SCA Training for Developers and Testers

Day 4: Developing Application XML Configuration Files
Day 4 AGENDA

- XML Basics
- Review of Domain Profile
- Software Package Descriptor
- Device Package Descriptor
- Properties Descriptor
- Software Component Descriptor
- Software Assembly Descriptor
- Device Configuration Descriptor
- DomainManager Configuration Descriptor
- Profile Descriptor
XML BASICS
XML - Background

XML - eXtensible Markup Language

- Markup is a way of dividing a continuous string of text (your document) into distinct pieces of data, thus adding context or meaning to raw text
- To apply markup to a document, we insert codes (referred to as tags) into our document

**Document Text – Not Tagged**
Babe Ruth
714 Home Runs
Bats Left Handed
Elected to Hall of Fame 1936

**Document Text - Tagged**
<PlayerInfo>
  <Name> Babe Ruth </Name>
  <HomeRuns> 714 </HomeRuns>
  <Bats> Left </Bats>
  <HallOfFame> 1936 </HallOfFame>
</PlayerInfo>
XML - Background

- HTML (HyperText Markup Language) is an application of SGML
- Developed in 1991 to describe Web pages
- Has a predefined set of tags
- Does not offer the ability to define new elements or attributes
- Does not easily support sophisticated processing
  - A document that doesn’t consist of typical components
  - (headings, paragraphs, lists, tables, and so on)
XML - Background

• XML (eXtensible Markup Language) is the result of a desire to create and implement a next-generation HTML
• Designed to structure data so that it can be easily transferred over a network and be consistently processed by the receiver
• XML can be used to describe data components, records and other data structures - even complex data structures
<?xml version="1.0"?>
<INVENTORY>
  <BOOK>
    <TITLE>The Adventures of Huckleberry Finn</TITLE>
    <AUTHOR>Mark Twain</AUTHOR>
    <PAGES>298</PAGES>
    <PRICE>$5.49</PRICE>
  </BOOK>
  <BOOK>
    <TITLE>Moby-Dick</TITLE>
    <AUTHOR>Herman Melville</AUTHOR>
    <PAGES>605</PAGES>
    <PRICE>$4.95</PRICE>
  </BOOK>
</INVENTORY>
XML - Basics

• Why Use XML?
  – Unlike HTML, XML allows you to define your own tags
  – Provides total control over the structure, features, and content of your documents
  – Context or meaning is attached directly to content because each element can be customized
  – Elements aren’t display oriented; they are information oriented
  – There is no proprietary format
  – The document or file is simply plain text with a .xml extension
XML Basics

• Does XML Replace HTML?
  – Currently, the answer to that question is no. HTML is still the primary language used to tell browsers how to display information on the web.
  – Rather than replacing HTML, XML is currently used in conjunction with HTML and vastly extends the capability of Web pages to:
    – Deliver virtually any type of document
    – Sort, filter, rearrange, find, and manipulate the information in other ways
    – Present highly structured information
XML Basics

• XML provides a three-part alternative to HTML

  – Structured Document
    • A structured document is marked up by an eXtensible Markup Language (XML). It is a document where you place your content

  – Rules Definitions
    • Rules are contained in a Document Type Definition (DTD) file. This file defines elements, attributes, and structure of document

  – Style Sheet
    • This file controls how the XML document displays on screen, such as in a browser
XML Basics

- Besides content, the XML document contains markup comprised of three main components:
  - Elements
  - Attributes
  - Entities
XML Basics - Element

• In XML, Element is a structural component that describes content
  – Elements can contain content
  – Elements can be empty
  – Most elements have three parts
    • start tag
    • content
    • end tag

<Name> Babe Ruth </Name>
XML Basics - Element

- An empty element (one without content) can be created by placing the end-tag immediately after the start tag

  `<HR> </HR>`

- An empty element can also be represented using the special empty-element tag

  `<HR/>`
XML Basics - Element

- Legal Element Names
  - Element names must begin with a Unicode letter, a colon (:) or an underscore (_).
  - Other following characters in XML element names may be Unicode letters, period, hyphen, underscore, colon or digits.
  - There is effectively no restriction on the lengths of characters in XML element names
XML Basics - Element

• Summary of Element Rules
  – The document must have exactly one top-level element (the document element or root element)
  – Elements must be properly nested
  – Each element must have both a start-tag and an end-tag
  – The element-type name in a start-tag must exactly match the name in the corresponding end-tag
  – Element-type names are case-sensitive

  <TITLE>Leaves of Grass</TITLE>   <!-- illegal element -->
XML Basics - Attribute

• XML elements can have attributes. Attributes contain information intended for the application and for extra information associated with an element (like an ID #) used only by programs that read and write this file.

    <GREETING LANGUAGE="ENGLISH">
    <MOVIE SOURCE="WavingHand.mov"/>
    </GREETING>

• The GREETING element has a LANGUAGE attribute, which has the value ENGLISH. The MOVIE element has a SOURCE attribute, which has the value WavingHand.com.
XML Architecture - Attribute

- Elements can possess more than one attribute. However, end tags cannot possess attributes.
- The following example is illegal:

  `<SCRIPT>…</SCRIPT LANGUAGE=“javascript”>`

- Another example: If “Name” is an empty element and it has “FirstName” and “LastName” as attributes, you should declare it as follows:

  `<Name FirstName=“James” LastName=“Bond”/>`
XML Architecture - Attribute

- Summary of Attribute Rules:

  - In XML, all attribute values must be quoted

  - The attribute specifications appear in an element’s start tag

  - There must be a space between the element name and the first attribute specification and between each attribute specification if there is more than one
XML Basics - DTD

• A **well-formed** document is one that conforms to the minimal set of rules that allow the document to be processed by a browser or other program. If a document isn’t well-formed, it can’t be considered an XML document.

• A well-formed XML document can also be **valid**. A valid XML document is a well-formed document that meets two further requirements:
  – The prolog of the document must include a proper document type declaration, which contains a document type definition (DTD) that defines the structure of the document.
  – The rest of the document must conform to the structure defined in the DTD.
Document Type Definition (DTD)

- A DTD provides a list of elements, attributes, notations, and entities contained in a document, as well as their relationship to one another. Also, DTDs...
  
  - specify a set of rules for the structure of a document
  - define exactly what is and is not allowed to appear inside a document
  - ensure the authors will use certain required elements, enter them in correct order, and supply the correct data
Building XML Documents

• Below is a summary of how ?, +, and * can be used with element declarations:

  – ?  Zero or one of the preceding item
  – +  One or more of the preceding item
  – *  Zero or more of the preceding item

• In general, a single parent element can have many children. To indicate that the children must occur in sequence, they are separated by commas. However, each such child element may be suffixed with a ?, +, or * to adjust the number of times it appears in that place in the sequence.
Building XML Documents

• We can indicate that the document author needs to input either one or another element by separating child elements with a vertical bar (|) rather than a comma (,) in the parent’s element declaration.

• For example, the following says that the PAYMENT element must have a single child of type CASH, CREDIT_CARD, or CHECK

  <!ELEMENT PAYMENT (CASH | CREDIT_CARD | CHECK)>
Building XML Documents

• DTDs can contain comments, just like the rest of an XML document. These comments cannot appear inside a declaration, but they can appear outside one.

• For example, the element declaration for the YEAR element might have a comment such as this:

```xml
<!-- Four-digit years like 1998, 1999, or 2000 -->
<!ELEMENT YEAR (#PCDATA)>
```
Elements/Attributes

- Ten Valid Attribute types

  - CDATA Character data - text that is not markup
  - Enumerated A list of possible values from which exactly one will be chosen
  - ID A unique name not shared by any other ID type attribute in the document
  - IDREF The value of an ID type attribute of an element in the document
  - IDREFS Multiple Ids of elements separated by white space
  - ENTITY The name of an entity declared in the DTD
  - ENTITIES Names of multiple entities declared in DTD, separated by white space
  - NMTOKEN An XML name
  - NMTOKENS Multiple XML names separated by white space
  - NOTATION The name of a notation declared in the DTD
Elements/Attributes

• Instead of specifying an explicit default value, an attribute declaration can require the author to provide a value, allow the value to be omitted completely, or even always use the default value.

