"

module CF {

   /* This interface provides a common API for the
      control of a software component.  */
   interface ControllableInterface {

      /* This exception indicates that an error occurred during an attempt
         to start the component. The message provides additional information
         describing the reason for the error. */
      exception StartError {
         CF::ErrorNumberType errorNumber;
         string msg;
      };
```

```
      /* This exception indicates that an error occurred during
         an attempt to stop the component. The message provides additional
         information describing the reason for the error. */
      exception StopError {
         CF::ErrorNumberType errorNumber;
         string msg;
      };

      /* This attribute specifies whether the component is started. */
      readonly attribute boolean started;

      /* This operation is provided to command a component implementing
         this interface to start internal processing. */
      void start ()
         raises (CF::ControllableInterface::StartError);

      /* This operation is provided to command a component implementing
         this interface to stop all internal processing. */
      void stop ()
         raises (CF::ControllableInterface::StopError);
   };
};
#endif
```

C.7.2.3 CFLifeCycle IDL

```
//Source file: CFLifeCycle.idl

#ifndef __CFLIFECYCLE_DEFINED
#define __CFLIFECYCLE_DEFINED

#include "CFCommonTypes.idl"

module CF {

   /* This interface defines the generic operations for initializing
      or releasing instantiated component-specific data and/or processing
      elements. */
   interface LifeCycle {

      /* This exception indicates an error occurred during component
         initialization. The messages provide additional information
         describing the reason why the error occurred. */
      exception InitializeError {
         CF::StringSequence errorMessages;
      };

      /* This exception indicates an error occurred during component
         releaseObject. The messages provide additional information
         describing the reason why the error occurred. */
      exception ReleaseError {
         CF::StringSequence errorMessages;
      };

      /* The purpose of this operation is to provide a mechanism
         to set an object to an known initial state. */
```

```
      void initialize ()
         raises (CF::LifeCycle::InitializeError);

      /* The purpose of this operation is to provide a means
         by which an instantiated component may be torn down. */
      void releaseObject ()
         raises (CF::LifeCycle::ReleaseError);
   };
};
#endif
```

C.7.2.4 CFPortAccessor IDL

```
//Source file: CFPortAccessor.idl

#ifndef __CFPORTACCESSOR_DEFINED
#define __CFPORTACCESSOR_DEFINED

module CF {

   interface PortAccessor {

      /* This structure defines a type for information needed to disconnect a
         connection. */
      struct ConnectionIdType {
         string connectionId;
         string portName;
      };

      /* The sequence of ConnectionIdType structures. */
      typedef sequence <ConnectionIdType> Disconnections;

      /* This structure defines a type for information needed to make a
         connection. */
      struct ConnectionType {
         ConnectionIdType portConnectionId;
         Object portReference;
      };

      /* This type defines a sequence of ConnectionType structures. */
      typedef sequence <ConnectionType> Connections;

      /* This structure identifies a port and associated error code
         to be provided in the InvalidPort exception. */
      struct ConnectionErrorType {
         ConnectionIdType portConnectionId;
         unsigned short errorCode;
      };

      /* This exception indicates one of the following errors has occurred in
         the specification of a PortAccessor association. */
      exception InvalidPort {
         ConnectionErrorType invalidConnections;
      };

      /* This operation supplies a component with a sequence of
```

```
         connection information. */
      void connectUsesPorts(
            in CF::PortAccessor::Connections portConnections)
         raises(CF::PortAccessor::InvalidPort);

      /* This operation releases a sequence of uses or
         provides ports from a given connection(s). */
      void disconnectPorts(
            in CF::PortAccessor::Disconnections portDisconnections)
         raises(CF::PortAccessor::InvalidPort );

      /* This operation provides a mechanism to
         obtain a specific provides port(s). */
      void getProvidesPorts(
            inout CF::PortAccessor::Connections portConnections )
         raises(CF::PortAccessor::InvalidPort);
      };

};
#endif
```

C.7.2.5 CFPropertySet IDL

```
//Source file: CFPropertySet.idl

#ifndef __CFPROPERTYSET_DEFINED
#define __CFPROPERTYSET_DEFINED

#include "CFProperties.idl"

module CF {

   /* This interface defines configure and query operations
      to access component properties/attributes. */
   interface PropertySet {

      /* This exception indicates the configuration of a component
         has failed (no configuration at all was done). The message
         provides additional information describing the reason why
         the error occurred. The invalid properties returned indicates
         the properties that were invalid. */
      exception InvalidConfiguration {
         string msg;
         CF::Properties invalidProperties;
      };

