

RS/6000 Cluster Technology



First Failure Data Capture Programming Guide and Reference

RS/6000 Cluster Technology



First Failure Data Capture Programming Guide and Reference

Note!

Before using this information and the product it supports, be sure to read the general information under "Notices" on page 213.

First Edition (April 2000)

This edition applies to:

- Version 3 Release 2 of the IBM Parallel System Support Programs for AIX (PSSP) Licensed Program (product number 5765-D51) and to all subsequent releases and modifications until otherwise indicated in new editions, and
- The Enhanced Scalability feature of Version 4 Release 3 of the IBM High Availability Cluster Multi-Processing for AIX (HACMP) Licensed Program (product number 5765-D28) and to all subsequent releases and modifications.

Order publications through your IBM representative or the IBM branch office serving your locality. Publications are not stocked at the address below.

IBM welcomes your comments. A form for readers' comments may be provided at the back of this publication, or you may address your comments to the following address:

International Business Machines Corporation
Department 55JA, Mail Station P384
2455 South Road
Poughkeepsie, NY 12601-5400
United States of America

FAX (United States & Canada): 1+914+432-9405

FAX (Other Countries):

Your International Access Code +1+914+432-9405

IBMLink (United States customers only): IBMUSM10(MHVRCFS)

IBM Mail Exchange: USIB6TC9 at IBMMAIL

Internet e-mail: mhvrdfs@us.ibm.com

If you would like a reply, be sure to include your name, address, telephone number, or FAX number.

Make sure to include the following in your comment or note:

- Title and order number of this book
- Page number or topic related to your comment

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

© **Copyright International Business Machines Corporation 1999, 2000. All rights reserved.**

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Figures	vii
Tables.	ix
About This Book	xi
Checking Which Level(s) of Code You Have	xi
PSSP	xi
HACMP	xi
RSCT	xi
Who Should Use This Book	xii
Typographic Conventions.	xii
Preface	xiii
Prerequisites	xiii
Limitations of Code Examples	xiii
Overview of the Chapters and Their Contents	xiii
Basic FFDC Terminology	xiv
Chapter 1. Introduction to First Failure Data Capture	1
FFDC Environment	4
Creating a Basic FFDC Environment	5
Creating an FFDC Error Stack Environment	5
Inheriting an FFDC Error Stack Environment	6
Creating an FFDC Error Stack Environment Instead of Inheriting	7
When the FFDC Error Stack Is Most Useful	8
What Is The Scope of an FFDC Error Stack?.	9
Should the Software Support Possible Use of an FFDC Error Stack?	11
Recording Failure Information	12
Recording in a Basic FFDC Environment	12
Recording in an FFDC Error Stack Environment	13
Reporting Results	14
Informing the Client of the Failure Information	14
Informing the Client At Execution Time.	14
Returning an FFDC Failure Identifier in a Message	17
Chapter 2. Installing and Maintaining FFDC	19
First Failure Data Capture Installation Media	19
First Failure Data Capture Directory.	19
Preventing Full Consumption of the /var File System	19
Maintenance of the First Failure Data Capture Directory	20
Disabling First Failure Data Capture Error Stack Usage	20
Diagnosing Common First Failure Data Capture Utility Problems	21
Chapter 3. Constructing Persistent Records of Failures	23
What Failure Recording Facility Should Be Used?	23
When the AIX Error Log Is Appropriate	23
When the FFDC Error Stack Is Appropriate	23
When AIX Error Log and FFDC Error Stack Are Used	24
Which Storage Works Best For My Software Type?	24
Constructing Meaningful Templates for AIX Error Logging.	26
AIX Error Log Template Now Support XPG/4 Message Catalogs	28
Selecting An Appropriate Log Event Type.	28
Carefully Describe the Failure Condition	30

