

## Problem

What are hierarchical components, and how can I add hierarchical components to the ACS CDB?

## Solution

The deployment of Components is very important to understand and maintain complex systems.

In a small system with few Components each Component can be given a simple name and they can just be deployed using this name and writing entries in the `MACI/Components/Components.xml` file.

But with growing complexity of the system, this file becomes very large and it is difficult to see the relations between components.

Most systems have a naturally hierarchical logical structure, i.e. it is possible to group Components according to functional groups and sub-groups and to logical and/or physical (for hardware devices) containment rules.

The ACS CDB allows deployment of Components according to these rules.

See also Chapter 14 of the Configuration Database user manual.

### TOC

- [Hierarchical Components and CDB Structure](#)
  - [Example CDB](#)
  - [Logical architecture of the example](#)
  - [Deployment configuration options](#)
    - [Components.xml](#)
    - [Hierarchical components as directory tree](#)
    - [Hierarchical components as single file](#)
    - [Deep hierarchy with pure-logical nodes](#)
    - [Hierarchy with pure-logical nodes and sub-nodes in one file](#)
    - [Putting multiple XML files in the same directory using XInclude](#)
      - [Example](#)
    - [Limitations and glitches](#)
  - [Characteristic Components and CDB for instances configuration](#)
  - [How to try out the example CDB](#)



**Updated with XInclude directives** -- [GianlucaChiozzi](#) - 06 Sep 2005

## Example CDB

We will describe how this can be done using an example CDB, attached here:

- [CDB.tar.gz](#): Example of Hierarchical CDB

As of ACS 5.0, the module `acsexmp1` contains also hierachical structures based on this same example. -- [GianlucaChiozzi](#) - 06 Sep 2005

## Logical architecture of the example

In this example CDB the componenents have been deployed in the following logical hierarchical structure:

- **ALMA\_DOOR**
- **ALMA\_BACK\_DOOR**
- **ALMA\_BACK\_DOOR/LAMP**
- CONTROL
  - ANTENNAS
    - ANTENNA\_1
      - **MOUNT**
        - **LAMP\_1**
      - **POWER\_SUPPLY**
    - ANTENNA\_2
      - **MOUNT**
      - **POWER\_SUPPLY**
    - ANTENNA\_3
      - **MOUNT**
      - **POWER\_SUPPLY**
    - ANTENNA\_H
      - **MOUNT**
      - **POWER\_SUPPLY**
- **TOWER\_1**
  - **FRONTDOOR**
- **TOWER\_H**
  - **FRONTDOOR**
- TestInclude\_01
- TestInclude\_02
- TestInclude\_03
- ... dynamic components defined using XInclude ...

This set of components allows us to show various deployment alternatives and to compare them.

Here the names in **BOLD** correspond to actual components, while the other names are simply logical grouping names and do not correspond to actual components.

From the logical point of view, the complete name of each component is built taking the hierarchical path using the '/' path separator.

For example:

- CONTROL/ANTENNAS/ANTENNA\_1  
Is a logical entity to group all Components that have to do with ANTENNA\_1.
- CONTROL/ANTENNAS/ANTENNA\_1/MOUNT  
Is the actual Component for the Mount of ANTENNA\_3
- CONTROL/ANTENNAS/ANTENNA\_1/MOUNT/LAMP\_1  
Is another Component logically in ANTENNA\_1  
More over this LAMP Component is also part of MOUNT\_1  
Remember that this is just a logical containment and not a physical containment. CONTROL/ANTENNAS/ANTENNA\_1/MOUNT and CONTROL/ANTENNAS/ANTENNA\_1/MOUNT/LAMP\_1 might be physically deployed in different Containers and in different hosts. for example because the LAMP\_1 is controlled by an analog I/O board in another computer.  
Usually this logical hierarchy means that LAMP\_1 is essential for the functioning of the parent node MOUNT and that is activated by it. But this is not a requirement.

## Deployment configuration options

The configuration of the deployment of the system is done in the branch

- **MACI/Components**

of the configuration database.

## Components.xml

The simplest option consists in listing all Components in the **Components.xml** file according to the schema `urn:schemas-cosylab-com:Components:1.0`.

Hierarchical names can be put here by putting the complete name with '/' separators in the `Name` attribute of the `Component`.