• These requirements are specified with the three keywords:
  – #REQUIRED
  – #IMPLIED
  – #FIXED
Elements/Attributes

• #REQUIRED - While XML can’t prevent someone from attributing incorrect authorship, it can at least require that authorship is attributed to someone by using #REQUIRED as the default value. For example:

```xml
<!ELEMENT AUTHOR EMPTY>
<!ATTLIST AUTHOR NAME CDATA #REQUIRED>
<!ATTLIST AUTHOR EMAIL CDATA #REQUIRED>
<!ATTLIST AUTHOR EXTENSION CDATA #REQUIRED>
```

• If the parser encounters an `<AUTHOR/>` tag that does not include one or more of these attributes, it returns and error
Elements/Attributes

• #IMPLIED - Sometimes there may not be a good option for a default value, but you do not want to require the author of the document to include a value. In other words, the field is optional. For example:

```xml
<!ELEMENT AUTHOR EMPTY>
<!ATTLIST AUTHOR NAME CDATA #REQUIRED>
<!ATTLIST AUTHOR EMAIL CDATA #REQUIRED>
<!ATTLIST AUTHOR EXTENSION CDATA #IMPLIED>
```
Elements/Attributes

• **#FIXED** - You may want to provide a default value for the attribute without allowing the author to change it. For example, you can require that everyone use a specified value for the COMPANY attribute:

  - `<!ELEMENT AUTHOR EMPTY>`
  - `<!ATTLIST AUTHOR NAME       CDATA #REQUIRED>`
  - `<!ATTLIST AUTHOR EMAIL      CDATA #REQUIRED>`
  - `<!ATTLIST AUTHOR EXTENSION CDATA #IMPLIED>`
  - `<!ATTLIST AUTHOR COMPANY    CDATA #FIXED “RAYTHEON”>`

• Document authors are not required to actually include the fixed attribute in their tags. If they do, however, they must use an identical value. Otherwise, the parser will return an error.
XML Authoring Tools

• Several commercial products exist to help a developer create XML without the worries of syntax and conforming to standards. Two such products are:

  – XMetal
    • from SoftQuad
    • www.softquad.com
  – XML Spy
    • from Altova
    • www.xmlspy.com
Domain Profile Review
Domain Profile Review

- The hardware devices and software components that make up an SCA system are described by a set of files that are collectively referred to as a Domain Profile.

- A Domain Profile contains a set of Software Profiles.

- A Software Profile is either a Software Assembly Descriptor (for applications) or a Software Package Descriptor (for all other software components).

- All the descriptive data is expressed through XML.
Domain Profile Purpose

• Requirements
  – Portability
    • Across Platforms and Platform Classes
  – Interoperability

• Domain Profile
  – Used to deploy Applications and logical Devices into a SCA compliant system
  – Based on the CORBA Components Model
  – Extended for purposes of supporting SCA Requirements for Device Management
Domain Profile Files

- SCA defines 8 file types for the Domain Profile
  - Software Component Descriptor (SCD)
    - Describes a CORBA component’s characteristics
  - Software Package Descriptor (SPD)
    - Describes a component implementation
  - Software Assembly Descriptor (SAD)
    - Describes an application’s deployment characteristics
  - Property File (PRF)
    - Describes properties for a component
  - Device Package Descriptor (DPD)
    - Identifies a class of hardware device and its characteristics
  - Device Configuration Descriptor (DCD)
    - Describes configuration characteristics for a DeviceManager
  - DomainManager Configuration Descriptor (DMD)
    - Describes configuration characteristics for a DomainManager
  - Profile Descriptor
    - Describes a type of file (SAD, SPD, DCD) along with the file name
Domain Profile DTD Relationships

Profile Access
- Software Profile
- Hardware Profile
- HW or SW Profile
- Domain Profile

Domain Profile

Software Assembly Descriptor

Software Component Descriptor

Software Package Descriptor

Device Package Descriptor

Properties Descriptor

Device Cfg Descriptor

DomainMgr Cfg. Desc.
SCA Concept of Application Components

A JTRS application is an "assembly" of 1..n software components.

Core Framework objects responsible for installing, starting up and tearing down applications.

A JTRS application is an "assembly" of 1..n software components.

Part of XML-based "Domain Profile"
Software Package Descriptor
Software Package Descriptor (SPD)

• Purpose
  – Provides implementations for a component
    • general implementation details
    • specific implementation details
      – 1..n Implementations of the software component
  – Indicates id and name of software package
  – Basic Information – Who and What
  – Points to Properties
    • homeproperties – Domain deployment and control properties. Common across developers.
    • properties – component specific deployment and control properties
  – Points to Software Component Descriptor
    • Port and Interface Information
Software Package Descriptor (SPD)

```
<<DTDElement>>
softpkg
  id : ID
  name : CDATA
  type : (sca_compliant | sca_non_compliant) = sca_compliant
  version : CDATA

<<DTDSSequenceGroup>>
softpkg grp
(from softpkg)

<<DTDElement>>
propertyfile
  type : CDATA

<<DTDElement>>
author

<<DTDElement>>
description

<<DTDElement>>
descriptor
  name : CDATA

<<DTDElement>>
implementation
  id : ID
  aepcompliance : (aep_compliant | aep_non_compliant) = aep_compliant

<<DTDElementPCDATA>>
title

<<DTDElementPCDATA>>
usesdevice
  type : CDATA

<<DTDElementPCDATA>>
description
```
<!ELEMENT softpkg
    ( title?,
    , author+,
    , description?,
    , propertyfile?,
    , descriptor?,
    , implementation+,
    , usesdevice*)>

!ATTLIST softpkg
    id ID #REQUIRED
    name CDATA #REQUIRED
    type (sca_compliant | sca_non_compliant) "sca_compliant"
    version CDATA #IMPLIED>
SPD - *softpkg* Element

- All files (*propertyfile, descriptor, implementation, usesdevice*) referenced by the SPD must be in the same directory as the SPD file.
- Element must have (as a minimum) a *propertyfile* which contains the dependency propertyref definitions, and os and processor definitions.
- *Id* attribute is a unique identifier for the package and is a DCE UUID per the standard (adopted by CORBA).
- Name is a user-friendly label for the package.
- *Version* specifies the version number of the component.
SPD Properties

• The union of properties for a SPD and the precedence order are:
  – SPD Implementation Properties - properties specific to one implementation
  – SPD level properties - properties for all implementations
  – SCD properties - component properties for all SPDs. An SCD can be referenced by many SPDs.

• Implementation properties are only used for the initial configuration and creation of a component by the Application Factory.
SPD - propertyfile Element

- The *propertyfile* element indicates the local filename of the property file associated with the software package.

```xml
<!ELEMENT propertyfile (
    localfile
)>
<!ATTLIST propertyfile
type CDATA #IMPLIED>
<!ELEMENT localfile EMPTY>
<!ATTLIST localfile
name CDATA #REQUIRED>

- The property file provides definition of command and control id value pairs used by the SCA Resource configure() and query() interfaces.
SPD - *title* Element

- The *title* element indicates a title for the software component being installed by the SPD.