      /* This exception indicates the configuration
         of a Component was partially successful. The invalid properties
         returned indicates the properties that were invalid. */
      exception PartialConfiguration {
         CF::Properties invalidProperties;
      };

      /* The purpose of this operation is to allow id/value pair
         configuration properties to be assigned to components
         implementing this interface. */
```

```
      void configure (
            in CF::Properties configProperties
            )
         raises (CF::PropertySet::InvalidConfiguration,
                 CF::PropertySet::PartialConfiguration);

      /* The purpose of this operation is to allow a component
         to be queried to retrieve its properties. */
      void query (
            inout CF::Properties configProperties
            )
         raises (CF::UnknownProperties);
   };
};
#endif
```

C.7.2.6 CFTestableInterface IDL

```
//Source file: CFTestableInterface.idl

#ifndef __CFTESTABLEINTERFACE_DEFINED
#define __CFTESTABLEINTERFACE_DEFINED

#include "CFProperties.idl"

module CF {

   /* This interface defines a set of operations that
      can be used to test component implementations. */
   interface TestableInterface {

      /* This exception indicates the requested testid for a test
         to be performed is not known by the component. */
      exception UnknownTest {
      };

      /* This operation allows components to be blackbox tested.
         This allows Built-In Tests to be implemented which provides
         a means to isolate faults (both software and hardware) within
         the system. */
      void runTest (
            in unsigned long testid,
            inout CF::Properties testValues
            )
         raises (CF::TestableInterface::UnknownTest, CF::UnknownProperties);
   };
};
#endif
```

C.7.3 Base Device

C.7.3.1 CFAggregateDevice IDL

```
//Source file: CFAggregateDevice.idl

#ifndef __CFAGGREGATEDEVICE_DEFINED
```

```
#define __CFAGGREGATEDEVICE_DEFINED

#include "CFCommonTypes.idl"

module CF {

   /* This interface provides aggregate behavior that can
      be used to add and remove devices from a parent device. This interface
      can be provided via inheritance or as a "provides port".
      Child devices use this interface to add or remove themselves from
      parent device when being created or torn-down. */
   interface AggregateDevice {

      /* This readonly attribute contains a list of devices that
         have been added to this device or a sequence length of zero
         if the device has no aggregation relationships with other devices. */
      readonly attribute CF::ObjectSequence devices;

      /* This operation provides the mechanism to associate
         a device with another device. */
      void addDevice (
            in Object associatedDevice,
            in string identifier
            )
         raises (CF::InvalidObjectReference);

      /* This operation provides the mechanism to disassociate
         a device from another device. */
      void removeDevice (
            in string identifier
            )
         raises (CF::InvalidObjectReference);
   };
};
#endif
```

C.7.3.2 CFCapacityManagement IDL

```
//Source file: CFCapacityManagement.idl

#ifndef __CFCAPACITYMANAGEMENT_DEFINED
#define __CFCAPACITYMANAGEMENT_DEFINED

#include "CFProperties.idl"
#include "CFPlatformTypes.idl"

module CF {

   /* This interface defines additional capabilities and an
      attribute for any logical device in the domain. */
   interface CapacityManagement {

      /* This enumeration type defines the device's usage states. */
      enum UsageType {
         IDLE,
         ACTIVE,
```

```
        BUSY
    };

    /* This readonly attribute contains the device's usage
       state.  The usageState indicates whether or not a device is
       actively in use at a specific instant, and if so, whether
       or not it has spare capacity for allocation at that instant. */
    readonly attribute CF::CapacityManagement::UsageType usageState;

    /* This exception returns the capacities that are
       not valid for this device. */
    exception InvalidCapacity {

       /* The message indicates the reason for the invalid capacity. */
       string msg;

       /* The invalid capacities sent to the allocateCapacity operation. */
       CF::Properties capacities;
    };

    /* This operation provides the mechanism to request
       and allocate capacity from the device. */
    boolean allocateCapacity (
          in CF::Properties capacities
          )
       raises (CF::CapacityManagement::InvalidCapacity,
               CF::InvalidState);

    /* This operation provides the mechanism to return
       capacities back to the device, making them available to other
       users. */
    void deallocateCapacity (
          in CF::Properties capacities
          )
       raises (CF::CapacityManagement::InvalidCapacity,
               CF::InvalidState);
    };
};
#endif
```

C.7.3.3 CFDeviceAttributes IDL

```
//Source file: CFDeviceAttributes.idl

#ifndef __CFDEVICEATTRIBUTES_DEFINED
#define __CFDEVICEATTRIBUTES_DEFINED

#include "CFComponentIdentifier.idl"

module CF {

   interface DeviceAttributes : ComponentIdentifier {

      /* This enumeration defines a device's operational states.
         The operational state indicates whether or not the object is
         functioning. */
```