For our example we have simply put here two `Components` that are at the root level of the hierarchy:

```
<?xml version="1.0" encoding="utf-8"?>
<Components xmlns="urn:schemas-cosylab-com:Components:1.0"
  xmlns:cdb="urn:schemas-cosylab-com:CDB:1.0"
  xmlns:baci="urn:schemas-cosylab-com:BACI:1.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xi="http://www.w3.org/2003/XInclude">
  <_ Name="ALMA_DOOR"
    Code="acsexmplDoorImpl"
    Type="IDL:alma/acsexmplBuilding/Door:1.0"
    Container="Container" />
  <_ Name="ALMA_BACK_DOOR"
    Code="acsexmplDoorImpl"
    Type="IDL:alma/acsexmplBuilding/Door:1.0"
    Container="Container" />
</Components>
```

## Hierarchical components as directory tree

The first option to build hierarchies consist in putting the layers as:

- one directory per layer with the name of the <LAYER>
- one XML file in each directory named <LAYER>.xml

**TOWER\_1** is such an example:

- A **TOWER**
  - has a main Component
  - and contains a **FRONTDOOR** sub-Component
- Therefore we have:
  - **TOWER\_1** directory
    - **TOWER\_1.xml** file, following the `urn:schemas-cosylab-com:Component:1.0` schema to describe *one single Component*
    - **FRONTDOOR** directory
      - **FRONTDOOR.xml** file, following the `urn:schemas-cosylab-com:Component:1.0` schema to describe *one single Component*

With this solution the xml file in each directory describes just one single component, looking like the following Component xml file: \_

```
<?xml version="1.0" encoding="utf-8"?>
<Component xmlns="urn:schemas-cosylab-com:Component:1.0"
  xmlns:cdb="urn:schemas-cosylab-com:CDB:1.0"
  xmlns:baci="urn:schemas-cosylab-com:BACI:1.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

  Name="FRONTDOOR"
  Code="acsexmplDoorImpl"
  Type="IDL:alma/acsexmplBuilding/Door:1.0"
  Container="Container"
/>
```

On one side this structure makes it very easy to add and remove components. On the other side, it is more difficult to see the the overview of CDB structure from the UNIX command line compared to doing the cut of a single Components.xml file.

The `cdbBrowser` is a good editor and the usage of more advanced UNIX commands can help when dealing with complex MACI / Components / directory structures.

## Hierarchical components as single file

The second option is to group the entire sub-hierarchy into one single file.

**TOWER\_H** is such an example:

- It has as well a **TOWER** as the example above.
- But we have just the **TOWER\_H.xml** file, following the `urn:schemas-cosylab-com:HierarchicalComponent:1.0` schema to describe *the root component and the FRONTDOOR sub-component*:

```
<?xml version="1.0" encoding="utf-8"?>
<HierarchicalComponent xmlns="urn:schemas-cosylab-com:HierarchicalComponent:1.0"
  xmlns:cdb="urn:schemas-cosylab-com:CDB:1.0"
  xmlns:baci="urn:schemas-cosylab-com:BACI:1.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  Name="TOWER_H"
  Code="acsexmplBuildingImpl"
  Type="IDL:alma/acsexmplBuilding/Building:1.0"
  Container="Container" >
  <_ Name="FRONTDOOR"
    Code="acsexmplDoorImpl"
    Type="IDL:alma/acsexmplBuilding/Door:1.0"
    Container="Container" />
</HierarchicalComponent>
```

This structure gives an advantage when we are dealing with true hierarchical component that ***must*** be deployed always together. In this way one single file is sufficient to describe the deployment of the whole hierarchy.

## Deep hierarchy with pure-logical nodes

In many cases it is useful to group Components under logical nodes that do not correspond to actual Components.

For example we want to group all Components in the `Control System` under the hood of `CONTROL`, but `CONTROL` is not a component in itself.

In this case it is just sufficient to create a `CONTROL` directory without putting a `Control.xml` file and then sub-directories for the child layers.

The `CONTROL/ANTENNAS/...` hierarchy is an example of this kind.

## Hierarchy with pure-logical nodes and sub-nodes in one file

One might also want to put multiple sub-nodes inside the same logical node. This is a parallel to the previous case where we have put the definition of both the `TOWER_H` component and of its sub-component(s) `FRONTDOOR` in one single file.

Let's assume we want to do a *similar* thing for `CONTROL/ANTENNAS/ANTENNA_H` and put the entire `ANTENNA` definition in one single file.