```xml
<!ELEMENT title (#PCDATA)>```
SPD - *author* Element

- The *author* element indicates the name(s), the company, and the web page of the developer producing the component being installed by the SPD.

```
<!ELEMENT author (name*, company?, webpage?)">

<!ELEMENT name (#PCDATA)>

<!ELEMENT company (#PCDATA)>

<!ELEMENT webpage (#PCDATA)>
```
SPD - *description* Element

- The *description* element presents any pertinent information about the software component being installed by the SPD.

```xml
<!ELEMENT description (#PCDATA)>```
SPD - descriptor Element

- The descriptor element indicates the local filename of the software component descriptor file (SCD) that specifies the interface information for the component being delivered to the system.
- descriptor element is optional since some software components are non-CORBA and will not have a SCD file

```
  <!ELEMENT descriptor ( localfile ) >
  <!ATTLIST descriptor
    name CDATA #IMPLIED>

  <!ELEMENT localfile EMPTY>
  <!ATTLIST localfile
    name CDATA #REQUIRED>
```
SPD - *implementation* Element

- Defines a specific implementation for a loadable and executable code file identified by the `code` element’s `localfile` element.
  
  - *Id* attribute uniquely identifies an implementation with a DCE UUID value
  - multiple *implementation* elements are allowed in the same `softpkg`
  - Each *implementation* is intended to allow the same component to support different types of processors, operating systems, etc.
  - Optional element content items are: `compiler`, `programminglanguage`, `humanlanguage`, `os`, `processor`, and `runtime`. 
<!ELEMENT implementation ( description?, propertyfile?, code, compiler?, programminglanguage?, humanlanguage?, runtime?, ( os | processor | dependency )+, usesdevice* )>

<!ATTLIST implementation id ID #REQUIRED
aepcompliance (aep_compliant | aep_non_compliant) "aep_compliant"
Implementation - code Element

- The *code* element indicates the filename of the code being delivered for this implementation; also, it provides information regarding the type of code module, its execution point, stack size, and priority if necessary.

```xml
<!ELEMENT code
    ( localfile
     , entrypoint?
     , stacksize?
     , priority?)>
<!ATTLIST codetype CDATA #IMPLIED>
<!ELEMENT localfile EMPTY>
<!ATTLIST localfile
    name CDATA #REQUIRED>

<!ELEMENT entrypoint (#PCDATA)>
<!ELEMENT stacksize (#PCDATA)>
<!ELEMENT priority (#PCDATA)>
```
Implementation - code Element, cont’d

• Code type attribute
  – SCA defined valid values:
    – Executable - means to use CF::Device::executeProcess operation to load and execute. This is a main process.
    – Driver and KernelModule - means load only
    – SharedLibrary - dynamic linking
      • Without a code entrypoint element means load only
      • With a code entrypoint element means load and execute using the CF::Device::execute operation. The entrypoint is passed as the CF::Device::execute operation’s functionName parameter.
**Implementation - compiler Element**

- The *compiler* element indicates the compiler used to build the software component being installed by the SPD.
- The *compiler* sub-element is optional

```xml
<!ELEMENT compiler EMPTY>
<!ATTLIST compiler
  name CDATA #REQUIRED
  version CDATA #IMPLIED>
```
Implementation - programminglanguage Element

• The *programminglanguage* element indicates the type of programming language used to build the software component being installed by the SPD.

• The *programminglanguage* sub-element is optional

```xml
<!ELEMENT programminglanguage EMPTY>
<!ATTLIST programminglanguage
    name CDATA #REQUIRED
    version CDATA #IMPLIED>
```
Implementation - humanlanguage Element

- The *humanlanguage* element indicates the human language used in the software component.
- The *humanlanguage* sub-element is optional

```
<!ELEMENT humanlanguage EMPTY>
<!ATTLIST humanlanguage
    name CDATA #REQUIRED>
```
Implementation – os Element

- The *os* element indicates the name and version of the operating system for which the software component is compatible.
  - The *os* attributes must be defined in a property file as allocation properties with names *os_name* and *os_version*
  - The *os* element is always interpreted as a dependency for the software component
  - Valid name values are shown in Attachment 2 to Appendix D

```
<!ELEMENT os EMPTY>
<!ATTLIST os
  name CDATA #REQUIRED
  version CDATA #IMPLIED>
```
Implementation – os Element

os element
<simple id="DCE:80BF17F0-6C7F-11d4-A226-0050DA314CD6" type="string"
    name="os_name" mode="readonly">
    <description> This property identifies the os_name XML allocation property. 
    </description>
    <!-- Valid values for the os_name element are: -->
    <!-- AIX, BSDi, VMS, DigitalUnix, DOS, HPBLS, HPUX, IRIX, -->
    <!-- Linix, LynxOS, MacOS, OS/2, AS/400, MVS, SCO CMW, -->
    <!-- SCO ODT, Solaris, SunOS, UnixWare, VxWorks, Win95, WinNT -->
    <!-- pSOS, RTXC -->
    <!-- The os_name values are case sensitive. -->
    <value></value>
    <kind kindtype="allocation"/>
    <action type="eq"/>
</simple>
Implementation - processor Element

- The *processor* element indicates the processor or processor family for which the software component is compatible.
- The *processor* element is always interpreted as a dependency for the software component.
- Valid name values are shown in Attachment 2 to Appendix D.

```xml
<!ELEMENT processor EMPTY>
<!ATTLIST processor
  name CDATA #REQUIRED>
```
Processor Element

processor element
  <simple id="DCE:9B445600-6C7F-11d4-A226-0050DA314CD6" type="string"
    name="processor_name" mode="readonly">
    <description> This property identifies the processor_name XML allocation property. </description>
  </simple>

<!-- Valid values for the processor_name element are: -->
<!-- x86, mips, alpha, ppc, sparc, 680x0, vax, AS/400, S/390, -->
<!-- ppcG3, ppcG4, ppcG5, C5x, C6x, ADSP21xx -->
<!-- The processor_name values are case sensitive. -->
  <value></value>
  <kind kindtype="allocation"/>
  <action type="eq"/>
Implementation - runtime Element

- The *runtime* element indicates a runtime (for example, a Java VM) required by the software component implementation.
- The *runtime* sub-element is optional

```xml
<!ELEMENT runtime EMPTY>
<!ATTLIST runtime
  name CDATA #REQUIRED
  version CDATA #IMPLIED>
```
**Implementation - dependency Element**

- The *dependency* element indicates the dependent relationships between the component being deployed and other file-load dependencies in the SCA compliant system.

- Used to specify **library dependencies** that need to be loaded before an implementation code is loaded.
Implementation - dependency Element cont’d

- DomainManager will use the softpkgref dependency definitions to assure that components necessary for proper operation are present and loaded prior to the main component.

- DomainManager will use the property ref dependency definitions to select the appropriated Device in the Domain to host the main component. If the propertyref dependency is a allocation type the selected Device will be allocated (allocateCapacity call).

```xml
<!ELEMENT dependency (softpkgref - software library dependency | propertyref - device allocation dependency)>
<!ATTLIST dependency type CDATA #REQUIRED - descriptive information of the type of dependency>
```
Dependency - propertyref Element

- The propertyref element provides the property id of a simple allocation property (from a propertyfile) and a value to be used in connection with that property. When used as a sub-element of the dependency element, the id and value are used by the DomainManager to perform the dependency check.

```xml
<!ELEMENT propertyref EMPTY>
<!ATTLIST propertyref
  refid CDATA #REQUIRED -allocation property's DCE UUID
  value CDATA #REQUIRED> -value to be used for operation
```
**Dependency - softpkgref Element**

- The `softpkgref` element provides a reference to an external software package.
- `Implref refid` provides the unique identifier of a particular implementation within the specified `softpkg` descriptor

```
<!ELEMENT softpkgref (localfile - reference to localfile, implref? - reference to softpkg implementation ) >
<!ELEMENT implref EMPTY>
<!ATTLIST implref refid CDATA #REQUIRED>
```
The *usesdevice* element indicates the following:

- describes uses relationships this component has with logical Devices in the domain and the capacity needed from the logical Devices
- specifies components that exist in the Domain, which the DomainManager is aware of, when a component is being loaded/created

```xml
<!ELEMENT usesdevice
  ( propertyref+ )>
<!ATTLIST usesdevice
  id ID #REQUIRED
  type CDATA #REQUIRED
```
Device Package Descriptor
Device Package Descriptor (DPD)

- Device Package Descriptor contains elements that describe Hardware specific information.

- Information from a DPD used by operator / maintainer or a Human Computer Interface.
Device Package Descriptor (DPD)
<!ELEMENT devicepkg (title?, author+, description?, hwdeviceregistration)>

<!ATTLIST devicepkg
    id ID #REQUIRED
    name CDATA #REQUIRED
    version CDATA #IMPLIED>
DPD

- The hwdeviceregistration element provides device-specific information for a hardware device. At a minimum, the hwdeviceregistration element must include a description, the manufacturer, the model number and the device’s hardware class(es).

