SOFTWARE COMMUNICATIONS ARCHITECTURE SPECIFICATION

APPENDIX C: CORE FRAMEWORK INTERFACE DEFINITION LANGUAGE (IDL)

01 October 2012
Version: 4.0.1

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

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 */

#include "orb.idl"

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

/* Base Application*/
#include "CFComponentIdentifier.idl"
```

```
#include "CFControllableComponent.idl"
#include "CFLifeCycle.idl"
#include "CFPortAccessor.idl"
#include "CFPropertySet.idl"
#include "CFResource.idl"
#include "CFTestableObject.idl"

/* Base Device*/
#include "CFAggregateDevice.idl"
#include "CFCapacityManagement.idl"
#include "CFDevice.idl"
#include "CFDeviceAttributes.idl"
#include "CFDeviceManagerAttributes.idl"
#include "CFExecutableDevice.idl"
#include "CFLoadableDevice.idl"
#include "CFLoadableObject.idl"
#include "CFManageableComponent.idl"

/* Framework Control*/
#include "CFApplication.idl"
#include "CFApplicationDeploymentData.idl"
#include "CFApplicationFactory.idl"
#include "CFComponentRegistry.idl"
#include "CFDeviceManager.idl"
#include "CFDomainInstallation.idl"
#include "CFDomainManager.idl"
#include "CFEventChannelRegistry.idl"
#include "CFManagerRegistry.idl"
#include "CFManagerRelease.idl"

/* Framework Services*/
#include "CFComponentFactory.idl"
#include "CFComponentManager.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 CFApplicationTypes IDL

```
//File: CFApplicationTypes.idl

#ifndef   CFAPPLICATIONTYPES_DEFINED
#define   CFAPPLICATIONTYPES_DEFINED

#include "CFApplication.idl"
#include "CFApplicationFactory.idl"

module CF {

   /* The ApplicationType defines the elements of an application. */
   struct ApplicationType{
      string name;
      string profile;
      Application app;
   };

   /* The ApplicationFactoryType defines the elements of an application
```

```
      factory. */
   struct ApplicationFactoryType{
      string name;
      string profile;
      ApplicationFactory appFactory;
   };

};
#endif
```

C.7.1.2 CFCommonTypes IDL

```
//Source file: CFCommonTypes.idl

#ifndef   CFCOMMONTYPES_DEFINED
#define   CFCOMMONTYPES_DEFINED

#include "orb.idl"

module CF {

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

   /* This type defines a sequence of strings */
   typedef CORBA::StringSeq 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
   };

   /* The InvalidFileName 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 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;
   };

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

   /* The PortAccessType structure defines a port. */
```

```
   struct PortAccessType {
      string portName;
      Object portReference;
   };

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

   /* This enum defines the basic types of a component */
   enum ComponentEnumType {
      APPLICATION_COMPONENT,
      DEVICE_COMPONENT,
      CF_SERVICE_COMPONENT,
      NON_CF_SERVICE_COMPONENT,
      FRAMEWORK_COMPONENT
   };

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

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

};
#endif
```

C.7.1.3 CFPlatformTypes IDL

```
//Source file: CFPlatformTypes.idl

#ifndef   CFPLATFORMTYPES_DEFINED
#define   CFPLATFORMTYPES_DEFINED
```

```
#include "CFCommonTypes.idl"

module CF {

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

   /* The IDL sequence, DeviceAssignmentSequence, 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.7.1.4 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 {
      /* The id attribute indicates the kind of value and type. The id can
         be an integer string or a unique alphanumeric identifier. */
      string id;
      /* The value attribute can be any static IDL type or basic
         type. */
      any value;
   };

   /* The Properties 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 {

   /* The ComponentIdentifier interface provides an identifier attribute for
```

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

C.7.2.2 CFControllableComponent IDL

```
//Source file: CFControllableComponent.idl

#ifndef   CFCONTROLLABLECOMPONENT_DEFINED
#define   CFCONTROLLABLECOMPONENT_DEFINED

#include "CFCommonTypes.idl"

module CF {

   /* The ControllableComponent interface provides a common API for the
      control of a software component.  */
   interface ControllableComponent {

      /* 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;
      };

      /* The StopError 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;
      };
```

```
      /* The readonly started attribute whether the component is started. */
      readonly attribute boolean started;

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

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

C.7.2.3 CFLifeCycle IDL

```
//Source file: CFLifeCycle.idl

#ifndef   CFLIFECYCLE_DEFINED
#define   CFLIFECYCLE_DEFINED

#include "CFCommonTypes.idl"

module CF {

   /* The LifeCycle 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 the initialize operation is to provide a mechanism
         to set an object to an known initial state. */
      void initialize ()
         raises (CF::LifeCycle::InitializeError);