Organize the Causes and Their Associated Actions	31
Do Not Place Descriptive Information in Detail Data	32
Making Correct Use of Detail Data	32
Other Rules Governing the Contents of an AIX Error Log Template	34
Using the AIX Error Log via the FFDC Utilities	35
The FFDC Failure Identifier token: the Link Between Related Failures	35
Required Information In Error Log Templates Used By FFDC Clients	36
How the FFDC Failure Identifier Is Passed Back	36
Recording Information to the AIX Error Log and the BSD System Log	37
The Programming Model.	37
Making Effective Use of the FFDC Error Stack.	38
The FFDC Error Stack Environment.	38
The FFDC Error Identifier: the Link Between Related Failures	39
How the FFDC Error Identifier Is Passed Back.	39
Recording Information To the FFDC Error Stack	39
The Programming Model.	40
Chapter 4. First Failure Data Capture Utilities	41
Programming Interfaces	41
fcinit/fc_init	41
fcteststk/fc_test_stack	42
fclogerr/fc_log_error	42
fcpushstk/fc_push_stack	43
fcdispfid/fc_display_fid.	44
fcfilter.	44
End-User Commands	44
fcdecode	44
fcreport	44
fcstkrpt	44
fcclear	45
fccheck	45
FFDC Utilities Manual Pages	45
<rsct/ct_ffdc.h> Header File.	46
fc_init Subroutine	48
fc_eid_init Subroutine	52
fc_eid_is_set Subroutine	53
fc_display_fid Subroutine.	55
fc_test_stack Subroutine	57
fc_push_stack Subroutine	59
fc_log_error Subroutine	66
fcinit Command	74
fcteststk Command	78
fcpushstk Command	80
fclogerr Command	87
fcdispfid Command	95
fcdecode Command	97
fcfilter Command.	99
fcstkrpt Command.	101
fcreport Command	103
fcclear Command	105
fccheck Command	108
Appendix A. Examples	111
Parent/Child Source-Code Examples	111
The parent.sh Script	111
The child1.c Program	119

The child2.sh Script	125
The grandp.c Program	133
The utils.c Utility Code Module	140
The ffdcxpl.msg Message Catalog File	145
Makefile for the Example Code	147
Daemon Source-Code Example	148
The client.c Program	148
The ffdclnt.err.E Error Log Template File	175
The ffdclnt.msg Message Catalog File	183
Appendix B. First Failure Data Capture Messages — 2615	187
Notices	213
Trademarks	214
Publicly Available Software	215
Software Update Protocol	215
Glossary of Terms and Abbreviations.	217
Bibliography	225
Information Formats	225
Finding Documentation on the World Wide Web	225
Accessing PSSP Documentation Online	225
Manual Pages for Public Code	226
RS/6000 SP Planning Publications	226
RS/6000 SP Hardware Publications	226
RS/6000 SP Switch Router Publications	226
RS/6000 SP Software Publications	227
AIX and Related Product Publications	229
DCE Publications	229
Red Books	229
Non-IBM Publications	229
Index	231
Readers' Comments — We'd Like to Hear from You	233

Figures

1.	Example of Complex Failure Situation	1
2.	Diagnosing a Complex Failure Situation	2
3.	Determining Root Cause of a failure with FFDC	3
4.	Tracking Failures with FFDC	4

Tables

1.	Methods for Establishing FFDC Environments	5
2.	Recording Failure Information	12
3.	First Failure Data Capture Log Event Types	28
4.	Mapping FFDC Types to AIX Error Log and BSD System Log Types	29
5.	Mapping Software Incidents to FFDC Event Types	29

About This Book

This book contains information to help you understand the First Failure Data Capture (FFDC) utilities. For information regarding the SP system and related software, refer to the appropriate documentation listed in the bibliography.

For a list of related books and information about accessing online information, see the bibliography in the back of the book.

Checking Which Level(s) of Code You Have

This book applies to PSSP Version 3 Release 2 and to HACMP/ES at the HACMP Version 4 Release 3 level.

It also applies to RSCT Version 1.2.

To find out which level(s) of code you have running on your system, follow the instructions in the sections below.

PSSP

To find out what version of PSSP is running on your control workstation (node 0), enter the following:

```
sp1st_versions -t -n0
```

In response, the system displays something similar to:

```
0 PSSP-3.2
```

If the response indicates **PSSP-3.2**, this book applies to the version of PSSP that is running on your system.

To find out what version of PSSP is running on the nodes of your system, enter the following from your control workstation:

```
sp1st_versions -t -G
```

In response, the system displays something similar to:

```
1 PSSP-3.2
2 PSSP-3.2
7 PSSP-3.1.1
8 PSSP-2.4
```

If the response indicates **PSSP-3.2**, this book applies to the version of PSSP that is running on your system.