```xml
<!ELEMENT hwdeviceregistration (
  propertyfile?,
  description,
  manufacturer,
  modelnumber,
  deviceclass,
  childhwdevice*)>
<!ATTLIST hwdeviceregistration
  id ID #REQUIRED
  name CDATA #REQUIRED
  version CDATA #IMPLIED>
```
DPD - modelnumber Element

- The modelnumber element is used to indicate the manufacture's model number, for the device being installed.

<!ELEMENT modelnumber (#PCDATA)>
DPD - deviceclass Element

- The deviceclass element is used to identify one or more hardware classes that make up the device being installed (e.g., RF, Modem, I/O, as defined in SCA section 4.2.2 HWModule(s) Class Structure).

```xml
<!ELEMENT deviceclass
    ( class+ )>

<!ELEMENT class (#PCDATA)>
```
DPD - childhwdevice Element

- The childhwdevice element indicates additional device-specific information for hardware devices that make up the root or parent hardware device registration. An example of childhwdevice would be a radio's RF module that has receiver and exciter functions within it. In this case, a CF Device representing the RF module itself would be a parent Device with its DPD, and the receiver and exciter are child devices to the module.

<!ELEMENT childhwdevice
    ( hwdeviceregistration
    | devicepkgref
    )>
DPD - devicepkgref Element

- The devicepkgref element is used to indicate the local filename of a Device Package Descriptor file pointed to by Device Package Descriptor (e.g., a devicepkg within a devicepkg).

```xml
<!ELEMENT devicepkgref (localfile)>
<!ATTLIST devicepkgref type DATA IMPLIED>
```
Properties Descriptor
Properties Descriptor (PRF)

Introduction

- The Properties Descriptor details component and device property settings

- Described properties will be used for dependency checking

- Described properties may also be used as values for the configure() and query() interfaces of the CF::Resource component
Properties Descriptor (PRF)

<<DTDElement>>
properties

<<DTDSequenceGroup>>
properties_grp_grp
(from properties_grp)

<<DTDElementPCDATA>>
description

<<DTDChoiceGroup>>
properties_grp_grp
(from properties_grp)

<<DTDElement>>
simple

<<DTDElement>>
simplesequence

<<DTDElement>>
test

<<DTDElement>>
struct

<<DTDElement>>
structsequence
Properties Descriptor (PRF)

• Purpose
  – To define a set of properties
  • Property Types
    – Simple
    – Simple Sequence
    – Test
    – Stuct
    – structsequence
  • Property Kinds
    – Configuration - describes readable and writeable properties for a component
    – Test - describes test properties for a component
    – Allocation - describes capacity type properties for Logical Devices
    – Executable - describes user defined executable parameters for main programs
    – Factory Parameters - describes ResourceFactory create options parameters
Properties DTD

- Multiple properties sub-elements can be within one properties element

```xml
!ELEMENT properties
  ( description?,
    ( simple
      | simplesequence
      | test
      | struct
      | structsequence
    )+
  )>
```
<!ELEMENT simple
  ( description?,
   value,
   units?,
   range?,
   enumerations?,
   kind*,
   action?,
 )>

<!ATTLIST simple
  id ID #REQUIRED - the ID value is used within the CF DataType
type( boolean | char | double | float
   | short | long | objref | octet
   | string | ulong | ushort) #REQUIRED
name CDATA #IMPLIED - used when ID is not meaningful, HCI
usage
mode (readonly | readwrite | writeonly) "readwrite">
Properties - simple Element

- The simple element provides definition for the unique id, name, type, and value of a property attribute to be used in the SCA CF::Resource class configure() and query() interfaces or for indication of component capabilities.

- Specifically designed to support id-value pair definitions
  - *type* attribute defines the data type of the property
  - *id* attribute for an allocation type is a DCE UUID value, otherwise, can be any valid XML ID type
  - *mode* attribute defines allowed direction flow for this property, only meaningful when the kind sub-element is ‘configure’
Simple - description Element

- The description element presents any pertinent information about the properties element being defined.
- description element is optional

<!ELEMENT description (#PCDATA)>
Simple - value Element

- The value element provides a value setting for the properties element being defined.

- If no value or default value is given, the property cannot be used for as an initial configuration or execute parameter of a component

```xml
<!ELEMENT value (#PCDATA)>``
Simple - units Element

- The units element defines the intended practical data representation to be used for the properties element.

- units element is optional

<!ELEMENT units (#PCDATA)>
Simple - range Element

- The range element defines the maximum and minimum values that are legal for the simple element.
- intended to provide a means to perform range checking of values
- range element is optional

<!ELEMENT range EMPTY>
<!ATTLIST range
    min CDATA #REQUIRED
    max CDATA #REQUIRED>
Simple - enumerations Element

- The enumerations element associates text labels with specific values for a property.
  - simple element type must be an integer type
  - If no enumeration values are specified, enumeration values will start at 0 and increment by 1 implicitly

- enumerations element is optional

```xml
<!ELEMENT enumerations
 ( enumeration+ )>
<!ELEMENT enumeration EMPTY>
<!ATTLIST enumeration
 label CDATA #REQUIRED
 value CDATA #IMPLIED>
```
Simple - kind Element

- The kind element specifies the kind of property being defined.
  - 5 kind choices are valid: configure, test, allocation, execparam, and factoryparam
  - simple element may have multiple kinds defined
  - default is ‘configure’
- each simple element must have at least one kind sub-element

```xml
<!ELEMENT kind EMPTY>
<!ATTLIST kind
  kindtype (allocation | configure | test | execparam | factoryparam) "configure">
```
Simple - kind Element cont’d

- Valid Property Kinds
  - `configure` = property used by `configure()` and `query()` operations from the CF::Resource interface
    - `mode = readonly` means only `query()` supported
    - `mode = writeonly` means only `configure()` supported
    - `mode = readwrite` means both supported
  - `test` = property used by `runtest()` operation from the CF::TestableObject interface
    - simple element must have type of ‘ulong’
  - `allocation` = property used by `allocateCapacity()` and `deallocateCapacity()` operation from the CF::Device interface
    - `allocateCapacity()` and `deallocateCapacity()` invoked during application creation
Simple - kind Element
cont’d

• execparam = property used by `execute()` and operation from the CF::ExecutableDevice interface
  – only simple elements can be used as execparams

• factoryparam = property used by `createResource()` operation from the CF::ResourceFactory interface
Simple - action Element

- The action element specifies the type of comparison to be done when checking a dependency against the property being defined.
  - 5 action choices are valid: eq, ne, gt, lt, ge, le, and external
  - during a comparison operation, the property is on the left side of the equality operator
  - external means the property is an allocation property and the comparison operation should be ‘??’
  - default is ‘external’
- action element is optional

```xml
<!ELEMENT action EMPTY>
<!ATTLIST action
type (eq | ne | gt | lt |
ge | le | external) "external"/>
```
Properties - simplesequence Element

- The simplesequence element is used to specify a list of properties with the same characteristics (i.e., type, range, units, etc.).
  - Same as simple element, except it has a list of values and default values instead of one value
  - Provides for support of sequence data types for properties
  - simplesequence cannot be allocation or executable kinds.
<!ELEMENT simplesequence
  ( description?,
   values - 1 or more value elements,
   units?,
   range?,
   kind*,
   action?)>

