Context File System Users Manual

Christopher Hess

{ckhess@cs.uiuc.edu}

1 Overview

The Context File System (CFS) is the means to access data in Gaia. All data sources are represented as containers of data objects, where each container holds objects of a particular type. Since containers are typed, there are well-known input and output data types that defines each container. For example, a PowerPointContainer takes a ".ppt" file as input and has slides as output. Therefore, the objects within the container are GIF slides. Applications open data sources in the format that they require. If one container does not satisfy the input data source type and output type the application requires, it may be necessary for several containers to be connected together to achieve the correct conversions. If such containers are available, the system will transparently link them together, therefore providing the application with the correct type. For example, if a PalmPilot opened the ".ppt" file as a BitmapContainer, it could receive bitmap slides of the PowerPoint presentation. The system would connect the BitmapContainer to the PowerPointContainer. Thus, the output would be the desired format of the application.

2 Architecture

CFS is composed of several basic components, including the mount server (Layout Manager), file server (Container Manager) and the resolver (DOSCore). These three main components are described below. Note that the implementation is referred to as the Data Object Service (DOS) for historical reasons.

2.1 ContainerManager

The ContainerManager acts as a factory for creating containers and performing any necessary container adaptations. In addition, it is responsible for managing all disk data that is resident on the machine it is running. When clients request access to data sources, the ContainerManager creates a container to represent the source and returns it to the client. The client may then get and put objects specific to the container type.

Another task of the ContainerManager is to instantiate additional containers when one single container does not satisfy the input and output types the user requires. In this case, the manager looks for a suitable group of containers that may be linked together to

perform any necessary conversions of data types to present the data to the application in the desired format.

2.2 LayoutManager

The LayoutManager is responsible for telling clients how they should construct their data namespace for a physical space, essentially defining the data layout for a particular entity. There is a single LayoutManager for each physical space that is configured for the storage in the particular space. A configuration file specifies what machines should export what parts of their disk and how the combined disks from all machines in a space should be organized into a single namespace. The LayoutManager describes the namespace in an XML file with describes that machines export what storage. Context hints may be added to descriptions to make the data only available in a particular context. The DOSCore, described below, performs this construction. These XML descriptions are similar to mount points in traditional file systems.

Layout descriptions differ from mount points in that they are dynamic. They may change depending on where a device or entity is located, or for what entity the namespace is being constructed. For example, when a person enters a new space, the system will query the service for the local layout of data storage for the room. The user may be allocated some local storage in the physical space or temporary storage may be added to the users' namespace.

In addition, entities may export storage that they posses. For example, a user may enter a space with a device that contains some storage. That storage may be mounted into the namespace of the physical space, which can then access the exported storage. A possible scenario would be treating a small handheld device as a mobile storage repository. A user could enter a space where a presentation is to be given. The users' mobile device may host the actual presentation, while the space contains the presentation software and projector. After entering the space and having the mobile device automatically mounted into the space, the presentation software could find the presentation from the mobile, since it is now a part of the namespace of the space.

2.2.1 Layout Configuration File

The LayoutManager configuration file provides a description of the virtual namespace. Tags specify the mount point, the native file system path, who owns the mount, and context tags. The following is an example configuration file:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE DOS:Layout SYSTEM "DOSLayout.dtd">

<DOS:Layout xmlns:DOS="http://choices.cs.uiuc.edu/gaia/DOS">

      <DOS:Storage>
            <DOS:Owner>ckhess</DOS:Owner>
            <DOS:Mount>/temp-desktop</DOS:Mount>
```

```
            <DOS:Host>srg181</DOS:Host>
            <DOS:Path>C:\Temp</DOS:Path>
      </DOS:Storage>

      <DOS:Storage>
            <DOS:Owner>ckhess</DOS:Owner>
            <DOS:Mount>/ckhess</DOS:Mount>
            <DOS:Context>
                  <DOS:Location>2401</DOS:Location>
            </DOS:Context>
      </DOS:Storage>

      <DOS:Storage>
            <DOS:Owner>system</DOS:Owner>
            <DOS:Mount>proxy</DOS:Mount>
            <DOS:Host>srg181</DOS:Host>
      </DOS:Storage>

      <DOS:Storage>
            <DOS:Owner>system</DOS:Owner>
            <DOS:Mount>/temp</DOS:Mount>
            <DOS:Host>srg181</DOS:Host>
            <DOS:Path>C:\Temp</DOS:Path>
      </DOS:Storage>