```
      enum OperationalType {
         ENABLED,
         DISABLED
      };

      /* This attribute contains the device's operational
         state. The operational state indicates whether or not the device
         is functioning. */
      readonly attribute CF::DeviceAttributes::OperationalType operationalState;


   };
};
#endif
```

C.7.3.4 CFExecutableInterface IDL

```
//Source file: CFExecutableInterface.idl


#ifndef __CFEXECUTABLEINTERFACE_DEFINED
#define __CFEXECUTABLEINTERFACE_DEFINED

#include "CFPlatformTypes.idl"

module CF {

      /* This interface defines execute and terminate behavior to a device. */
      interface ExecutableInterface

      {

      /* This exception indicates that a process,
         as identified by the processId parameter, does not exist on this
         device. The message provides additional information describing
         the reason for the error. */
      exception InvalidProcess {
         CF::ErrorNumberType errorNumber;
         string msg;
      };

      /* This exception indicates that a function, as identified by
         the input name parameter, hasn't been loaded on this device. */
      exception InvalidFunction {
      };

      /* This type defines a structure to hold the process number or thread id
         within the system. The number is unique to the Processor operating system
         that created the process/thread. */
      struct ExecutionID_Type {
         unsigned long long threadId;
         unsigned long long processId;
         string processCollocation;
         CF::ULongSeq cores;
      };

      /* This exception indicates that input parameters
```

```
      are invalid for the execute operation. Each parameter's ID and
      value must be a valid string type. The invalidParms is a list
      of invalid parameters specified in the execute operation. */
   exception InvalidParameters {
      CF::Properties invalidParms;
   };

   /* This exception indicates the input options are
      invalid on the execute operation. The invalidOptions is a list
      of invalid options specified in the execute operation. */
   exception InvalidOptions {
      CF::Properties invalidOpts;
   };

   /* The STACK_SIZE_ID is the identifier for the ExecutableInterface's
      execute options parameter. */
   const string STACK_SIZE_ID = "STACK_SIZE";

    /* The PRIORITY_ID is the identifier for the ExecutableInterface's
      execute options parameters. */
   const string PRIORITY_ID = "PRIORITY";

    /* The EXEC_DEVICE_PROCESS_SPACE is the identifier for the ExecutableInterface's
      execute options PROCESS_COLLOCATION_ID parameter. */
   const string EXEC_DEVICE_PROCESS_SPACE = "DEVICE";

    /* The PROCESS_COLLOCATION_ID is the identifier for the ExecutableInterface's
      execute options PROCESS_COLLOCATION_ID parameter. */
   const string PROCESS_COLLOCATION_ID = "PROCESS_COLLOCATION";

    /* The ENTRY_POINT_ID is the identifier for the ExecutableInterface's
      execute options parameters. */
   const string ENTRY_POINT_ID = "ENTRY_POINT";

  /* The CORE_AFFINITY_ID is the identifier for the ExecutableInterface's
      execute options parameters. */
   const string CORE_AFFINITY_ID = "CORE_AFFINITY";

   /* This exception indicates that an attempt to invoke
      the execute operation on a device failed. The message provides
      additional information describing the reason for the error. */
   exception ExecuteFail {
      CF::ErrorNumberType errorNumber;
      string msg;
   };

   /* This operation provides the mechanism for terminating
      the execution of a process/thread on a specific device that was
      started up with the execute operation. */
   void terminate (
         in CF::ExecutableInterface::ExecutionID_Type executionId
         )
      raises (CF::ExecutableInterface::InvalidProcess,
              CF::InvalidState);

   /* This operation provides the mechanism for starting up and
```

```
         executing a software process/thread on a device. */
      CF::ExecutableInterface::ExecutionID_Type execute (
            in string filename,
            in CF::Properties options,
            in CF::Properties parameters
            )
         raises (CF::InvalidState,
                 CF::ExecutableInterface::InvalidFunction,
                 CF::ExecutableInterface::InvalidParameters,
                 CF::ExecutableInterface::InvalidOptions,
                 CF::InvalidFileName,
                 CF::ExecutableInterface::ExecuteFail);
   };
};
#endif
```

C.7.3.5 CFLoadableInterface IDL

```
//Source file: CFLoadableInterface.idl

#ifndef __CFLOADABLEINTERFACE_DEFINED
#define __CFLOADABLEINTERFACE_DEFINED

#include "CFFileSystem.idl"
#include "CFPlatformTypes.idl"

module CF {

   /* This interface provides a device with software
      loading and unloading behavior. */
   interface LoadableInterface {