If you are running mixed levels of PSSP, be sure to maintain and refer to the appropriate documentation for whatever versions of PSSP you are running.

HACMP

To find out which version of HACMP is running on a particular node, enter the following:

```
ls1pp -L | grep cluster.es.server.rte
```

RSCT

To find out which code version of IBM RS/6000 Cluster Technology is running on a particular node, enter the following:

Who Should Use This Book

This book is intended for system administrators responsible for setting up and maintaining the SP System. It assumes that readers have a working knowledge of AIX or UNIX and experience with network systems. It is also assumed that the user has experience with the Scalable POWERparallel Systems and has access to the IBM Parallel Systems Support Programs for AIX Library.

Typographic Conventions

This book uses the following typographic conventions:

Typographic	Usage
Bold	<ul style="list-style-type: none"> • Bold words or characters represent system elements that you must use literally, such as commands, flags, and path names.
<i>Italic</i>	<ul style="list-style-type: none"> • <i>Italic</i> words or characters represent variable values that you must supply. • <i>Italics</i> are also used for book titles and for general emphasis in text.
Constant width	Examples and information that the system displays appear in constant width typeface.
[]	Brackets enclose optional items in format and syntax descriptions.
{ }	Braces enclose a list from which you must choose an item in format and syntax descriptions.
	A vertical bar separates items in a list of choices. (In other words, it means “or.”)
< >	Angle brackets (less-than and greater-than) enclose the name of a key on the keyboard. For example, <Enter> refers to the key on your terminal or workstation that is labeled with the word Enter.
...	An ellipsis indicates that you can repeat the preceding item one or more times.
<Ctrl-x>	The notation <Ctrl-x> indicates a control character sequence. For example, <Ctrl-c> means that you hold down the control key while pressing <c>.

Preface

This publication describes the First Failure Data Capture (FFDC) utilities, which assist C language, C++ language, and script language programmers in implementing serviceability in the source code. RSCT software uses the FFDC utilities to help achieve serviceability within the software by ensuring the software is consistent in its reporting of failure information, giving the customer and IBM Customer Support a reasonable assumption as to the content and clarity of failure information. This consistency makes the task of understanding and isolating failures easier for those who have to perform that task. These utilities also provide the capability for the software writer to establish a link between related failure conditions, making it possible to follow a failure from its symptom to its cause.

Prerequisites

The reader is assumed to be familiar with using a Unix system and programming C language programs to run on a Unix system. It is also assumed that the reader is familiar with the C standard library and the system calls provided by a Unix system for a C language program. The reader is assumed to be either a designer of software intended for Unix systems in general and AIX systems specifically, or a programmer writing software for these platforms. Knowledge of the customary shell environments used by Unix systems and AIX platforms, such as Bourne, Korn, and C shells, is also assumed. The reader is expected to understand Unix process organization, and relationships between parent and child processes on a Unix system. Finally, some familiarity with network programming is assumed.

Statements made about AIX functionality may not be applicable to releases prior to AIX 4.3.2. This is especially true in the case of AIX Error Log template design, which has been enhanced.

Limitations of Code Examples

Source code examples are presented in this document with the intent of demonstrating serviceability issues within source code and how these issues impact the design of the source code. Some of these examples are complete, and can be compiled and executed as presented. Others are code fragments intended for demonstration purposes only, and would require modification in order to compile or execute on a Unix or AIX platform. These code examples are designed to be free of error, except in cases where the example is specifically designed to demonstrate an error scenario. If the reader should discover an unintentional error in any of the source code examples, please bring the error to the attention of IBM.

Code examples presented in this document are intended for the sole purpose of illustrating FFDC and its underlying concepts. These examples may not demonstrate actual meaningful or useful code segments.

Overview of the Chapters and Their Contents

Chapter 1, "Introduction to First Failure Data Capture," introduces and demonstrates the First Failure Data Capture Utilities.

Chapter 2, "Installing and Maintaining FFDC," discusses the installation media and maintenance of the FFDC data capture directory. It discusses how to disable the creation of First Failure Data Capture error stacks on a node.

Chapter 3, "Constructing Persistent Records of Failures," gives special attention to recording information about failure. It describes how to make the best use of the AIX Error Log, BSD System Log, and FFDC Error Stack.

Chapter 4 describes the First Failure Data Capture utilities.