      /* The purpose of the releaseObject 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;
};

/* The connectUsesPorts supplies a component with a sequence of
connection information. */
void connectUsesPorts(
      in CF::PortAccessor::Connections portConnections)
   raises(CF::PortAccessor::InvalidPort);

/* The disconnectPorts 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 );
```

```
      /* The getProvidesPorts 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 {

   /* The PropertySet 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;
      };

      /* The PartialConfiguration 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 CFResource IDL

```
//Source file: CFResource.idl

/* The following provides the corresponding Unit of Functionality in SCA
   Appendix F for compiler directives in this file (e.g. COMPILER_DIRECTIVE:
   Unit of Functionality).
      - INTERROGABLE: Interrogable
      - CONNECTABLE: Connectable
      - CONFIGURABLE: Configurable
      - TESTABLE: Testable
      - CONTROLLABLE: Controllable
*/

#ifndef   CFRESOURCE_DEFINED
#define   CFRESOURCE_DEFINED
```

```
#include "CFLifeCycle.idl"
#if defined (INTERROGABLE) || defined (V222_COMPAT)
#include "CFComponentIdentifier.idl"
#endif
#if defined (CONNECTABLE   ) || defined (V222_COMPAT)
#include "CFPortAccessor.idl"
#endif
#if defined (CONFIGURABLE  ) || defined (V222_COMPAT)
#include "CFPropertySet.idl"
#endif
#if defined (TESTABLE      ) || defined (V222_COMPAT)
#include "CFTestableObject.idl"
#endif
#if defined (CONTROLLABLE  ) || defined (V222_COMPAT)
#include "CFControllableComponent.idl"
#endif

module CF {
   /* The Resource interface provides a common interface for the control
      and configuration of a software component. */
   interface Resource : LifeCycle
   #if defined (INTERROGABLE) || defined (V222_COMPAT)
    ,ComponentIdentifier
   #endif
   #if defined (CONNECTABLE ) || defined (V222_COMPAT)
    ,PortAccessor
   #endif
   #if defined (CONFIGURABLE ) || defined (V222_COMPAT)
    ,PropertySet
   #endif
   #if defined (TESTABLE ) || defined (V222_COMPAT)
    ,TestableObject
   #endif
   #if defined (CONTROLLABLE ) || defined (V222_COMPAT)
    ,ControllableComponent
   #endif
```

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

C.7.2.7 CFTestableObject IDL

```
//Source file: CFTestableObject.idl

#ifndef   CFTESTABLEOBJECT_DEFINED
#define   CFTESTABLEOBJECT_DEFINED

#include "CFProperties.idl"

module CF {

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

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

      /* The runTest 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::TestableObject::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 {

   /* The AggregateDevice 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 {

      /* The readonly devices 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;

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

      /* The removeDevice 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 {

   /* The CapacityManagement 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
      };

      /* The readonly usageState 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;

      /* The InvalidCapacity 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;
      };

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

      /* The deallocateCapacity 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 CFDevice IDL

```
//Source file: CFDevice.idl

/* The following provides the corresponding Unit of Functionality in SCA
   Appendix F for compiler directives in this file (e.g. COMPILER_DIRECTIVE:
   Unit of Functionality).
      - INTERROGABLE: Interrogable
      - CONNECTABLE: Connectable
      - CONFIGURABLE: Configurable
      - TESTABLE: Testable
      - CONTROLLABLE: Controllable
      - MANAGEABLE: Manageable
```

```
      - ALLOCATABLE: Allocatable
      - AGGREGATABLE: Aggregatable
*/

#ifndef   CFDEVICE_DEFINED
#define   CFDEVICE_DEFINED

#include "CFLifeCycle.idl"
#if defined (INTERROGABLE) || defined (V222_COMPAT)
#include "CFDeviceAttributes.idl"
#endif
#if defined (CONNECTABLE   ) || defined (V222_COMPAT)
#include "CFPortAccessor.idl"
#endif
#if defined (CONFIGURABLE  ) || defined (V222_COMPAT)
#include "CFPropertySet.idl"
#endif
#if defined (TESTABLE      ) || defined (V222_COMPAT)
#include "CFTestableObject.idl"
#endif
#if defined (CONTROLLABLE) || defined (V222_COMPAT)
#include "CFControllableComponent.idl"
#endif
#if defined (MANAGEABLE   ) || defined (V222_COMPAT)
#include "CFManageableComponent.idl"
#endif
#if defined (ALLOCATABLE )  || defined (V222_COMPAT)
#include "CFCapacityManagement.idl"
#endif
#if defined (AGGREGATABLE ) || defined (V222_COMPAT)
#include "CFParentDevice.idl"
#endif

module CF {
      /* The Device interface defines capabilities and attributes for any
         logical Device in the domain. */
      interface Device : LifeCycle
```