      /* This enumeration defines the type of load to be performed.
         The load types are in accordance with the code element
         within the softpkg element's implementation element. */
      enum LoadType {
         KERNEL_MODULE,
         DRIVER,
         SHARED_LIBRARY,
         EXECUTABLE
      };

      /* This exception indicates that the device
         is unable to load the type of file designated by the
         loadKind parameter. */
      exception InvalidLoadKind {
      };

      /* This exception indicates that an error occurred during
         an attempt to load the device. The message provides additional
         information describing the reason for the error. */
      exception LoadFail {
         CF::ErrorNumberType errorNumber;
         string msg;
      };
```

```
      /* This operation provides the mechanism for loading software
         on a specific device. The loaded software may be subsequently
         executed on the device, if the device is an executable device. */
      void load (
            in CF::FileSystem fs,
            in string fileName,
            in CF::LoadableInterface::LoadType loadKind
            )
         raises (CF::InvalidState,
                 CF::LoadableInterface::InvalidLoadKind,
                 CF::InvalidFileName,
                 CF::LoadableInterface::LoadFail);

      /* This operation provides the mechanism to unload software
         that is currently loaded. */
      void unload (
            in string fileName
            )
         raises (CF::InvalidState,
                 CF::InvalidFileName);
   };
};
#endif
```

C.7.3.6 CFAdministratableInterface IDL

```
//Source file: CFAdministratableInterface.idl

#ifndef __CFADMINISTRATABLEINTERFACE_DEFINED
#define __CFADMINISTRATABLEINTERFACE_DEFINED

module CF {

   interface AdministratableInterface {

      /* This enumeration type defines a device's administrative states.
         The administrative state indicates the permission to use
         or prohibition against using the device. */
      enum AdminType {
         LOCKED,
         SHUTTING_DOWN,
         UNLOCKED
      };

      /* This attribute indicates the permission to use
         or prohibition against using the device. The adminState attribute
         contains the device's admin state value. */
      attribute CF::AdministratableInterface::AdminType adminState;

   };
};
#endif
```

C.7.3.7 CFAggregateDeviceAttributes IDL

```
//Source file: CFAggregateDeviceAttributes.idl
```

```
#ifndef __CFAGGREGATEDEVICEATTRIBUTES_DEFINED
#define __CFAGGREGATEDEVICEATTRIBUTES_DEFINED

#include "CFAggregateDevice.idl"

module CF {

   interface AggregateDeviceAttributes {

     /* This readonly attribute contains the object reference of
        the AggregateDevice with which this device is associated or a nil
        object reference if no association exists. */
      readonly attribute CF::AggregateDevice compositeDevice;

   };
};
#endif
```

C.7.4 Framework Control

C.7.4.1 CFApplicationManager IDL

```
//Source file: CFApplicationManager.idl

#ifndef __CFAPPLICATIONMANAGER_DEFINED
#define __CFAPPLICATIONMANAGER_DEFINED


#include "CFLifeCycle.idl"
#include "CFPortAccessor.idl"
#include "CFPropertySet.idl"
#include "CFTestableInterface.idl"
#include "CFControllableInterface.idl"
module CF {

   /* This interface provides for the control, configuration,
      and status of an instantiated application in the domain. */
   interface ApplicationManager : LifeCycle, PortAccessor, PropertySet,
TestableInterface, ControllableInterface {
      /* This attribute contains the name of the created application.
         The ApplicationFactory interface's create operation name parameter
         provides the name content. */
      readonly attribute string name;
   };
};
#endif
```

C.7.4.2 CFDeploymentAttributes IDL

```
//Source file: CFDeploymentAttributes.idl

#ifndef __CFDEPLOYMENTATTRIBUTES_DEFINED
#define __CFDEPLOYMENTATTRIBUTES_DEFINED

#include "CFPlatformTypes.idl"
```

```
module CF {

   /* This interface provides deployment attributes
      for an application. */
   interface DeploymentAttributes {




      /* This attribute contains the list of application
         Components that have been successfully deployed with this application
         or ApplicationFactory during instantiation or a sequence length of zero
         if no application Components have been deployed. */
      readonly attribute CF::Components deployedComponents;
   };
};
#endif
```

C.7.4.3 CFApplicationFactory IDL

```
//Source file: CFApplicationFactory.idl

#ifndef __CFAPPLICATIONFACTORY_DEFINED
#define __CFAPPLICATIONFACTORY_DEFINED

#include "CFPlatformTypes.idl"
#include "CFSpecializedInfo.idl"

module CF {

   /* This interface class provides an interface to request
      the creation of a specific type of application in the domain.
      The Software Profile determines the type of application that is created
      by the ApplicationFactory. */
   interface ApplicationFactory {