<!ATTLIST simplesequence
  id ID #REQUIRED
  type( boolean | char | double | float
       | short | long | objref | octet
       | short | string | ulong | ushort | uuid) #REQUIRED
  name CDATA #IMPLIED - used when ID is not meaningful, HCI
  usage
  mode (readonly | readwrite | writeonly) "readwrite">
PRF Properties Element Format Recommendations

- Order of `simple` and `simplesequence` declarations within a `properties` element is important
  - properties with specified values (or default values) and mode of ‘readwrite’ or ‘writeonly’
  - properties without a specified value (or default value)
  - properties with a mode of ‘readonly’
Properties Conversion

- The simple/simplesequence properties are converted to CF Properties type (a list of properties (CF DataTypes)) as follows:
  - CF DataType’s id value maps to simple/simplesequence id attribute value
  - CF DataType’s value
    - when property is a simple and kind is execparam the CF DataType’s value is a CORBA string type and has the string value of simple value/default value.
    - when property is simple and kind is not execparam the CF DataType’s value is a CORBA basic type that corresponds to the simple type attribute and has the value of the simple value/default value string converted to the CORBA basic type.
Properties Conversion cont’d

- The simple/simplesequence properties are converted to CF Properties type (a list of properties (CF DataTypes)) as follows:
  - CF DataType’s value
    - when property is simplesequence, the CF DataType’s value is a CORBA sequence type that corresponds to the sequence types defined in the CF and PortTypes CORBA modules based upon the type element. The simplesequence values are converted to the appropriate sequence type.
Properties – test element

- The test element is used to specify a list of test properties for executing the runTest() operation to perform a component specific test. Input values are used to configure the test to be performed (e.g., frequency and RF power output level). When the test has completed, result values contain the results of the testing (e.g., Pass or a fault code/message).

```xml
<!ELEMENT test
  ( description
   , inputvalue?
   , resultvalue
  )>
<!ATTLIST test
  id ID #REQUIRED>
```
Properties – inputvalue and resultvalue elements

• The inputvalue element is used to provide test configuration properties.
• The resultvalue element is used to provide test result properties.

<!ELEMENT inputvalue
    ( simple+ )>

<!ELEMENT resultvalue
    ( simple+ )>
Properties – struct element

- The `struct` element is used to group properties with different characteristics (i.e., similar to a structure or record entry). Each item in the struct element can be a different simple type (e.g., short, long, etc.).

```xml
<!ELEMENT struct
  ( description?,
    simple+,
    configurationkind? )>
<!ATTLIST struct
  id ID #REQUIRED
  name CDATA #IMPLIED
  mode (readonly | readwrite | writeonly) "readwrite"/>
```

```xml
<!ELEMENT configurationkind EMPTY>
<!ATTLIST configurationkind
  kindtype (configure | factoryparam) "configure"/>```

Properties – structsequence element

- The *structsequence* element is used to specify a list of properties with the same struct characteristics.

```xml
<!ELEMENT structsequence (
  description?,
  structvalue+,
  configurationkind?)>
<!ATTLIST structsequence
  id ID #REQUIRED
  structrefid CDATA #REQUIRED
  name CDATA #IMPLIED
  mode (readonly | readwrite | writeonly) "readwrite">

<!ELEMENT structvalue (simpleref+)>```

```xml
<!ELEMENT simpleref EMPTY>
<!ATTLIST simpleref
  refid CDATA #REQUIRED
  value CDATA #REQUIRED>```
Software Component Descriptor
Software Component Descriptor (SCD) Intro

- Defines elements necessary for describing the dynamic (i.e. CORBA) input and output interfaces for the software component

- Components must ultimately be children of one of the Core Framework interface classes

- Based upon the CORBA Component Descriptor specification
Software Component Descriptor (SCD)

• Provides
  – A “class-like” definition of a CORBA component.
  – A mechanism for classification of components based on supported interfaces.
  – Software component’s CORBA version, interface repository id, and CF parent class type
  – Component Ports
    • Uses Port - interfaces the component uses from other components
    • Implements the CF _Port_ interface
    • Provides Port - interfaces (services) provided by a component
    • Ports object references are retrieved by using the CF _Resource_ getPort operation.
Software Component Descriptor (SCD)

```
<<DTDElement>>
softwarecomponent

<<DTDSequenceGroup>>
softwarecomponent_grp
(from softwarecomponent)

{1}
<<DTDElementPCDATA>>
corbaversion

{2}
<<DTDElementEMPTY>>
componentrepid
repid : CDATA

{3}
<<DTDElementPCDATA>>
componentype

{4}
<<DTDElement>>
componentfeatures

{5}
<<DTDElement>>
interfaces

{6}
<<DTDElement>>
propertyfile
type : CDATA
```
<!ELEMENT softwarecomponent
  ( corbaversion - version of CORBA the component is assuming
   | componentrepid - interface repository id of the component
   | componenttype - type of component (Resource, Device, etc.)
   | componentfeatures - component’s ports
   | interfaces - a list of interface definitions
   | propertyfile? - Properties file reference
  )>
Softwarecomponent - corbaversion Element

- The corbaversion element indicates the specific version of CORBA that the delivered software component supports.
- version is represented by a major and minor number separated by a “.” (i.e. 2.2, 3.0)

<!ELEMENT corbaversion (#PCDATA)>
Softwarecomponent - componentrepid Element

- The componentrepid element uniquely identifies the interface that the component is implementing.
- The id is derived from the CF::Resource, CF::Device, or CF::ResourceFactory interface classes
- componentrepid element may be referred to by the componentfeatures element

<!ELEMENT componentrepid EMPTY>
<!ATTLIST componentrepid
  repid CDATA #REQUIRED>  - fully qualified repository id of the component.
Softwarecomponent - componenttype Element

- The *componenttype* element identifies the type of the software component object.

```xml
<!ELEMENT componenttype (#PCDATA)>
```

- For SCA components, allowed types are:
  - resource
  - device
  - resourcefactory
  - domainmanager
  - log
  - filesystem
  - filemanager
  - devicemanager
  - namingservice
  - eventservice
Softwarecomponent - componentfeatures Element

- The componentfeatures element describes the component’s inherited interfaces, as well as its provides and uses ports.
- As a minimum, the component’s interface has to be a CF::Resource, CF::Device, or CF::ResourceFactory.
- All inherited interfaces must be depicted as supportsinterface elements.

```xml
<!ELEMENT componentfeatures - describes component inheritance, supported
      ( supportsinterface* interfaces, uses and provides
        ports of
        , ports the component.
      )>
```
The supportsinterface element identifies an IDL interface that the component supports (i.e. provides to the outside world).

Are interfaces which were inherited by the component’s specific interface (i.e. component’s interface can be widened to be a supportsinterface)

Repid refers to repository id in interfaces element

supportsname used by CF::Resource::getport() and CF::Port::connectport(),disconnectport() operations

```xml
<!ELEMENT supportsinterface EMPTY>
<!ATTLIST supportsinterface

  repid CDATA #REQUIRED

  supportsname CDATA #REQUIRED>
```
Component Features - ports Element

- The *ports* element describes the interfaces a component provides and uses.
  - Provides ports do not need to be a CF::*Port* type
  - Uses ports *must* be a CF::*Port* type

```xml
<!ELEMENT ports - describes what interfaces a component provides and uses
  (provides
  | uses
  )*>  

<!ELEMENT provides - specifies an interface that is provided by the component
  ( porttype*)> 
<!ATTLIST provides
  repid CDATA #REQUIRED
  providesname CDATA #REQUIRED>
```
Component features - ports

Element cont’d

- Any number of uses and provides elements can be given in any order
- \textit{providesname} and \textit{usesname} used by CF::Resource::getport() and CF::Port::connectport(), disconnectport() operations
- \textit{providesnames} and \textit{usesnames} used in the Software Assembly Descriptor (SAD) when connecting ports together
- \textit{Repid} refers to repository id in \textit{interfaces} element
- If optional \textit{porttype} is not given, ‘control’ is the default value

```xml
<!ELEMENT uses> — specifies an interface that is used
  ( porttype*)> — by the component
<!ATTLIST uses
  repid CDATA #REQUIRED
  usesname CDATA #REQUIRED>

<!ELEMENT porttype EMPTY> — type of port
<!ATTLIST porttype
  type ( data | control | responses | test ) #REQUIRED>
```
**Softwarecomponent - interfaces Element**

- The *interfaces* element describes one or more interfaces that the component, either directly or through inheritance, provides, uses, or supports.