```
      #if defined (INTERROGABLE) || defined (V222_COMPAT)
       ,DeviceAttributes
      #endif
      #if defined (CONNECTABLE ) || defined (V222_COMPAT)
       ,PortAccessor
      #endif
      #if defined (CONFIGURABLE ) || defined (V222_COMPAT)
       ,PropertySet
      #endif
      #if defined (TESTABLE ) || defined (V222_COMPAT)
       ,TestableObject
      #endif
      #if defined (CONTROLLABLE ) || defined (V222_COMPAT)
       ,ControllableComponent
      #endif
      #if defined (MANAGEABLE ) || defined (V222_COMPAT)
       ,ManageableComponent
      #endif
      #if defined (ALLOCATABLE ) || defined (V222_COMPAT)
       ,CapacityManagement
      #endif
      #if defined (AGGREGATABLE ) || defined (V222_COMPAT)
       ,ParentDevice
      #endif
      {
      };
};
#endif
```

C.7.3.4 CFDeviceAttributes IDL

```
//Source file: CFDeviceAttributes.idl

#ifndef   CFDEVICEATTRIBUTES_DEFINED
#define   CFDEVICEATTRIBUTES_DEFINED

#include "CFComponentIdentifier.idl"

module CF {
```

```
   interface DeviceAttributes : ComponentIdentifier {

      /* This enumeration type defines a Device's operational states.
         The operational state indicates whether or not the object is
         functioning. */
      enum OperationalType {
         ENABLED,
         DISABLED
      };

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

      /* The softwareProfile attribute is the XML description for this
         logical Device. The softwareProfile attribute contains a profile
         DTD element with a file reference to the SPD profile file. */
      readonly attribute string softwareProfile;

   };
};
#endif
```

C.7.3.5 CFDeviceManagerAttributes IDL

```
//Source file: CFDeviceManagerAttributes.idl

#ifndef   CFDEVICEMANAGERATTRIBUTES_DEFINED
#define   CFDEVICEMANAGERATTRIBUTES_DEFINED

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

module CF {

   /* The DeviceManagerAttributes interface provides attributes for a device
      manager. */
```

```
   interface DeviceManagerAttributes {

      /* The readonly deviceConfigurationProfile attribute contains the
         device manager's profile descriptor. */
      readonly attribute string deviceConfigurationProfile;

      /* The readonly fileSys attribute contains the FileSystem associated
         with this device manager. */
      readonly attribute CF::FileSystem fileSys;

      /* The readonly registeredComponents attribute contains a list of
         Components that have registered with this device manager or a
         sequence length of zero if no components have registered with the
         device manager. */
      readonly attribute CF::Components registeredComponents;
   };
};
#endif
```

C.7.3.6 CFExecutableDevice IDL

```
//Source file: CFExecutableDevice.idl

/* The following provides the corresponding Unit of Functionality in SCA
   Appendix F for compiler directives in this file (e.g. COMPILER_DIRECTIVE:
   Unit of Functionality).
      - INTERROGABLE: Interrogable
      - CONNECTABLE: Connectable
      - CONFIGURABLE: Configurable
      - TESTABLE: Testable
      - CONTROLLABLE: Controllable
      - MANAGEABLE: Manageable
      - ALLOCATABLE: Allocatable
      - AGGREGATABLE: Aggregatable
*/

#ifndef   CFEXECUTABLEDEVICE_DEFINED
#define   CFEXECUTABLEDEVICE_DEFINED
```

```
#include "CFPlatformTypes.idl"
#include "CFLoadableObject.idl"
#include "CFLifeCycle.idl"
#if defined (INTERROGABLE) || defined (V222_COMPAT)
#include "CFDeviceAttributes.idl"
#endif
#if defined (CONNECTABLE   ) || defined (V222_COMPAT)
#include "CFPortAccessor.idl"
#endif
#if defined (CONFIGURABLE  ) || defined (V222_COMPAT)
#include "CFPropertySet.idl"
#endif
#if defined (TESTABLE      ) || defined (V222_COMPAT)
#include "CFTestableObject.idl"
#endif
#if defined (CONTROLLABLE) || defined (V222_COMPAT)
#include "CFControllableComponent.idl"
#endif
#if defined (MANAGEABLE   ) || defined (V222_COMPAT)
#include "CFManageableComponent.idl"
#endif
#if defined (ALLOCATABLE ) || defined (V222_COMPAT)
#include "CFCapacityManagement.idl"
#endif
#if defined (AGGREGATABLE ) || defined (V222_COMPAT)
#include "CFParentDevice.idl"
#endif

module CF {
      /* This interface extends the LifeCycle and Loadable Object interfaces
         by adding execute and terminate behavior to a Device. */
      interface ExecutableDevice : LifeCycle, LoadableObject
      #if defined (INTERROGABLE) || defined (V222_COMPAT)
       ,DeviceAttributes
      #endif
      #if defined (CONNECTABLE ) || defined (V222_COMPAT)
       ,PortAccessor
```