Since `ANTENNA_H` is just a logical node and does not correspond to a component, we cannot use the `HierarchicalComponent` schema as above.

But we can use the `Components` schema as for the `Components.xml` file.

We create then the file `CONTROL/ANTENNAS/ANTENNA_H/ANTENNA_H.xml` like this:

```
<?xml version="1.0" encoding="utf-8"?>
<Components xmlns="urn:schemas-cosylab-com:Components:1.0"
  xmlns:cdb="urn:schemas-cosylab-com:CDB:1.0"
  xmlns:baci="urn:schemas-cosylab-com:BACI:1.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xi="http://www.w3.org/2003/XInclude">
  <_ Name="MOUNT"
    Code="acsexmplMountImpl"
    Type="IDL:alma/MOUNT_ACS/Mount:1.0"
    Container="Container" />
  <_ Name="POWER_SUPPLY"
    Code="acsexmplPowerSupplyImpl"
    Type="IDL:alma/PS/PowerSupply:1.0"
    Container="Container" />
</Components>
```

In principle, the whole `CONTROL` configuration could be place in just one single `CONTROL.xml` file, but this is not advisable because we loose the modularity of the configuration database.

## Putting multiple XML files in the same directory using XInclude

In all these examples we always have just one single xml file per directory in the hierarchy.

Since ACS 4.1.0, it is also possible to put multiple xml files in the same directory, each with a number of Components inside, by using the recent [XInclude](#) and [XPointer](#) XML specification.

See [CDBProblems#Multiple\\_Xml\\_Files\\_In\\_Cdb\\_Direct](#) for a detailed discussion about usage and implementation of XInclude and Xpointer in xerces-2 java, our parser (and the patches we have implemented).

- We can put Component specifications in separate files to be included in a main `Components.xml`
- Each of these external files must be a well formed Components description file=
- The Components described in such files are included in the main `Components.xml` using the following syntax: (2)

```
<xi:include href="relative path/MyIncludeFile.xml" xpointer="element(/1)" />
```

**(1) Note:**

Using this technique include XML files can be in principle generally used everywhere in the CDB.  
The only hard constraint is that:

- **the name of the include file cannot be the <directory>.xml**,  
because this is the file that the CDB engine will try to load directly when navigating the CDB structure.

Guidelines, conventions and other approved usages of the XInclude facility are not part of this proposal and will have to be addressed with HLA, SE and IT at a later time.

**(2) Note:**

The XPointer syntax is currently not fully implemented. Only the expression "element(/1)" is supported.

## Example

- We can put in the Components directory (or in a subsystem sub-directory) the file **IncludeTest.xml**

```
<?xml version="1.0" encoding="utf-8"?>
<Components xmlns="urn:schemas-cosylab-com:Components:1.0"
  xmlns:cdb="urn:schemas-cosylab-com:CDB:1.0"
  xmlns:baci="urn:schemas-cosylab-com:BACI:1.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <_ Name="TestInclude_01"
    Code="acsexmplPowerSupplyImpl"
    Type="IDL:alma/PS/PowerSupply:1.0"
    Container="Container"/>
  <_ Name="TestInclude_02"
    Code="acsexmplPowerSupplyImpl"
    Type="IDL:alma/PS/PowerSupply:1.0"
    Container="bilboContainer" />
  <_ Name="TestInclude_03"
    Code="acsexmplPowerSupplyImpl"
    Type="IDL:alma/PS/PowerSupply:1.0"
    Container="bilboContainer" />
</Components>
```

- Using the hierarchical CDB structure describe here above, **CORRELATOR** might have the file **CORRELATOR/IncludeComponents.xml** inside the **CORRELATOR** directory.

- Both files will be included in the main **Components.xml** eventually together with other component specifications:

```
<?xml version="1.0" encoding="utf-8"?>
<Components xmlns="urn:schemas-cosylab-com:Components:1.0"
  xmlns:cdb="urn:schemas-cosylab-com:CDB:1.0"
  xmlns:baci="urn:schemas-cosylab-com:BACI:1.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xi="http://www.w3.org/2001/XInclude">
  <_ Name="ALMA_DOOR"
    Code="acsexmplDoorImpl"
    Type="IDL:alma/acsexmplBuilding/Door:1.0"
    Container="Container" />
  <xi:include href="IncludeTest.xml" xpointer="element(/1)" />
  <xi:include href="CORRELATOR/IncludeComponents.xml" xpointer="element(/1)" />
</Components>
```

This technique is particularly useful to define Dynamic Components in multiple files per each subsystem.