      /* This exception is raised when the parameter
         DeviceAssignmentSequence contains one or more invalid
         application component-to-device assignment(s). */
      exception CreateApplicationRequestError {
         CF::DeviceAssignmentSequence invalidAssignments;
      };

      /* This exception is raised when a create request is valid but
         the application is unsuccessfully instantiated due to internal
         processing errors. The message provides additional information
         describing the reason for the error. */
      exception CreateApplicationError {
         CF::ErrorNumberType errorNumber;
         string msg;
      };

      /* This exception is raised when the input initConfiguration
```

```
            parameter is invalid. */
         exception InvalidInitConfiguration {
            CF::Properties invalidProperties;
         };

         /* This attribute contains the name of the type of application
            that can be instantiated by the ApplicationFactory. */
         readonly attribute string name;

         /* This structure associates a component with a process collocation
            and or processor core. */
         struct ExecutionAffinityType
         {
            string componentId;
            string processCollocation;
            CF::ULongSeq coreAffinities;
         };

         /* The sequence provides an unbounded sequence of ExecutionAffinityType
            elements. */
         typedef sequence <ExecutionAffinityType> ExecutionAffinitySequence;

         /* This operation is used to create an application within
            the system domain. */
         CF::ComponentType create (
               in string name,
               in CF::Properties initConfiguration,
               in CF::DeviceAssignmentSequence deviceAssignments,
               in CF::Properties deploymentDependencies,
               in CF::ApplicationFactory::ExecutionAffinitySequence
                  executionAffinityAssignments
               )
            raises (CF::ApplicationFactory::CreateApplicationError,
                  CF::ApplicationFactory::CreateApplicationRequestError,
                  CF::ApplicationFactory::InvalidInitConfiguration);

   };

};
#endif
```

C.7.4.4 CFComponentRegistry IDL

```
//Source file: CFComponentRegistry.idl

#ifndef __CFCOMPONENTREGISTRY_DEFINED
#define __CFCOMPONENTREGISTRY_DEFINED

#include "CFCommonTypes.idl"

module CF {

   /* This interface is used to manage the registration of
      logical devices and services. */
   interface ComponentRegistry {
```

```
      /* This exception indicates that an internal error has
         occurred to prevent DomainManager registration operations from
         successful completion. */
      exception RegisterError {
         CF::ErrorNumberType errorNumber;
         string msg;
      };

      /* This operation registers the Component and its static provides
         ports. */
      void registerComponent(
            in CF::ComponentType registeringComponent
            )
         raises( CF::InvalidObjectReference, RegisterError );
   };

};
#endif
```

C.7.4.5 CFDomainInstallation IDL

```
//Source file: CFDomainInstallation.idl

#ifndef __CFDOMAININSTALLATION_DEFINED
#define __CFDOMAININSTALLATION_DEFINED

#include "CFPlatformTypes.idl"

module CF {

    interface DomainInstallation {

      /* This exception is raised when an application installation has
         not completed correctly. The message provides additional
         information describing the reason for the error. */
      exception ApplicationInstallationError {
         CF::ErrorNumberType errorNumber;
         string msg;
      };

      exception ApplicationAlreadyInstalled {
      };

      /* This exception indicates the application ID is invalid. */
      exception InvalidIdentifier {
      };

      /* This exception is raised when an application uninstallation has
         not completed correctly. The message provides additional
         information describing the reason for the error. */
      exception ApplicationUninstallationError {
         CF::ErrorNumberType errorNumber;
         string msg;
      };

      /* This operation is used to register new
```

```
        application software in the DomainManager's Domain profile. */
      CF::ComponentType installApplication (
            in string profileFileName
            )
         raises (CF::InvalidProfile,
                 CF::InvalidFileName,
                 CF::DomainInstallation::ApplicationInstallationError,
                 CF::DomainInstallation::ApplicationAlreadyInstalled);

      /* This operation is used to uninstall an
         application and its associated ApplicationFactory from the
         DomainManager. */
      void uninstallApplication (
            in string identifier
            )
         raises (CF::DomainInstallation::InvalidIdentifier,
                 CF::DomainInstallation::ApplicationUninstallationError);

  };

};
#endif
```

C.7.4.6 CFDomainManager IDL

```
//Source file: CFDomainManager.idl

#ifndef __CFDOMAINMANAGER_DEFINED
#define __CFDOMAINMANAGER_DEFINED

#include "CFComponentIdentifier.idl"
#include "CFFileManager.idl"
#include "CFPlatformTypes.idl"

module CF {

   /* This interface is for the control and
      configuration of the radio domain. */
   interface DomainManager : ComponentIdentifier
   {