Appendix A provides specific source code examples that were designed for serviceability, and which exploit the FFDC interfaces.

Appendix B contains First Failure Data Capture messages.

Basic FFDC Terminology

The following terms are used in this publication with the following meanings:

AIX Error Log

A storage facility provided by the AIX operating system for recording information about failures and significant incidents to persistent storage. This log exists on each AIX system in the `/var/adm/ras/errlog` file, and is implemented as a circular file.

Associated failure

A failure that occurs as a direct result of another failure. An example of this type of failure is a failure in client application that is caused by a failure in a service it was using. Such failures are usually symptoms or effects of the initial failure.

BSD System Log

A storage facility provided in BSD compliant systems for recording information about significant events. This log can be configured to be a file or a notification system. System administrators define the location of the log. When recording to a file, the log is implemented as a flat file, and can grow to consume all available disk space if not properly maintained. The use of the BSD System Log is supported for compatibility with other UNIX platforms, but the AIX Error Log is the preferred location for such information. This manual may use the phrase AIX Error Log to refer to both the AIX Error Log and the BSD System Log at the same time.

client

Any user of software at any point in time. A "client" can be a script that invokes a command, an application requesting specific action from a system service, a subroutine of a module making a call to another subroutine or invoking a routine from a programming library, an application issuing a system call, a daemon of a distributed application or service contacting a peer daemon on a remote node, or a module that attempts to use a system resource. The definition is purposely broad, and is used to mean "any user or software that requests a function".

FFDC Environment

A UNIX application execution environment tailored to make efficient use of the First Failure Data Capture utilities. Two types of environments are provided by the FFDC utilities:

- A basic FFDC environment that permits the software to use FFDC utilities in recording information to the AIX Error Log and BSD System Log facilities.
- An FFDC Error Stack Environment that permits the software to make use of an FFDC Error Stack in addition to the AIX Error Log and BSD System Log facilities.

FFDC Error Stack

A storage facility provided by the First Failure Data Capture utilities to record information about failures and significant incidents to persistent storage. The FFDC Error Stack is designed to capture information for a set of processes or threads working together in a common function. This concept was introduced in the section "Making Effective Use of the FFDC Error Stack" on page 38; consult that section for further details.

FFDC Failure Identifier

A 42-character token generated by the First Failure Data Capture recording facilities that uniquely identifies the failure report. This token can be used by the FFDC commands to locate the exact

failure report anywhere in the system. Associated failures can incorporate the FFDC Failure Identifier returned by a service as part of its failure report, establishing a link between the associated failure and its root cause.

Initial failure

The first failure in a sequence of related failure conditions or an isolated failure condition. A failure of the type may be a cause if another failure occurs because this failure condition is present.

resource

Anything that software utilizes to complete its function that cannot be classified as "software". Some examples of resources are files, devices, memory, sockets, shared memory, and networks.

root cause

The source of a failure. A root cause may manifest itself as one or more symptoms that result because of the failure.

service

Any software that is being asked to perform a function by other software or an end user. A "service" can be a system service running as a daemon, a subroutine invoked by another subroutine in the same module, a library routine, a command invoked from a shell or an end user, a kernel extension, a daemon of a distributed application or service being contacted by another of its peers from a remote node, or a system resource. The definition is purposely broad, and is used to mean "any software that attempts to perform a function requested by someone else."

serviceability

The software's capacity to be properly and correctly serviced. This involves:

- Designing and implementing software to anticipate and detect failure conditions
- Correctly identifying failure conditions when they do exist
- Notifying appropriate clients and administrators of failure conditions when they do exist
- Reporting adequate information about failure conditions so that the nature of the failure and its impact on the software's function can be understood
- Recording persistent records of failures for use in problem isolation and determination efforts
- Applying repairs to software after the software is installed.

software

Any subsystem, application, command, script, utility, library, or licensed program product (LPP) for AIX.

symptom

An effect caused by a failure condition. Symptoms usually do not provide enough information to determine the failure that caused this effect.

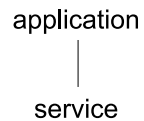
Chapter 1. Introduction to First Failure Data Capture

The First Failure Data Capture (FFDC) utilities can be employed by software for recording persistent records of failures and significant software incidents. FFDC provides a means for associating failures to one another, allowing software to link effects of a failure to their causes, and thereby facilitate the quick location of the root cause of a failure. Doing so allows problem investigators to identify the root cause sooner, without having to diagnose the symptoms or take possibly unneeded or useless corrective action.

