Problem

How is the interface repository loaded?

Solution

The Interface Repository is a standard CORBA Service that is loaded with the interfaces described in IDL files and provides applications with a dynamic description of these interfaces.

You can think of the Interface Repository as a *language independent mechanism for introspection.*

The Interface Repository is used by generic applications that build dynamically their access to CORBA objects.
The most important example is the **`ACS Object Explorer (objexp)`**

Interface Repository startup

When ACS is started, the Interface Repository service is started together with other basic services.

In older versions of ACS, this was done in the **`acsStartORBSRVC`** script. As of ACS 8.0, the actual management of the Interface Repository has been shifted to the **acsInterfaceRepository** script. **acsStartORBSRVC** and **acsStart** both call this script as part of their start-up process.

Since ACS 3.1, we are using the MICO implementation of the Interface Repository and the command line to start the service is:

```
$MICO_HOME/bin/ird -ORBNoResolve -ORBIIOPAddr inet:$HOST:$ACS_IR_PORT --ior $IR_IOR &
```

Interface Repository loading

Initially the interface repository is empty.
It is necessary to load into it the interfaces contained in the IDL files.
The ACS startup procedure tries to load into the interface repository the contents of ***ALL IDL files it can find***, unless the --noloadifr option is passed to **acsStart**.

Before ACS version 5.0.3, this is done once more in the **`acsStartORBSRVC`** script by calling the **`acsIrfeed`** script:

```
acsIrfeed  -IRcorbaloc corbaloc::$HOST:$ACS_IR_PORT/InterfaceRepository &
```

Since ACS version 5.0.3, **acsstartupLoadIFR** has been used, called by **acsStartORBSRVC**, to load the interface repository. As of ACS 8.0, this too has been moved to the **acsInterfaceRepository** script.

Since there are usually many IDL files, this can take quite some time.

The Interface Repository Loading Process

Loading all IDL files in the system

The **`acsstartupLoadIFR`** script does the following:

1. Calls a C++ program that builds a list of all IDL files from the directories listed in the *$IDL_PATH* environment variable and generates a temporary IDL file that simply includes all these file.
2. Loads the generated "global" IDL file in the Interface Repository itself by calling the MICO IDL compiler with the following command line:

```
        $MICO_HOME/bin/idl $IDL_PATH --no-codegen-c++ --feed-ir -ORBIfaceRepoAddr inet:$HOST:$ACS_IR_PORT
$ACS_IDL_IRFEED_FILE
```

Notes:

1. This value is set by **.bash_profile.acs** in the order $INTROOT/idl, the idl directory from each directory given in $INTLIST, and $ACSROOT/idl.
2. Generates a temporary IDL file that simply includes all these file in discovery order.
It makes sure that no files are duplicated and that the first file found in the path is used, so that a version of file installed in `INTROOT` has precedence over an older version in ACSROOT.
3. acserr.idl is always the first file loaded because of problems with MICO Interface Repository loader.
4. ACSIRSentinel.idl is always the last file loaded. It is used as a marker to indicate that the loading process has completed.

The MICO IDL compiler is instructed by the **`--feed-ir`** option to generate information suitable for the Interface Repository service.

But this step is from any respect equivalent to compiling a big IDL file that contains all interfaces known to the system.

If there is any problem/error with this compilation, the process is aborted and the loading into the Interface Repository is left incomplete.

A typical problem is when some interface is duplicated in more IDL files. This problem is often not discovered when compilling the IDL files in isolation to build skeletons and stubs for coding, but appears immediatly when loading all interfaces at the same time into the Interface Repository.

Other problems can also appear because of incompatibilities between the MICO IDL compiler and the other IDL compilers used in ACS. Code that is accepted by these compilers can be rejected or not understood properly by the MICO one.

Note: As of ACS 6.0.4, ACS provides a check option on **acsstartupLoadIFR** which dumps a copy of the generated "global" IDL file.

Load just specific IDL files

It is also possible to load manually specific IDL files by listing them in the **`acsstartupLoadIFR`** command line:

```
   acsstartupLoadIFR -IRcorbaloc corbaloc::$HOST:$ACS_IR_PORT/InterfaceRepository idlfile1.idl idlfile2.idl
.....
```

Note: The directory where the idl files reside must be in the `IDL_PATH`

This allows you also to reload an already loaded IDL interface without having to stop and restart the whole ACS after a minor change.

Browsing the Interface Repository

Bundled with ACS you get the JacORB Inteface Repository Browser.
This tool allows you to graphically browse the Interface Repository and is extremely valuable to debug problems with Interface Repository loading.

You can launch the tools from the ACS Command Center or following the instructions in FAQJacORBTools

-- GianlucaChiozzi - 27 Jul 2004

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?