      /* This readonly attribute contains a profile
         element with a file reference to the DomainManager Configuration
         Descriptor (DMD) profile IDL. */
      readonly attribute string domainManagerProfile;

      /* This readonly attribute is containing a sequence
         of registered managers in the domain. */
      readonly attribute CF::Components managers;

      /* This readonly attribute contains a list of ApplicationComponents that
         have been instantiated in the domain. */
      readonly attribute CF::Components
         applications;
```

```
      /* This readonly attribute contains a list with one
         ApplicationFactoryComponent per application (SAD file and associated
         files) successfully installed. */
      readonly attribute CF::Components
         applicationFactories;

      /* This readonly attribute contains the DomainManager's
         FileManager. */
      readonly attribute CF::FileManager fileMgr;
   };
};
#endif
```

C.7.4.7 CFEventChannelRegistry IDL

```
//Source file: CFEventChannelRegistry.idl

#ifndef __CFEVENTCHANNELREGISTRY_DEFINED
#define __CFEVENTCHANNELREGISTRY_DEFINED

#include "CFCommonTypes.idl"

module CF {

   interface EventChannelRegistry {

      /* This exception indicates that a registering consumer is already
         connected to the specified event channel. */
      exception AlreadyConnected {
      };

      /* This exception indicates that a DomainManager was not able to
         locate the event channel. */
      exception InvalidEventChannelName {
      };

      /* This exception indicates that the unregistering consumer
         was not connected to the specified event channel. */
      exception NotConnected {
      };

      /* This operation is used to connect
         a consumer to a domain's event channel. */
      void registerWithEventChannel (
            in Object registeringObject,
            in string registeringId,
            in string eventChannelName
            )
         raises (CF::InvalidObjectReference,
                 CF::EventChannelRegistry::InvalidEventChannelName,
                 CF::EventChannelRegistry::AlreadyConnected);

      /* This operation is used to disconnect
         a consumer from a domain's event channel. */
      void unregisterFromEventChannel (
            in string unregisteringId,
```

```
            in string eventChannelName
            )
        raises (CF::EventChannelRegistry::InvalidEventChannelName,
                CF::EventChannelRegistry::NotConnected);
   };
};
#endif
```

C.7.4.8 CFFullComponentRegistry IDL

```
//Source file: CFFullComponentRegistry.idl

#ifndef __CFFULLCOMPONENTREGISTRY_DEFINED
#define __CFFULLCOMPONENTREGISTRY_DEFINED

#include "CFComponentRegistry.idl"

module CF {

   /* This interface is used to manage the shutdown of
      logical devices and services. */
   interface FullComponentRegistry : ComponentRegistry {

      /* This exception indicates that an internal error has occurred
         to prevent unregister operations from successful
         completion. The message provides additional information describing the
         reason for the error. */
      exception UnregisterError {
         CF::ErrorNumberType errorNumber;
         string msg;
      };

      /* This operation unregisters the component. */
      void unregisterComponent( in string identifier )
         raises( UnregisterError );
   };

};
#endif
```

C.7.4.9 CFReleasableManager IDL

```
//Source file: CFReleasableManager.idl

#ifndef __CFRELEASABLEMANAGER_DEFINED
#define __CFRELEASABLEMANAGER_DEFINED

module CF {

   /* This interface is used for terminating an instantiated
      device manager. */
   interface ReleasableManager {

      /* This operation provides the mechanism to terminate
         a device manager, unregistering it from the domain manager. */
      void shutdown ();
```

```
   };
};
#endif
```

C.7.5 Framework Services

C.7.5.1 CFComponentFactory IDL

```
//Source file: CFComponentFactory.idl

#ifndef __CFCOMPONENTFACTORY_DEFINED
#define __CFCOMPONENTFACTORY_DEFINED

#include "CFProperties.idl"
#include "CFLifeCycle.idl"

module CF {

   /* A ComponentFactory can be used to create or destroy a Component. */
   interface ComponentFactory : LifeCycle
   {

      /* This exception indicates that the
         createComponent operation failed to create the Component. The
         message is component-dependent, providing additional information
         describing the reason for the error. */
      exception CreateComponentFailure {
         CF::ErrorNumberType errorNumber;
         string msg;
      };

      /* This operation provides the capability to create
         Components. */
      CF::ComponentType createComponent (
            in string componentId,
            in CF::Properties qualifiers
            )
         raises (CF::ComponentFactory::CreateComponentFailure);
   };
};
#endif
```

C.7.5.2 CFFile IDL