```xml
<!ELEMENT interfaces - a list of interface definitions
 ( interface+ )>

<!ELEMENT interface - interface definition
 ( inheritsinterface*)>
<!ATTLIST interface
 repid CDATA #REQUIRED - fully qualified repository id
 name CDATA #REQUIRED> - non-qualified name of interface

<!ELEMENT inheritsinterface EMPTY>
<!ATTLIST inheritsinterface
 repid CDATA #REQUIRED> - repository id of inherited interface
```
Software Assembly Descriptor
Software Assembly Descriptor (SAD)

- Provides a “class-like” definition for an application abstraction.
- Contains a list of components to be created and how these components are to be connected together.
- Based upon the CORBA Components Specification Component Assembly Descriptor.
- References other Software Profile files.
Software Assembly Descriptor (SAD)

• Purpose
  – Defines components to be executed
  – Defines interconnections between component objects comprising the application.
  – Defines the assembly controller component for an application.
  – Defines the externally visible ports for an application.
<!ELEMENT softwareassembly
    ( description?,
    , componentfiles  - referenced SPD files for components creation
    , partitioning    - components to be created
    , assemblycontroller? - main component for controlling the assembly
    , connections? - components connections
    , externalPorts? - assembly’s visible ports, besides CF
    )>

<!ATTLIST softwareassembly
    id ID #REQUIRED - unique ID (UUID) of SAD within domain
    name CDATA #IMPLIED> - user friendly name
SAD - softwareassembly Element

- Componentfiles, partitioning, and assemblycontroller elements are required.
- When a partitioning element is specified, the related componentfiles element must also be specified.
- Id is a DCE UUID value which uniquely identifies the assembly.
- Name is a user-friendly name for the software assembly. It will be used as a label for the CF::ApplicationFactory interface object’s name attribute.
Softwareassembly - description Element

- The *description* element presents any pertinent information about the software components being installed by the SAD.

`<!ELEMENT description (#PCDATA)>`
Softwareassembly - componentfiles Element

- The *componentfiles* element indicates that an assembly is made up of 1..n component files.

```xml
<!ELEMENT componentfiles - a list of component files
   ( componentfile+ )>

<!ELEMENT componentfile
   ( localfile )>

<!ATTLIST componentfile
   id ID #REQUIRED - unique component file id within SAD
   type CDATA #IMPLIED - Software Package Descriptor type
```
Software assembly - componentfiles Element cont’d

- componentfiles element is one or more componentfile elements
- componentfile element specifies a SPD file
- componentfile.id attribute is a unique identification string for use within the SAD
- componentfile.type attribute is always “Software Package Descriptor”
- componentfile.localfile element provides the name of a file containing a SPD

<!ELEMENT localfile EMPTY> - file located in the same directory as the SAD file
<!ATTLIST localfile
    name CDATA #REQUIRED>
Softwareassembly - partitioning Element

- The partitioning element specifies a deployment pattern of component incarnations (i.e. application code) to hosts.
- partitioning element consists of a set of componentplacement elements or hostcollocation lists
- a componentinstantiation element is captured inside a componentplacement element
- hostcollocation element specifies 1..n componentplacements which are required to be located on a common device

```xml
<!ELEMENT partitioning ( componentplacement | hostcollocation )*>
```

- list of software components to deploy
- individual component to be deployed
- components to be deployed on same host
Partitioning - componentplacement Element

- The componentplacement element defines a particular deployment of a component.
- componentfileref element identifies the component to be deployed
- componentinstantiation element identifies the actual component created using a DCE UUID
- multiple component instantiations of the same kind can be created within the same componentplacement element

<!ELEMENT componentplacement
  ( componentfileref - component file to be used for deployment,
    componentinstantiation+ - component instantiation characteristics
  )>
Partitioning - hostcollocation Element

- The hostcollocation element defines a set of component instances to be deployed on a common device or host.
- Optional id element uniquely identifies the set of component instances to be created
- Optional name element provides a user-friendly label for the set of component instances

<!ELEMENT hostcollocation - components that are collocated on the same host>

<!ATTLIST hostcollocation
  id   ID     #IMPLIED
  name CDATA #IMPLIED>
Component placement -
componentfileref Element

• The componentfileref element references a component file within the componentfiles element of the SAD.

• refid attribute corresponds to one of the componentfile ids within the SAD’s componentfiles element

```xml
<!ELEMENT componentfileref EMPTY>
<!ATTLIST componentfileref
  refid CDATA #REQUIRED> - componentfile id attribute within SAD
```
Componentplacement - componentinstantiation Element

- The componentinstantiation element describes a particular incarnation of a component relative to a componentplacement element.

```xml
<!ELEMENT componentinstantiation
  ( usagename?, componentproperties?, findcomponent? )>
<!ATTLIST componentinstantiation
  id ID #REQUIRED>
```

```xml
<!ELEMENT usagename (#PCDATA)>
```
Componentinstantiation Element cont’d

• optional componentproperties element provides a list of property values used in configuring the component incarnation

```xml
<!ELEMENT componentproperties
 ( simpleref
 | simplesequenceref
 | structref
 | structsequenceref
 )+ >
```
Component instantiation Element cont’d

• Precedence for component instantiation properties is:
  – `componentproperties` element from SAD
  – DPD properties
  – SCD properties
  – SPD implementation properties
  – SPD properties
Component instantiation - findcomponent Element

- The findcomponent element provides the method to be used to obtain the CORBA object reference for the component instance.
- Either by a componentresourcefactory or the namingservice

```xml
<!ELEMENT findcomponent - how to obtain the component obj ref
    ( componentresourcefactoryref
    | namingservice
    )>
```
componentresourcefactoryref Element

- componentresourcefactoryref element provides the deployed components object reference from a component factory.
- optional resourcefactoryproperties element provides qualifier properties which will be used for the CF::ResourceFactory::create() call when creating the object instance

```
<!ELEMENT componentresourcefactoryref
  ( resourcefactoryproperties?)>

<!ELEMENT resourcefactoryproperties
  ( simpleref
   | simplesequencerefer
   | structref
   | structsequencerefer
  )+ >
```
Findcomponent – namingservice Element

- The namingservice element is used to indicate to the CF ApplicationFactory the requirement to find a component interface.
- The CF ApplicationFactory will use the name attribute to search the CORBA Naming Service for the appropriate Interoperable Object Reference (IOR).

```xml
<!ELEMENT namingservice EMPTY
<!ATTLIST namingservice
  name CDATA #REQUIRED
```
Softwareassembly - assemblycontroller Element

- The assemblycontroller element indicates the component that is the main controller for this entire assembly.
- Only one assemblycontroller allowed per assembly
- componentinstantiationref element corresponds to one of the componentinstantiation ids within componentplacement elements
- CF Application object delegates its CF::Resource start, stop, runTest, configure, and query operations to this component

<!ELEMENT assemblycontroller
    ( componentinstantiationref - identifies the main controller component within the SAD. )>
Softwareassembly -
connections Element

- The connections element provides the mapping of communications path connections between components in the assembly.
- Only one connections element allowed per assembly
- Multiple communications paths may be specified within the connections element
- Information regarding the two components involved in a communications path is provided within the connectinterface element

<!ELEMENT connections - a list of components
connections
( connectinterface*)>
Connections - connectinterface Element

- The connectinterface element describes the endpoints for a single communications path between two components within the assembly.
- Optional id attribute is used by the CF ApplicationFactory as the connection ID on the CF::Port::connectPort operation, otherwise the ApplicationFactory creates a connection ID.

