MAKING THE DCE 1.1 SERVICEABILITY AND MESSAGE API'S PUBLIC

1. INTRODUCTION

Most people who have seen the DCE 1.1 serviceability proposal [RFC 24] have said that they want to facility made available for general use. We expect that many DCE application programmers would use the API if it were available.

This note presents a proposal on how this could be done. In addition to technical answers, it calls out the known political and logistical issues involved, although it does not provide any solutions for them. We are interested in receiving feedback on these issues.

2. BACKGROUND

DCE 1.0 provides an architected 32-bit number known as a status code, known by the C datatype `error_status_t`. This number is partitioned as follows:

```
HIGH                       LOW
31 .. 28    27 .. 12   11 .. 0
Technology  Component     Code
```

The special value of all-bits-zero is reserved to indicate success.

Two technologies are currently in use: DCE (0) and DFS (1). One has been reserved for DME (2). One has been ''appropriated'' by a DCE application vendor (7). There are fifteen possible technologies.

The component is a ''rad-40'' word of three characters. In DCE 1.0 the alphabet was `a..z{0..9` with the last three characters dependent on the local machine's collating sequence. For DCE 1.1 the alphabet is `a..z_0..9-+@`. (Since neither the `{` character nor the unspecified characters were used, this is an upward-compatible change.) Within DCE we have tried to pick mnemonic component names such as ''sec'' for security and ''dts'' for the time service. For example, ''rpc'' maps into the three numbers 17, 15, and 2, This is packed into a single number as 17 * 40 * 40 + 15 * 40 + 2, which is 27802 decimal or 0x6C9A hex. There are 64,000 possible components within a technology.

The code is an index into the message catalog. Note that according to [XPG4], message numbers start with one so there are 4095 possible messages within a component.

In DCE 1.0, the the routine `dce_error_inq_text` converts the technology and component fields back into strings. This is done by mapping the technology number back to a fixed string, and by reverse-engineering the rad-40 calculation. This results in a message catalog name such as ''dcedts.cat'' which is passed into a `catopen` call. The code is masked off and passed into a `catgets` call. When the message is retrieved, the technology and component strings are appended. This results in an error message like the following:

```
time service has not been enabled (dce / dts)
```

In DCE 1.1, the same decoding action is done by the `dce_msg_cat_open` routine to retrieve a message from a catalog. The primary difference is that the technology and component are not appended.

To summarize: the technology and component determine the name of the message catalog, and only `dce_error_inq_text` displays this information. In DCE 1.1 this will be made a bit more explicit because of the documentation, and the need to use the `sams` tool to create message and serviceability tables.

3. PROPOSAL

OSF will allocate a specific technology for third-party use, which we will call ''ISV''. Components within that technology will be assigned upon request. We expect that almost all requests will come for DCE developers as we would not advertise the API outside of DCE. We are particularly interested in hearing from members of the DCE community about how interested they are; i.e., any estimates on how many requests they would make.

We have not yet determined which number to assign to this component. From a programming viewpoint, it is easiest to pick three. There is political appeal to picking seven; it also has technical merit if we want to leave space for additional OSF offerings, and stay away from the high-bit-one number space.

Within OSF there was concern raised that 4095 messages might not be sufficient. Because of this we would change the messaging routines to recognize the ISV technology so that for this technology the component and code fields would be equally divided into 14 bits each. This would allow 16,383 messages. Rather than using the rad-40 encoding, we would use the text representation of the number to determine the name of the message catalog. This would allow 16,384 components.

Adding the code to perform this alternate mapping would be simple, probably requiring under 50 lines of code.

It would not be appropriate for `dce_error_inq_text` to display the technology and component after the message; ‘‘(isv/42)’’ is fairly useless. Instead, we would require that message code one be used to identify the application. When displaying a message in the ISV technology, `dce_error_inq_text` would also retrieve message one within that component and append it to the original message. This would result in messages like the following:

```
Mailbox full (Foobar Mailserver(tm))
```

Because the serviceability interface is an interface that all DCE 1.1 servers will export, it has already been written in a ‘‘generic’’ manner, including a set of convenience functions so that it can be integrated into any existing ACL model.

4. LIMITATIONS AND BENEFITS

The DCE messaging API does not support message catalog sets.

A registrant really only receives two things:

(a) A unique set of numbers for status codes and message ID's.

(b) A unique name for their message catalog.

There are two important benefits that result from this, however:

(a) ISV servers can use the DCE 1.1 serviceability interface and routines.

(b) Applications could invoke other ISV RPC's and understand the failures without having to make special arrangements.

5. UNRESOLVED ISSUES

We do not know how much work would be involved for OSF to maintain this registry. Keeping a simple list of assigned numbers is fairly straightforward, but the demand is unknown. We do not know how much effort will be required to make sure that vendors do not register the same product multiple times. We do not plan to add additional registry information, but experience with the code set registry [RFC 40] indicates that people may request this.

Preliminary manual pages exist for the API's and message tool. More documentation would have to be written, most likely in the *Application Development Guide*. In addition, some documentation would be needed that explains how to integrate the serviceability remote interface.

REFERENCES

[RFC 24] R. Salz, DCE-RFC 24.1, ‘‘DCE 1.1 Serviceability Proposal’’, April, 1993.

[RFC 40] S. Martin, DCE-RFC 40.0, ‘‘OSF Character and Code Set Registry’’, April 1993.

[XPG4] *X/Open XPG4 Portability Guide*, message catalog routines.

AUTHOR'S ADDRESS

Rich Salz
Open Software Foundation
11 Cambridge Center
Cambridge, MA 02142
USA

Internet email: rsalz@osf.org
Telephone: +1-617-621-7253