```
#endif
#if defined (CONFIGURABLE ) || defined (V222_COMPAT)
 ,PropertySet
#endif
#if defined (TESTABLE ) || defined (V222_COMPAT)
 ,TestableObject
#endif
#if defined (CONTROLLABLE ) || defined (V222_COMPAT)
 ,ControllableComponent
#endif
#if defined (MANAGEABLE ) || defined (V222_COMPAT)
 ,ManageableComponent
#endif
#if defined (ALLOCATABLE ) || defined (V222_COMPAT)
 ,CapacityManagement
#endif
#if defined (AGGREGATABLE ) || defined (V222_COMPAT)
 ,ParentDevice
#endif
{

/* The InvalidProcess 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 process number within the system. The process
   number is unique to the Processor operating system that created
```

```
      the process. */
   typedef long ProcessID_Type;

   /* The InvalidParameters 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;
   };

   /* The InvalidOptions 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 ExecutableDevice's
      execute options parameter. */
   const string STACK_SIZE_ID = "STACK_SIZE";

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

   /* The ExecuteFail 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;
   };

   /* The terminate 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::ExecutableDevice::ProcessID_Type processId
            )
         raises (CF::ExecutableDevice::InvalidProcess,
                  CF::InvalidState);

      /* The execute operation provides the mechanism for starting up and
         executing a software process/thread on a device. */
      CF::ExecutableDevice::ProcessID_Type execute (
            in string name,
            in CF::Properties options,
            in CF::Properties parameters
            )
         raises (CF::InvalidState,
                  CF::ExecutableDevice::InvalidFunction,
                  CF::ExecutableDevice::InvalidParameters,
                  CF::ExecutableDevice::InvalidOptions,
                  CF::InvalidFileName,
                  CF::ExecutableDevice::ExecuteFail);
   };
};
#endif
```

C.7.3.7 CFLoadableDevice IDL

```
//Source file: CFLoadableDevice.idl

/* The following provides the corresponding Unit of Functionality in SCA
   Appendix F for compiler directives in this file (e.g. COMPILER_DIRECTIVE:
   Unit of Functionality).
      - INTERROGABLE: Interrogable
      - CONNECTABLE: Connectable
      - CONFIGURABLE: Configurable
      - TESTABLE: Testable
      - CONTROLLABLE: Controllable
      - MANAGEABLE: Manageable
      - ALLOCATABLE: Allocatable
      - AGGREGATABLE: Aggregatable
*/
```

```
#ifndef   CFLOADABLEDEVICE_DEFINED
#define   CFLOADABLEDEVICE_DEFINED

#include "CFLoadableObject.idl"
#include "CFLifeCycle.idl"
#if defined (INTERROGABLE) || defined (V222_COMPAT)
#include "CFDeviceAttributes.idl"
#endif
#if defined (CONNECTABLE   ) || defined (V222_COMPAT)
#include "CFPortAccessor.idl"
#endif
#if defined (CONFIGURABLE  ) || defined (V222_COMPAT)
#include "CFPropertySet.idl"
#endif
#if defined (TESTABLE      ) || defined (V222_COMPAT)
#include "CFTestableObject.idl"
#endif
#if defined (CONTROLLABLE) || defined (V222_COMPAT)
#include "CFControllableComponent.idl"
#endif
#if defined (MANAGEABLE   ) || defined (V222_COMPAT)
#include "CFManageableComponent.idl"
#endif
#if defined (ALLOCATABLE ) || defined (V222_COMPAT)
#include "CFCapacityManagement.idl"
#endif
#if defined (AGGREGATABLE ) || defined (V222_COMPAT)
#include "CFParentDevice.idl"
#endif

module CF {
   /* This interface extends the Lifecycle and LoadableObject interfaces by
      adding software loading and unloading behavior to a Device. */
   interface LoadableDevice : LifeCycle, LoadableObject
   #if defined (INTERROGABLE) || defined (V222_COMPAT)
    ,DeviceAttributes
```

```
   #endif
   #if defined (CONNECTABLE ) || defined (V222_COMPAT)
    ,PortAccessor
   #endif
   #if defined (CONFIGURABLE ) || defined (V222_COMPAT)
    ,PropertySet
   #endif
   #if defined (TESTABLE ) || defined (V222_COMPAT)
    ,TestableObject
   #endif
   #if defined (CONTROLLABLE ) || defined (V222_COMPAT)
    ,ControllableComponent
   #endif
   #if defined (MANAGEABLE ) || defined (V222_COMPAT)
    ,ManageableComponent
   #endif
   #if defined (ALLOCATABLE ) || defined (V222_COMPAT)
    ,CapacityManagement
   #endif
   #if defined (AGGREGATABLE ) || defined (V222_COMPAT)
    ,ParentDevice
   #endif
   {
   };
};
#endif
```

C.7.3.8 CFLoadableObject IDL