```
//Source file: CFFile.idl

#ifndef __CFFILE_DEFINED
#define __CFFILE_DEFINED

#include "CFCommonTypes.idl"

module CF {

   /* This exception indicates a file-related error occurred.
      The message provides information describing the error. */
   exception FileException {
```

```
      CF::ErrorNumberType errorNumber;
      string msg;
   };

   /* This interface provides the ability to read and write files
      residing within a distributed FileSystem. A file can be thought of
      conceptually as a sequence of octets with a current filePointer
      describing where the next read or write will occur. */
   interface File {

      /* This exception indicates an error occurred during a read
         or write operation to a File. The message is component-dependent,
         providing additional information describing the reason for
         the error. */
      exception IOException {
         CF::ErrorNumberType errorNumber;
         string msg;
      };

      /* This exception indicates the file pointer is out of range based upon
         the current file size. */
      exception InvalidFilePointer {
      };

      /* The readonly attribute contains the file name given
         to the FileSystem open/create operation. */
      readonly attribute string fileName;

      /* The readonly attribute contains the file position
         where the next read or write will occur. */
      readonly attribute unsigned long filePointer;

      /* Applications require the read operation in order to retrieve
         data from remote files. */
      void read (
            out CF::OctetSequence data,
            in unsigned long length
            )
         raises (CF::File::IOException);

      /* This operation writes data to the file referenced. */
      void write (
            in CF::OctetSequence data
            )
         raises (CF::File::IOException);

      /* This operation returns the current size of the file. */
      unsigned long sizeOf ()
         raises (CF::FileException);

      /* This operation releases any OE file resources associated
         with the component. */
      void close ()
         raises (CF::FileException);

      /* This operation positions the file pointer where
```

```
         next read or write will occur. */
      void setFilePointer (
            in unsigned long filePointer
            )
         raises (CF::File::InvalidFilePointer, CF::FileException);
   };
};
#endif
```

C.7.5.3 CFFileManager IDL

```
//Source file: CFFileManager.idl

#ifndef __CFFILEMANAGER_DEFINED
#define __CFFILEMANAGER_DEFINED

#include "CFFileSystem.idl"

module CF {

   /* Multiple, distributed FileSystems may be accessed through
      a FileManager. The FileManager interface appears to be a single
      FileSystem although the actual file storage may span multiple
      physical file systems. */
   interface FileManager : FileSystem {

      /* This structure identifies the FileSystems mounted within
         the FileManager. */
      struct MountType {
         string mountPoint;
         CF::FileSystem fs;
      };

      /* This type defines an unbounded sequence of mounted FileSystems. */
      typedef sequence <MountType> MountSequence;

      /* This exception indicates a mount point does not exist within
         the FileManager. */
      exception NonExistentMount {
      };

      /* This exception indicates the FileSystem is a null (nil) object
         reference. */
      exception InvalidFileSystem {
      };

      /* This exception indicates the mount point is already in
         use in the FileManager. */
      exception MountPointAlreadyExists {
      };

      /* This operation associates a FileSystem with a mount point
         (a directory name). */
      void mount (
            in string mountPoint,
            in CF::FileSystem file_System
```

```
            )
         raises (CF::InvalidFileName,
                 CF::FileManager::InvalidFileSystem,
                 CF::FileManager::MountPointAlreadyExists);

      /* This operation removes a mounted FileSystem from
         the FileManager whose mounted name matches the input mountPoint
         name. */
      void unmount (
            in string mountPoint
            )
         raises (CF::FileManager::NonExistentMount);


      /* This operation returns the FileManager's mounted
         FileSystems. */
      CF::FileManager::MountSequence getMounts ();

   };
};
#endif
```

C.7.5.4 CFFileSystem IDL

```
//Source file: CFFileSystem.idl

#ifndef __CFFILESYSTEM_DEFINED
#define __CFFILESYSTEM_DEFINED

#include "CFProperties.idl"
#include "CFFile.idl"

module CF {

   /* This interface defines the operations to enable
      remote access to a physical file system. */
   interface FileSystem {

      /* This exception indicates a set of properties unknown by
         the FileSystem object. */
      exception UnknownFileSystemProperties {
         CF::Properties invalidProperties;
      };

      /* This constant indicates file system size. */
      const string SIZE = "SIZE";

      /* This constant indicates the available space on the file system. */
      const string AVAILABLE_SPACE = "AVAILABLE_SPACE";

      /* This enumerations indicates the type of file entry. A file system can
         have PLAIN or DIRECTORY files and mounted file systems contained
         in a FileSystem. */
      enum FileType {
         PLAIN,
         DIRECTORY,
```

```
      FILE_SYSTEM
   };