</DOS:Layout>
```

In order to export the storage specified in the Path tag, a ContainerManagers must be running on the specified machine. The DOSCore lazily builds its mapping tables, caching the ContaierManager references after they are accessed for the first time. Therefore, non-existing mangers will signify exceptions when they are accessed, not at DOSCore startup time. Make sure that the mapping are valid for the platform you are running on.

Context tags may be attached to files and directories to make them available in a particular context. Available tags are Situation, Group, Location, Space, and Time. These tags are automatically added with copying a file to a context directory.

2.3 DOSCore

The main client component is the DOSCore. This component provides lookup services and maintains the current mount table entries. When a client wishes to open a data source, the core resolves the pathname and gets a handle to the ContainerManager on the machine hosting the data. The mount table is used to convert from the local namespace to the actual native directory on a machine.

3 Compiling the Context File System

To compile CFS, first make sure that the Unified Object Bus (UOB) libraries and executable are available. Checkout DOS from the CVS repository. For WinNT, open the workspace in DataObjectService/Build/WinNT. On Solaris, type “gmake” in

DataObjectService/Build/Solaris. The client demo component will also be compiled (a simple shell) which can be used to test the servers.

4 Context File System Templates and Wrappers

CFS wraps the CORBA details in templates and wrapper classes. Templates allow code reuse for accessing the CORBA client stubs, while handling the different container data types. Although the templates ease development of the client access to CORBA, application programmers should not be exposed to the list of template parameters. Therefore, the template classes are hidden with light wrapper classes. These wrappers simply make the API to data more clean and easy to use.

5 Container Catalog

Container Type	Data Type
DirectoryContainer	Dirents
ByteContainer	Bytes
PowerPointContainer	GIFs
WordContainer	Bytes
TextContainer	Chars
GIFContainer	GIFs
PixmapContainer	Pixmaps
BitmapContainer	Bitmaps

6 Using the Context File System

To start using CFS, the LayoutManager must be running with the configuration file setup so that data can be accessed from a CFS client:

ComponentContainer SYSTEM –c Base/ORBACUSExporter, -c Base/CORBAComponentManager –c CORBA/LayoutManager –d+ –f <config file>

The <config file> contains the mappings to create the client storage namespace. Examples can be found in DataObjectService/Script directory.

Once the LayoutManager is running and registered in the Name Service, the ContainerManager may be run to allow access to the data on that machine:

ComponentContainer SYSTEM –c Base/ORBACUSExporter, -c Base/CORBAComponentManager –c CORBA/ContainerManager –d+

For testing, it is possible to run the LayoutManager and ContainerManager in the same UOB:

ComponentContainer SYSTEM –c Base/ORBACUSExporter, -c Base/CORBAComponentManager –c CORBA/LayoutManager –d+ –f <config file> -l<layout manager context> –c CORBA/ContainerManager

The *Tests* directory contains several sample scripts. dosd.bat contains the script defined above and can be used for testing. DOSLayout.win.cfg and DOSLayout.solaris.cfg are sample configuration files that can be passed to the LayoutManger. They may be edited to alter the mappings. A test shell program (DSH) may be used to test the running components and is also available in the *Tests* directory.

7 Programming with CFS

The low-level system details described above are hidden from the application developer by the wrapper classes. To gain access to a particular data source, simply instantiate a container of the desired type.

For example, to open a directory and read its contents, the following may be used:

```
try {


    DirectoryContainer c(<dirname>, DOS::Read);

    int count;
    Dirents dirent;

    while ((count = c.get(dirents, 16)) > 0) {
        for (int i = 0; i < count; i++)
            cout << dirents[i].name << endl;
    }

    c.close();

} catch (...) {
  cerr << "error" << endl;
}
```

To list the contents of a file, the ByteContainer may be used. ByteContainers are for unstructured data or byte streams. These also maintain the more traditional view of files as byte streams.

```
try {

    ByteContainer c(<filename>, DOS::Read);

    int count;
    Bytes bytes;

    while ((count = c.get(chars, 256)) > 0)
      cout.write(bytes, count);

    c.close();
```

```
} catch (...) {
  cerr << "error" << endl;
}
```

To display PowerPoint presentations, files may be opened as a PowerPointContainer. If the file was not a .ppt, the creation of the container would fail, since the input type was incorrect.

```
try {

    GIFViewer viewer;
    PowerPointContainer c(<filename>, DOS::Read);

    int count;
    GIFs gifs;

    while ((count = c.get(gifs, 1)) > 0) {
        // Display the slide.
        viewer.display(gif[0]);
    }

} catch (...) {
  cerr << "error" << endl;
}
```

To display PowerPoint slides on a PalmPilot, the following could be used:

```
try {

    BitmapViewer viewer;
    BitmapContainer c(<filename>, DOS::Read);

    int count;
    Bitmaps bitmaps;

    while ((count = c.get(bitmaps, 1)) > 0) {
        // Display the slide.
        viewer.display(bitmap[0]);
    }

} catch (...) {
  cerr << "error" << endl;
}
```

The system will instantiate a PowerPointContainer to open the .ppt file, then convert the data objects to bitmaps. However, the application merely requests that the source be opened to contain bitmap data objects.

7.1 Modes

There are several modes when opening containers. They are:

```
DOS::Read
DOS::Write
DOS::Append
DOS::New
```

These get mapped to the modes in standard `fopen` for ByteContainers.

7.2 Compiling Programs Using CFS

When compiling a program to use DOS, the file DataObjectService/DOSCore/DOS.h must be included within the module once. Also, the stubs for the containers being used should also be compiled in. For example, to use a ByteContainer within a program, ByteContainerC.cpp should be compiled into the project.