```
//Source file: CFLoadableObject.idl

#ifndef   CFLOADABLEOBJECT_DEFINED
#define   CFLOADABLEOBJECT_DEFINED

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

module CF {
```

```
/* This interface extends the LoadableDevice interface by adding software
   loading and unloading behavior to a device. */
interface LoadableObject {

   /* This LoadType 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
   };

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

   /* The LoadFail 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;
   };

   /* The load 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 ExecutableDevice. */
   void load (
         in CF::FileSystem fs,
         in string fileName,
         in CF::LoadableObject::LoadType loadKind
         )
      raises (CF::InvalidState,
```

```
                CF::LoadableObject::InvalidLoadKind,
                CF::InvalidFileName,
                CF::LoadableObject::LoadFail);

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

C.7.3.9 CFManageableComponent IDL

```
//Source file: CFManageableComponent.idl

#ifndef   CFMANAGEABLECOMPONENT_DEFINED
#define   CFMANAGEABLECOMPONENT_DEFINED

module CF {

   interface ManageableComponent {

      /* 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
      };

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

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

C.7.3.10 CFParentDevice IDL

```
//Source file: CFParentDevice.idl

#ifndef   CFPARENTDEVICE_DEFINED
#define   CFPARENTDEVICE_DEFINED

#include "CFAggregateDevice.idl"

module CF {

   interface ParentDevice {

     /* The compositeDevice 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 CFApplication IDL

```
//Source file: CFApplication.idl

/* The following provides the corresponding Unit of Functionality in SCA
   Appendix F for compiler directives in this file (e.g. COMPILER_DIRECTIVE:
   Unit of Functionality).
      - CONNECTABLE: Connectable
      - CONFIGURABLE: Configurable
      - TESTABLE: Testable
      - CONTROLLABLE: Controllable
      - APP_INTERROGABLE: Interrogable
```

```
*/

#ifndef  CFAPPLICATION_DEFINED
#define  CFAPPLICATION_DEFINED

/* Note: The following compiler directives are enabled when builiding this
file
   and its included files.
 */
#define CONNECTABLE
#define CONFIGURABLE
#define TESTABLE
#define CONTROLLABLE

#include "CFResource.idl"
#if defined (APP_INTERROGABLE  ) || defined (V222_COMPAT)
#include "CFApplicationDeploymentData.idl"
#endif
#include "CFLifeCycle.idl"
#include "CFPortAccessor.idl"
#include "CFPropertySet.idl"
#include "CFTestableObject.idl"
#include "CFControllableComponent.idl"

module CF {

   /* The Application interface provides for the control, configuration,
      and status of an instantiated application in the domain. */
interface Application : Resource LifeCycle, PortAccessor, PropertySet,
   TestableObject, ControllableComponent
   #if defined (APP_INTERROGABLE) || defined (V222_COMPAT)
    ,ApplicationDeploymentData
   #endif
   {
      /* This attribute is the XML profile information for the application.
         The string value contains a profile element with a file reference
         to the SAD. */
      readonly attribute string profile;

      /* This name attribute contains the name of the created Application.
         The ApplicationFactory interfaces's create operation name parameter
```

```
        provides the name content. */
      readonly attribute string name;
  };
```

```
};
#endif
```

C.7.4.2 CFApplicationDeploymentData IDL

```
//Source file: CFApplicationDeploymentData.idl

#ifndef   CFAPPLICATIONDEPLOYMENTDATA_DEFINED
#define   CFAPPLICATIONDEPLOYMENTDATA_DEFINED

#include "CFPlatformTypes.idl"

module CF {

   /* The ApplicationDeploymentData interface provides deployment attributes
      for an application. */
   interface ApplicationDeploymentData {

      /* The ComponentProcessIdType defines a type for associating
         a component with its process ID. */
      struct ComponentProcessIdType {
         string componentId;
         unsigned long processId;
      };

      /* The ComponentProcessIdSequence type defines an unbounded sequence
         of components' process IDs. */
      typedef sequence <ComponentProcessIdType> ComponentProcessIdSequence;

      /* The ComponentElementType defines a type for associating a component
         with an element (e.g., naming context, implementation ID). */
      struct ComponentElementType {
         string componentId;
         string elementId;
      };

      /* This type is an unbounded sequence of ComponentElementTypes. */
      typedef sequence <ComponentElementType> ComponentElementSequence;
```

```
      /* This attribute contains the list of components' process IDs within
         the Application for components that are executing on a device. */
      readonly attribute
CF::ApplicationDeploymentData::ComponentProcessIdSequence
componentProcessIds;

      /* The componentDevices attribute contains a list of devices which
         each component either uses, is loaded on or is executed on. Each
         component (componentinstantiation element in the Application's
         software profile) is associated with a device. */
      readonly attribute CF::DeviceAssignmentSequence componentDevices;

      /* This attribute contains the list of components' SPD implementation
         IDs within the Application for those components created. */
      readonly attribute
CF::ApplicationDeploymentData::ComponentElementSequence
         componentImplementations;

      /* The registeredComponents attribute contains the list of application
         Components that have registered with this Application or
         ApplicationFactory during instantiation or a sequence length of zero
         if no application Components have registered with this Application
or
         ApplicationFactory. */
      readonly attribute CF::Components registeredComponents;
   };
};
#endif
```