```xml
<!ELEMENT connectinterface - one connection
   ( usesport - the uses port used in the connection
     , ( providesport - the provides port or supported componentsupportedinterface interface used in the connection
       | findby
     )
   )>
<!ATTLIST connectinterface - optional ID for the connection
   id ID #IMPLIED>
```
Connectinterface - usesport Element

- The usesport element identifies a component’s port which is utilizing the interface provided by another component within the assembly.

```
<!ELEMENT usesport
  ( usesidentifier - uses port name as defined in SCD,
    (componentinstantiationref - several methods to find the component
     | devicethatloadedthiscomponentref
     | deviceusedbythiscomponentref
     | findby
    ))>
```
Connectinterface - usesport
Element cont’d

• usesidentifier element refers to a component’s port identifier as provided in the Software Component Descriptor (SCD)

• usesidentifier will be used in call to the CF::Resource::getPort() operation

<!ELEMENT usesidentifier (#PCDATA)>

<!ELEMENT componentinstantiationref EMPTY>
<!ATTLIST componentinstantiationref
  refid     CDATA  #REQUIRED> - identifies the componentinstantiation element (id) within the SAD
devicethatloadedthiscomponentref Element

- The devicethatloadedthiscomponentref element establishes an object reference to a Device that loaded a component.
- refid attribute refers to a componentinstantiation element from this SAD
- this method for obtaining an object reference does not actually instantiate a new object, rather it obtains the object reference of an active Device to be used later within the connections element of the SAD
componentusedbythiscomponentref Element

- The componentusedbythiscomponentref element establishes an object reference to a Device in the system which is being used by a component.
- refid attribute refers to a componentinstantiation element from this SAD
- this method for obtaining the object reference of an active Device to be used earlier during the deployment of a component.

<!ELEMENT componentusedbythiscomponentref EMPTY>

<!ATTLIST componentusedbythiscomponentref
  refid CDATA #REQUIRED> - identifies the component within the SAD
  usesrefid CDATA #REQUIRED> - identifies the uses component
Findcomponent - findby Element

• The findby element indicates the method to be used to locate a particular component interface.

```xml
<!ELEMENT findby
  ( namingservice
  | domainfinder
 )>

<!ELEMENT namingservice EMPTY>
<!ATTLIST namingservice
  name CDATA #REQUIRED>

<!ELEMENT domainfinder EMPTY>
<!ATTLIST domainfinder
  type (fileManager|log|eventchannel|namingservice) #REQUIRED
  name CDATA #IMPLIED>
```
Findcomponent - findby Element cont’d

- Only one selection allowed per findby element
- namingservice element indicates CORBA Naming Service use
- namingservice.name attribute provides text string used to search the Naming Service for the desired interface
- stringifiedobjectref element provides a stringified object reference (IOR) for the desired component interface
- domainfinder element indicates that the desired component interface belongs to a service that has already registered with and may be obtained from the DomainManager
Findcomponent - findby Element cont’d

- domainfinder.type attribute indicates the type of the service
  - filemanager
  - logger
  - namingservice
- domainfinder.name attribute indicates instance name of the service
- if domainfinder.name is unspecified, then the object reference returned is the CF::DomainManager’s filemanager, logger, or naming service corresponding to the type requested
Connectinterface - providesport Element

- The providesport element identifies a component’s port which makes available the interface to be used by another component within the assembly.

```xml
<!ELEMENT providesport
  ( providesidentifier - provides port name as defined in SCD
  , (componentinstantiationref - two methods to find the component
  | devicethatloadedhiscomponentref - device object ref
  | deviceusedbythiscomponentref -
  | findby
  )
  )>

<!ELEMENT providesidentifier (#PCDATA)>  

<!ELEMENT componentinstantiationref EMPTY>  
<!ATTLIST componentinstantiationref refid CDATA #REQUIRED> - identifies the componentinstantiation element (id) within the SAD
Connectinterface - providesport Element cont’d

• providesidentifier element refers to a component’s port identifier as provided in the Software Component Descriptor (SCD)

• providesidentifier will be used in call to the CF::Resource::getPort() operation

• two methods are available for locating the component instance: componentinstantiationref and findby

• componentinstantiationref element corresponds to one of the componentinstantiation ids within the componentplacement elements of the SAD

• findby element is the same format as in the SAD’s componentinstantiation element
Connectinterface - componentsupportedinterface

- The componentsupportedinterface element identifies a component with a supports interface that can satisfy a connection to a uses port.

```xml
<!ELEMENT componentsupportedinterface
   ( supportedidentifier - supported interface name as defined in SCD,
                        (componentinstantiationref - two methods to find the component
                         | findby
                         )
   )>

<!ELEMENT supportedidentifier (#PCDATA)>

<!ELEMENT componentinstantiationref EMPTY>
<!ATTLIST componentinstantiationref
   refid CDATA #REQUIRED>
```

- identifies the componentinstantiation element (id) within the SAD
componentsupportedinterface cont’d

- supportedidentifier element refers to a component’s supported interface identifier as provided in the Software Component Descriptor (SCD)
- supportedidentifier will be used in call to the CF::Resource::getPort() operation
- two methods are available for locating the component instance: componentinstantiationref and findby
- componentinstantiationref element corresponds to one of the componentinstantiation ids within the componentplacement elements of the SAD
- findby element is the same format as in the SAD’s componentinstantiation element
Softwareassembly -
externalports Element

• The *externalports* element describes the interfaces which are made externally available by the component.

```xml
<!ELEMENT externalports (port+)>

<!ELEMENT port (description?, (usesidentifier | providesidentifier | supportedidentifier), componentinstantiationref)>

<!ELEMENT description (#PCDATA)>

```

- a list of external ports for the assembly
- defines an external port
- identifies a component’s port that is visible for the assembly
Softwareassembly -
externalports Element cont’d

- Multiple port elements can be described in a single externalports element
- usesidentifier, providesidentifier, and supportedidentifier elements identify the type and identity of the external port
- The identifiers above refer to the appropriate component’s port identifier as provided in the Software Component Descriptor (SCD)
- componentinstantiationref element is the same as in the usesport, providesport, and component supportedinterface elements, respectively
- CF::Application::getport() operation is used to access the assembly’s visible ports
Device Configuration Descriptor
Device Configuration Descriptor (DCD) Intro

- Describes the DeviceManager startup behavior
- Based upon the Software Assembly Descriptor (SAD) DTD
- Specifies Devices to be deployed
- Specifies Services to be obtained or created
Device Configuration Descriptor (DCD)

Purpose

- DeviceManager startup behavior
  - The logical Devices to be deployed
  - The composition of logical Device(s)
  - How to obtain the DomainManager object reference
  - The Services to be obtained or created
    - Logger
  - Specify the mount point names for File Systems
Device Configuration Descriptor (DCD)
<!ELEMENT deviceconfiguration
  ( description? - optional description of configuration
   , devicemanagersoftpkg - deviceManagers SPD
   , componentfiles? - referenced SPD files for components' creation
   , partitioning? - the components to be created
   , connections? - connections for the components created
   , domainmanager - how to obtain DomainManager object ref
   , filesystemnames? - mount names for file systems
  )>

<!ATTLIST deviceconfiguration
  id ID #REQUIRED - values for the identifier and label
  name CDATA #IMPLIED - attributes of the DeviceManager
DCD - deviceconfiguration Element