See [FAQDynamicComponentsAndCDBStructure](#) for details.

## Limitations and glitches

With ACS 4.1.3 there are a few problems/glitches using all options described here. These problems will be fixed with the coming ACS releases.

In summary:

- If in the same xml file there are both a component and some subcomponents (for example MOUNT and MOUNT/LAMP), the subcomponents are not parsed correctly

```
%ACTION{ closed="" closer="" cost="1d" created="2005-09-06" creator="Main.GianlucaChiozzi" due="2005-09-15" notify="Main.GianlucaChiozzi" package="jmanager" pri="high" proposer="Main.GianlucaChiozzi" state="open" uid="002445" who="Main.MatejSekoranja" }%  Important ACS 5.0
```

Yesterday and today I have been merging all the examples with hierarchical components in the acsexmpl test CDB.

While doing this I have uncovered a problem with the Manager.  
The file Components.xml contains now the following specifications for components:

```
ALMA_BACK_DOOR
ALMA_BACK_DOOR/LAMP
```

LAMP is supposed to be hierarchically nested under ALMA\_BACK\_DOOR.

With this specification, the Manager recognises ONLY ALMA\_BACK\_DOOR and I do not see any ALMA\_BACK\_DOOR/LAMP in the component lists in the Command Center or in the Manager.

Everything is fine in the CDB, because the cdbBrowser shows what is expected.

If I remove the entry for ALMA\_BACK\_DOOR, then ALMA\_BACK\_DOOR/LAMP appears and I can activate it without problems.

For more details see: [ZLegacy/ACS.CDBProblems](#) %ENDACTION% \* The Manager displays an error message parsing logical nodes that do not contain a real Component. This is only a problem with error handling in the Manager and everything works fine.

- If multiple layers of nodes are put in the same Component specification, they do not appear as nodes in the cdbBrowser tree, but as one node with the name portion.  
For example MOUNT/LAMP above would appear as a MOUNT/LAMP node and not as node MOUNT containing node LAMP.  
This seems to be only a display problem and does not seem to affect functionality, but needs to be investigated. *Functionality is OK and we do not need to know anything, but being aware of the fact --* [GianlucaChiozzi](#) - 06 Sep 2005

For more details see: [ZLegacy/ACS.CDBProblems](#)

```
%ACTION{ cost="1d" created="2005-02-21" creator="Main.GianlucaChiozzi" due="2005-04-30" package="cdb" pri="normal" proposer="Main.GianlucaChiozzi" state="open" uid="001962" who="Main.AlessandroCaproni" }% FAQHierarchicalComponentsAndCDBStructure - Cleanup the list of problems when they are solved. %ENDACTION%
```

## Characteristic Components and CDB for instances configuration

Characteristic Components keep their own instance configuration in the CDB in the **alma** branch.

Each component has a node there with the same hierarchical structure as the Component name.

So the component:

- CONTROL/ANTENNAS/ANTENNA\_H/MOUNT

has an entry in the CDB at:

- alma/CONTROL/ANTENNAS/ANTENNA\_H/MOUNT

See also [ZLegacy/ACS.FAQGeneralCompCDB](#) for details.

## How to try out the example CDB

- Unpack the [CDB.tar.gz](#) file somewhere
- Set the environment variable `ACS_CDB` to the place where the CDB has been unpacked.
- Start ACS with the ACS Command Center or with *acsStart*
- Browse the CDB with the `cdBrowser`
- Start a CPP Container called `Container`
  - All `Components` defined in this example CDB are configured to run in this `Container`
  - Their implementation is available in a normal ACS installation as part of *acsexmpl*
- Use the Object Explorer to activate/deactivate the `Components` and say hierachical names.

-- [GianlucaChiozzi](#) - 06 Sep 2005

## Related articles

- [How can more people do development with ACS on the same machine without disturbing each other?](#)
- [Which ports are used by ACS?](#)
- [Problems connecting to ACS servers on a remote machine: bad /etc/hosts](#)
- [Why does the `getComponent` method of `ZLegacy/ACS.ContainerServices` return an object of type `None`?](#)
- [Why are some of my print statements not showing up in the container output section of `acscommandcenter`?](#)