C.7.4.3 CFApplicationFactory IDL

```
//Source file: CFApplicationFactory.idl

#ifndef   CFAPPLICATIONFACTORY_DEFINED
#define   CFAPPLICATIONFACTORY_DEFINED

#include "CFPlatformTypes.idl"
#include "CFApplication.idl"

module CF {
```

```
/* The ApplicationFactory 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;
   };

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

   /* This attribute contains the application software profile that
      the factory uses when creating an application. The string value
      contains a profile element with a file reference to the SAD */
   readonly attribute string softwareProfile;
```

```
      /* The create operation is used to create an Application within
         the system domain. */
      CF::Application create (
            in string name,
            in CF::Properties initConfiguration,
            in CF::DeviceAssignmentSequence deviceAssignments,
            in CF::Properties deploymentDependencies
            )
         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 {

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

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

```
    };

};
#endif
```

C.7.4.5 CFDeviceManager IDL

```
//Source file: CFDeviceManager.idl

/* The following provides the corresponding Unit of Functionality in SCA
    Appendix F for compiler directives in this file (e.g. COMPILER_DIRECTIVE:
    Unit of Functionality).
        - CONNECTABLE: Connectable
        - CONFIGURABLE: Configurable
        - MANAGEMENT_RELEASABLE: Management Releasable
        - INTERROGABLE: Interrogable
*/

#ifndef __CFDEVICEMANAGER_DEFINED
#define __CFDEVICEMANAGER_DEFINED

#include "CFComponentIdentifier.idl"
#if defined (CONNECTABLE__) || defined (V222_COMPAT)
#include "CFPortAccessor.idl"
#endif
#if defined (CONFIGURABLE__) || defined (V222_COMPAT)
#include "CFPropertySet.idl"
#endif
#if defined (MANAGEMENT_RELEASABLE) || defined (V222_COMPAT)
#include "CFManagerRelease.idl"
#endif
#if defined (INTERROGABLE) || defined (V222_COMPAT)
#include "CFDeviceManagerAttributes.idl"
#endif

module CF {
    /* The DeviceManager interface is used to manage a set of logical Devices
        and services. */
    interface DeviceManager : ComponentIdentifier
```

```
    #if defined (CONNECTABLE ) || defined( V222_COMPAT )
     ,PortAccessor
    #endif
    #if defined (CONFIGURABLE ) || defined( V222_COMPAT )
     ,PropertySet
    #endif
    #if defined (MANAGEMENT_RELEASABLE ) || defined( V222_COMPAT )
     ,ManagerRelease
    #endif
    #if defined (INTERROGABLE) || defined( V222_COMPAT )
     ,DeviceManagerAttributes
    #endif
    {
    };
};

#endif
```

C.7.4.6C.7.4.5 CFDomainInstallation IDL

```
//File: CFDomainInstallation.idl

#ifndef   CFDOMAININSTALLATION_DEFINED
#define   CFDOMAININSTALLATION_DEFINED

#include "CFPlatformTypes.idl"
#include "CFApplicationTypes.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;
      };

      /* The installApplication operation is used to register new
         application software in the DomainManager's Domain profile*/
      ApplicationFactoryType installApplication (
            in string profileFileName
            )
         raises (CF::InvalidProfile,
                 CF::InvalidFileName,
                 CF::DomainInstallation::ApplicationInstallationError,
                 CF::DomainInstallation::ApplicationAlreadyInstalled);