- *Componentfiles* and *partitioning* elements are optional; if not provided, no device components are started up on the node except for a *DeviceManager*.
- If a *partitioning* element is specified, the related *componentfiles* element must also be specified.
- *Id* can be a UUID value or a manufacturer’s part number/serial number string.
- *Name* is a user-friendly name for the parent device label.
Deviceconfiguration - description Element

- The *description* element presents any pertinent information about the device configuration being installed by the DCD.

```xml
<!ELEMENT description (#PCDATA)>
```
Deviceconfiguration - devicemanagersoftpkg Element

• The devicemanagersoftpkg element refers to the SPD for a CF DeviceManager that corresponds to this DCD. The SPD file is referenced by a localfile element. The referencedfile can be used to describe the DeviceManager implementation.

```xml
<!ELEMENT devicemanagersoftpkg
  ( localfile
  )>
```
Deviceconfiguration - componentfiles Element

- The `componentfiles` element indicates that a node uses 1..n component files to define the components that are started up on the node.

```xml
<!ELEMENT componentfiles ( componentfile+)>
<!ELEMENT componentfile ( localfile )>
<!ATTLIST componentfile
    id ID #REQUIRED
    type CDATA #IMPLIED>
```

- a list of component files
- unique component file id within DCD
- Software Package Descriptor type
Device configuration - partitioning Element

- The partitioning element specifies a deployment pattern of component incarnations (i.e. Devices or services).
- partitioning element consists of a set of component placement elements

```xml
<!ELEMENT partitioning (
    componentplacement
)*>
```
Partitioning - componentplacement Element

- The componentplacement element defines a particular deployment of a component.
- componentfileref element provides component deployment SPD

```xml
<!ELEMENT componentplacement
 ( componentfileref - component file (SPD) to be used for deployment
 , deployedondevice? - refers to a device to deploy this device
 , compositepartofdevice? - this device is a composite a device
 , devicepkgfile? - component hw file (DPD)
 , componentinstantiation* - component instantiation characteristics
 )>
```
Component placement - componentfileref Element

- The componentfileref element references a component file within the componentfiles element of the DCD.
- refid attribute corresponds to one of the componentfile ids within the DCD’s componentfiles element

```xml
<!ELEMENT componentfileref EMPTY>
<!ATTLIST componentfileref
  refid CDATA #REQUIRED>-- componentfile id attribute within DCD
```
Component placement - deployondevice Element

- The deployondevice element is used to reference a componentinstantiation element on which this componentinstantiation is deployed.

```xml
<!ELEMENT deployondevice EMPTY>
<!ATTLIST deployondevice
    refid CDATA #REQUIRED>
```
Component placement - compositepartofdevice Element

- The compositepartofdevice element is used when an aggregate relationship exists for this component to be deployed and another component instantiation in the DCD.

- Upon execution of this Device the CF DeviceManager passes the component the object reference of the parent Device.

```xml
<!ELEMENT compositepartofdevice EMPTY>
<!ATTLIST compositepartofdevice
  refid CDATA #REQUIRED>
```
Componentplacement - componentinstantiation Element

- The componentinstantiation element describes a particular incarnation of a component relative to a componentplacement element.

```xml
<!ELEMENT componentinstantiation ( usagename?, componentproperties?)>
<!ATTLIST componentinstantiation id ID #REQUIRED>
<!ELEMENT usagename (#PCDATA)>
```

- Device’s label attribute value
- instantiation’s specific properties
- Device’s identifier attribute value
Component instantiation
Element cont’d

• optional componentproperties element provides a list of property values used in configuring the component incarnation

• Precedence for component instantiation properties is:
  – componentproperties element from DCD
  – DPD properties
  – SCD properties
  – SPD implementation properties
  – SPD properties

<!ELEMENT componentproperties
  ( simpleref
   | simplesequenceref
   | structref
   | structsequenceref
  )+ >
Deviceconfiguration -
domainmanager Element

• The domainmanager element indicates how the DeviceManager is to obtain the CF::DomainManager reference.

• The DomainManager registers with the NamingService using the following format:
  /domainName/DomainManager

<!ELEMENT domainmanager
  ( namingservice)>

<!ELEMENT namingservice EMPTY>
<!ATTLIST namingservice
  name CDATA #REQUIRED>
Deviceconfiguration - filesystemnames Element

- The filesystemnames element indicates the mount point name to be used for each valid file system.
- Multiple file system mount points may be declared in a filesystemnames element.
- filesystemname.mountname is a simple name string

```xml
<!ELEMENT filesystemnames - a list of file system mount names
   (filesystemname+)
>
<!ELEMENT filesystemname EMPTY>
<!ATTLIST filesystemname
   mountname CDATA #REQUIRED - mount name for file system device
deviceid  CDATA #REQUIRED> - identifies a device identifier
```
DomainManager Configuration Descriptor
DomainManager Configuration Descriptor (DMD)

- Describes the DomainManager startup behavior
  - Description
  - DomainManager’s SPD
  - Specifies Services to be used by the DomainManager implementation
DomainManager Configuration Descriptor (DCD)

<<DTDElement>>
domainmanagerconfiguration

- id : ID
- name : CDATA

<<DTDSquenceGroup>>
domainmanagerconfiguration_grp
(from domainmanagerconfiguration)

{1} 0..1
<<DTDElementPCDATA>>
description

{2}
<<DTDElement>>
devicemanagersoftpkg

{3}
<<DTDElement>>
services
DMD DTD

<!ELEMENT domainmanagerconfiguration
  ( description? - optional description
  , domainmanagersoftpkg - SPD
  , services - Services to use )>

<!ATTLIST domainmanagerconfiguration
  id ID #required - UUID
  name #CDATA #required> - name
DomainManager configuration - 
*description* Element

- The *description* element presents any pertinent information about the DomainManager configuration of the DMD.

<!ELEMENT description (#PCDATA)>
The domainmanagersofpkg element refers to the SPD for a CF DomainManager that corresponds to this DMD. The SPD file is referenced by a localfile element. The referenced file can be used to describe the DomainManager implementation.

<!ELEMENT domainmanagersofpkg
  ( localfile )>
DomainManagerconfiguration - services Element

- The services element in the DMD is used by the CF DomainManager to determine which service (Log, etc.) instances to use.

```
<!ELEMENT services
 ( service+)
 >

<!ELEMENT service
 ( usesidentifier,
   findby
 )>

<!ELEMENT usesidentifier (#PCDATA)> - name of Log service

<!ELEMENT findby
 ( namingservice
 | domainfinder
 )>
```
Profile Descriptor
Profile Descriptor Intro

- Not a separate file like the other descriptors
- Indicates the type of XML file being referenced
- Provides information to access the profile (the filename and path)
- Given out as a value for the Application, ApplicationFactory, DeviceManager, DomainManager and Device profile attributes.
<!ELEMENT profile EMPTY> - full file path name relative to CF DomainManager or DeviceManager FileSystems
<!ATTLIST profile
  filename CDATA #REQUIRED
  type CDATA #IMPLIED
- SAD, SPD, DCD
Profile Element

- The profile element specifies the absolute profile file pathname relative to a mounted FileSystem.
- Filename attribute’s path is relative to a CF::DomainManager’s or CF::DeviceManager’s FileSystems.
- Filename’s specified path can also be used to access other local file elements in the profile.
- Valid type attribute values are:
  - SAD
  - SPD
  - DCD
Summary

- XML used for JTRS profiles
  - Used to describe HW and SW in a std. format
  - Used to deploy Applications and logical Devices in JTRS compliant system

- SCA defines 8 file types for the Domain Profile
  - Software Component Descriptor (SCD)
  - Software Package Descriptor (SPD)
  - Software Assembly Descriptor (SAD)
  - Property File (PRF)
  - Device Package Descriptor (DPD)
  - Device Configuration Descriptor (DCD)
  - DomainManager Configuration Descriptor (DMD)
  - Profile Descriptor
Open Discussion