   /* This structure indicates the information returned
      for a file. */
   struct FileInformationType {
      string name;
      CF::FileSystem::FileType kind;
      unsigned long long size;
      CF::Properties fileProperties;
   };

   typedef sequence <FileInformationType> FileInformationSequence;

   /* The CREATED_TIME_ID is the identifier for the created time file
      property. */
   const string CREATED_TIME_ID = "CREATED_TIME";

   /* The MODIFIED_TIME_ID is the identifier for the modified time file
      property. */
   const string MODIFIED_TIME_ID = "MODIFIED_TIME";

   /* The LAST_ACCESS_TIME_ID is the identifier for the last access time
      file property. */
   const string LAST_ACCESS_TIME_ID = "LAST_ACCESS_TIME";

   /* This operation removes the file with the given filename. */
   void remove (
         in string fileName
         )
      raises (CF::FileException, CF::InvalidFileName);

   /* This operation copies the source file with the specified
      sourceFileName to the destination file with the specified
      destinationFileName. */
   void copy (
         in string sourceFileName,
         in string destinationFileName
         )
      raises (CF::InvalidFileName, CF::FileException);

   /* This operation checks to see if a file exists based
      on the filename parameter. */
   boolean exists (
         in string fileName
         )
      raises (CF::InvalidFileName);

   /* This operation provides the ability to obtain a list
      of files along with their information in the file system according
      to a given search pattern. */
   CF::FileSystem::FileInformationSequence list (
         in string pattern
         )
      raises (CF::FileException, CF::InvalidFileName);
```

```
      /* This operation creates a new File based upon the provided
         file name and returns a File to the opened file. */
      CF::File create (
            in string fileName
            )
         raises (CF::InvalidFileName, CF::FileException);

      /* This operation opens a file for reading or writing based
         upon the input fileName. */
      CF::File open (
            in string fileName,
            in boolean read_Only
            )
         raises (CF::InvalidFileName, CF::FileException);

      /* This operation creates a file system directory based on
         the directoryName given. */
      void mkdir (
            in string directoryName
            )
         raises (CF::InvalidFileName, CF::FileException);

      /* This operation removes a file system directory based
         on the directoryName given. */
      void rmdir (
            in string directoryName
            )
         raises (CF::InvalidFileName, CF::FileException);

      /* This operation returns file system information to the
         calling client based upon the given fileSystemProperties' ID. */
      void query (
            inout CF::Properties fileSystemProperties
            )
         raises (CF::FileSystem::UnknownFileSystemProperties);
   };
};
#endif
```

C.8 STANDARDEVENT MODULE

The StandardEvent module contains the types necessary for a standard event producer to generate standard SCA events.

C.8.1 SE_DomainEvent IDL

```
//Source file: SE_DomainEvent.idl

#ifndef __SE_DOMAINEVENT_DEFINED
#define __SE_DOMAINEVENT_DEFINED

#include "CFCommonTypes.idl"

module StandardEvent {

   /* This enumeration is utilized in the ComponentChangeEventType to indicate
```

```
      whether an object that has been added to or removed from the domain. */
   enum ComponentChangeType {
      ADDED,
      REMOVED
   };

   /* This structure is used to indicate that an event source has been
      added to or removed from the domain. */
   struct ComponentChangeEventType {
      string               producerId;
      ComponentChangeType  componentChange;
      CF::ComponentType    domainComponent;
   };

};

#endif
```

C.8.2 SE_StateEvent IDL

```
//Source file: SE_StateEvent.idl

#ifndef __SE_STATEEVENT_DEFINED
#define __SE_STATEEVENT_DEFINED

module StandardEvent {

   /* This enumeration is utilized
      in the StateChangeEventType. It is used to identify the category
      of state change that has occurred. */
   enum StateChangeCategoryType {
      ADMINISTRATIVE_STATE_EVENT,
      OPERATIONAL_STATE_EVENT,
      USAGE_STATE_EVENT
   };

   /* This enumeration is utilized
      in the StateChangeEventType. It is used to identify the specific
      states of the event source before and after the state change
      occurred. */
   enum StateChangeType {
      LOCKED,
      UNLOCKED,
      SHUTTING_DOWN,
      ENABLED,
      DISABLED,
      IDLE,
      ACTIVE,
      BUSY
   };

   /* This structure is used to indicate that
      the state of the event source has changed. The event producer
      will send this structure into an event channel on behalf of
      the event source. */
   struct StateChangeEventType {
```

```
      string producerId;
      string sourceId;
      StandardEvent::StateChangeCategoryType stateChangeCategory;
      StandardEvent::StateChangeType stateChangeFrom;
      StandardEvent::StateChangeType stateChangeTo;
   };

};

#endif
```