      /* The uninstallApplication 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.7.4.7~~C.7.4.6 CFDomainManager IDL

```
//Source file: CFDomainManager.idl

/* The following provides the corresponding Unit of Functionality in SCA
   Appendix F for compiler directives in this file (e.g. COMPILER_DIRECTIVE:
   Unit of Functionality).
      - EVENT_CHANNEL: Event Channel
      - APPLICATION_INSTALLABLE: Application Installable
*/

#ifndef  CFDOMAINMANAGER_DEFINED
#define  CFDOMAINMANAGER_DEFINED

#include "CFComponentIdentifier.idl"
#include "CFFileManager.idl"
#include "CFApplicationTypes.idl"
#include "CFManagerRegistry.idl"
#if defined (CONFIGURABLE  ) || defined (V222_COMPAT)
#include "CFPropertySet.idl"
#endif
#if defined (EVENT_CHANNEL     ) || defined (V222_COMPAT)
#include "CFEventChannelRegistry.idl"
#endif
#if defined (APPLICATION_INSTALLABLE     ) || defined (V222_COMPAT)
#include "CFDomainInstallation.idl"
#endif

module CF {

   /* The DomainManager interface is for the control and
      configuration of the radio domain. */
   interface DomainManager : ComponentIdentifier
   #if defined (CONFIGURABLE) || defined (V222_COMPAT)
    ,PropertySet
   #endif
```

```
#if defined (EVENT_CHANNEL_) || defined (V222_COMPAT)
 ,EventChannelRegistry
#endif
#if defined (APPLICATION_INSTALLABLE) || defined (V222_COMPAT)
 ,DomainInstallation
#endif
{

   /* This type defines an unbounded sequence of Applications. */
   typedef sequence <ApplicationType> ApplicationSeq;

   /* This type defines an unbounded sequence of ApplicationFactories. */
   typedef sequence <ApplicationFactoryType> ApplicationFactorySeq;

   /* This type defines an unbounded sequence of DeviceManagers. */
   typedef sequence <ManagerType> ManagerSeq;

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

   /* The deviceManagers attribute is read-only containing a sequence
      of registered Managers in the domain. */
   readonly attribute CF::DomainManager::ManagerSeq managers;

   /* The applications attribute contains a list of Applications that
      have been instantiated in the domain. */
   readonly attribute CF::DomainManager::ApplicationSeq
      applications;

   /* The readonly applicationFactories attribute contains a list with
      one ApplicationFactory per application (SAD file and associated
      files) successfully installed. */
   readonly attribute CF::DomainManager::ApplicationFactorySeq
      applicationFactories;
```

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

~~C.7.4.8~~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 {
      };

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

      /* The registerWithEventChannel 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);

      /* The unregisterFromEventChannel 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.7.4.9~~C.7.4.8 CFFullComponentRegistry IDL

```
//Source file: CFFullComponentRegistry.idl

#ifndef   CFFULLCOMPONENTREGISTRY_DEFINED
#define   CFFULLCOMPONENTREGISTRY_DEFINED

#include "CFComponentRegistry.idl"

module CF {

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

      /* The unregisterComponent operation unregisters the component. */
      void unregisterComponent( in string identifier )
         raises( CF::UnregisterError );
   };
```

```
};
#endif
```

~~C.7.4.10~~C.7.4.9 CFFullManagerRegistry IDL

```
//Source file: CFFullManagerRegistry.idl

#ifndef   CFFULLMANAGERREGISTRY_DEFINED
#define   CFFULLMANAGERREGISTRY_DEFINED

#include "CFManagerRegistry.idl"

module CF {

   /* The FullManagerRegistry interface extends the ManagerRegistry interface
      with manager unregistration capability. */
   interface FullManagerRegistry : ManagerRegistry {

      /* The unregisterManager operation is used to unregister a
         DeviceManager component from the domain manager. */
      void unregisterManager( in string identifier )
         raises (CF::UnregisterError);
   };

};
#endif
```

~~C.7.4.11~~C.7.4.10 CFManagerRegistry IDL

```
//Source file: CFManagerRegistry.idl

#ifndef   CFMANAGERREGISTRY_DEFINED
#define   CFMANAGERREGISTRY_DEFINED

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

module CF {

   /* The ManagerType structure defines the basic elements of a
```

```
      manager component (e.g. DeviceManagerComponent).   */
   struct ManagerType {
      CF::ComponentType managerComponent;
      CF::Components registeredComponents;
      CF::FileSystem fileSys;
      string profile;
   };

   /* The ManagerRegistry interface is used to manage the registration of
      managers. */
   interface ManagerRegistry {

      /* The registerManager operation is used to register a device
         manager, its device(s), and its services.   */
      void registerManager(
            in CF::ManagerType registeringManager
            )
         raises (CF::InvalidObjectReference,
                 CF::InvalidProfile,
                 CF::RegisterError );
   };

};
#endif
```

~~C.7.4.12~~C.7.4.11 CFManagerRelease IDL

```
//Source file: CFManagerRelease.idl

#ifndef   CFMANAGERRELEASE_DEFINED
#define   CFMANAGERRELEASE_DEFINED

module CF {

   /* The ManagerRelease interface is used for terminating an instantiated
      device manager. */
   interface ManagerRelease {

      /* The shutdown operation provides the mechanism to terminate
```

```
         a DeviceManager, unregistering it from the DomainManager. */
      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"
#if defined (INTERROGABLE) || defined (V222_COMPAT)
#include "CFComponentIdentifier.idl"
#endif

module CF {

   /* A ComponentFactory can be used to create or destroy a Component. */
   interface ComponentFactory : LifeCycle
   #if defined (INTERROGABLE) || defined (V222_COMPAT)
    ,ComponentIdentifier
   #endif
   {

      /* The CreateComponentFailure 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;
      };
```

```
      /* The createComponent 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 CFComponentManager IDL

```
//Source file: CFComponentManager.idl

#ifndef   CFCOMPONENTMANAGER_DEFINED
#define   CFCOMPONENTMANAGER_DEFINED

#include "CFComponentFactory.idl"

module CF {

   /* A Component Manager extends ComponentFactory by adding component
      management capability for a component created by a ComponentFactory. */
   interface ComponentManager : ComponentFactory
     {

      /* There is client side and server side representation
         of a Component. This operation provides the mechanism of releasing
         the component in the operating environment on the server side.   This
         method should only be called once since the server will be
         destroyed.
         The client still has to release its client side reference of the
         Component. true is returned indicating that the component server has
         been successfully released, otherwise false is returned indicating
         that no component was released i.e. server did not exists */
      boolean releaseComponent ( in string componentId);

      /* The getComponent operation provides the capability to return a
```

```
         reference to a component that has already been created */
      CF::ComponentType getComponent (in string componentId);
   };
};
#endif
```

C.7.5.3 CFFile IDL

```
//Source file: CFFile.idl

#ifndef   CFFILE_DEFINED
#define   CFFILE_DEFINED

#include "CFCommonTypes.idl"

module CF {

   /* The CF FileException indicates a file-related error occurred.
      The message provides information describing the error. */
   exception FileException {
      CF::ErrorNumberType errorNumber;
      string msg;
   };

   /* The File 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 {

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

/* The readonly filePointer 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);

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

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

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

/* The setFilePointer 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.7.5.4 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 {

      /* The Mount 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 {
    };

    /* The mount 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);

    /* The unmount 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);


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

};
```

```
};
#endif
```

C.7.5.5 CFFileSystem IDL

```
//Source file: CFFileSystem.idl

#ifndef   CFFILESYSTEM_DEFINED
#define   CFFILESYSTEM_DEFINED

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

module CF {

   /* The FileSystem 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";

      /* The FileType 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
      };
```

```
/* The FileInformationType 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";

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

/* The copy 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);

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

   /* The list 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);

   /* The create 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);

   /* The open 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);

   /* The mkdir operation creates a file system directory based on
      the directoryName given. */
   void mkdir (
         in string directoryName
         )
```

```
         raises (CF::InvalidFileName,CF::FileException);

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

      /* The query 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 STANDARDEVENTMODULE

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

module StandardEvent {

   /* Type SourceCategoryType is an enumeration that is utilized
      in the DomainManagementObjectAddedEventType and
      DomainManagementObjectRemovedEventType.
      It is used to identify the type of object that has been
      added to or removed from the domain. */
   enum SourceCategoryType {
      DEVICE_MANAGER,
      DEVICE,
      APPLICATION_FACTORY,
      APPLICATION,
      SERVICE
   };

   /* Type DomainManagementObjectRemovedEventType is a structure used
      to indicate that the event source has been removed from the domain.
      The event producer will send this structure into an event channel
      on behalf of the event source. */
   struct DomainManagementObjectRemovedEventType {
      string producerId;
      string sourceId;
      string sourceName;
      StandardEvent::SourceCategoryType sourceCategory;
   };

      /* Type DomainManagementObjectAddedEventType is a structure used
```

```
      to indicate that the event source has been added to the domain.
      The event producer will send this structure into an event channel
      on behalf of the event source. */
   struct DomainManagementObjectAddedEventType {
      string producerId;
      string sourceId;
      string sourceName;
      StandardEvent::SourceCategoryType sourceCategory;
      Object sourceIOR;
   };

};

#endif
```

C.8.2 SE_StateEvent IDL

```
//Source file: SE_StateEvent.idl

#ifndef   SE_STATEEVENT_DEFINED
#define   SE_STATEEVENT_DEFINED

module StandardEvent {

   /* Type StateChangeCategoryType is an enumeration that 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
   };

   /* Type StateChangeType is an enumeration that 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
   };

   /* Type StateChangeEventType is a structure 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
```