Problem determination in distributed computing systems is a daunting task. Such systems usually have many interdependencies among hardware resources, firmware, and many layers of software. A component can encounter any number of failure conditions, and problems in one component often results in side effect problems experienced by other components. Sometimes these problems manifest themselves as the same symptom in the various components that the problem impacts, but more often a problem in one component of a distributed system causes a seemingly unrelated symptom to occur in a component that depends upon it.

Software application developers in distributed systems are familiar with the challenge of developing applications for these systems. The success or failure of the application directly depends upon the proper functioning of all of the components in the system. It is not uncommon for an application to depend upon a specific software service, application, or distributed system component, which in itself depends upon other services and components as well.

The application developer may perceive the programming environment to be a relatively simple one between the application and the service:



The failure situation reality is actually much more complex, as shown in the following figure.

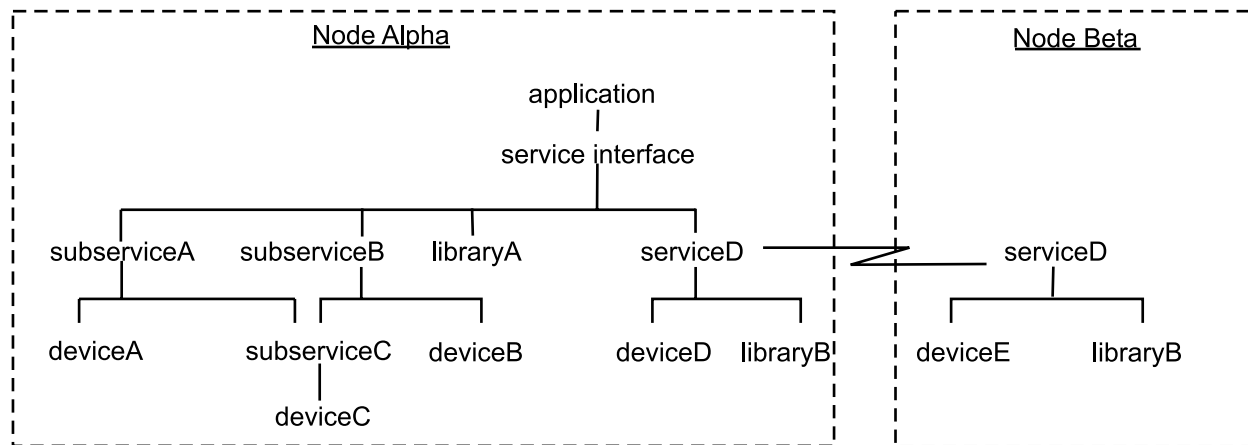


Figure 1. Example of Complex Failure Situation

Failures can occur at any level within the interdependencies of the distributed system. Failures in the lower levels of these interdependencies impact the ability of the components that depend on them to deliver their function, and often result in a failure being experienced by these dependent components. These new failure conditions in turn cause other failure conditions. The chain continues until the failure manifests itself as some symptom at the application level, whether that symptom be incorrect results, recovery activity, or complete failure.

When the application experiences the failure, unless it is a direct result of a usage error by the application, the cause of the failure may be difficult to determine from the symptoms noticed by the application.

Anything other than a usage failure is perceived as a failure in the service. But how does one identify the true cause of the failure condition when the service is composed of many interdependent components? By the time the failure manifests itself as a symptom in the application, the failure has been experienced by possibly many components that comprise the service. Each individual component has experienced a different effect of the same condition. Each component may have recorded information about the failure, but each component recorded its view of the failure. These views may not match up very well with the symptom seen by the application. For example, the failure to engage a device at the lowest levels of these interdependencies may result in any number of symptoms at the application level; failure to read from a database, failure to authenticate oneself as a user of the system, failure to connect to a remote system, or any number of other symptoms.

Without any mechanisms to assist the application developer, the only way to understand the root cause of a symptom is to first understand the interdependencies of the distributed system. Even with this understanding, problem determination becomes a nondeterministic guessing game. See the following figure that depicts a failed service for a complex system.

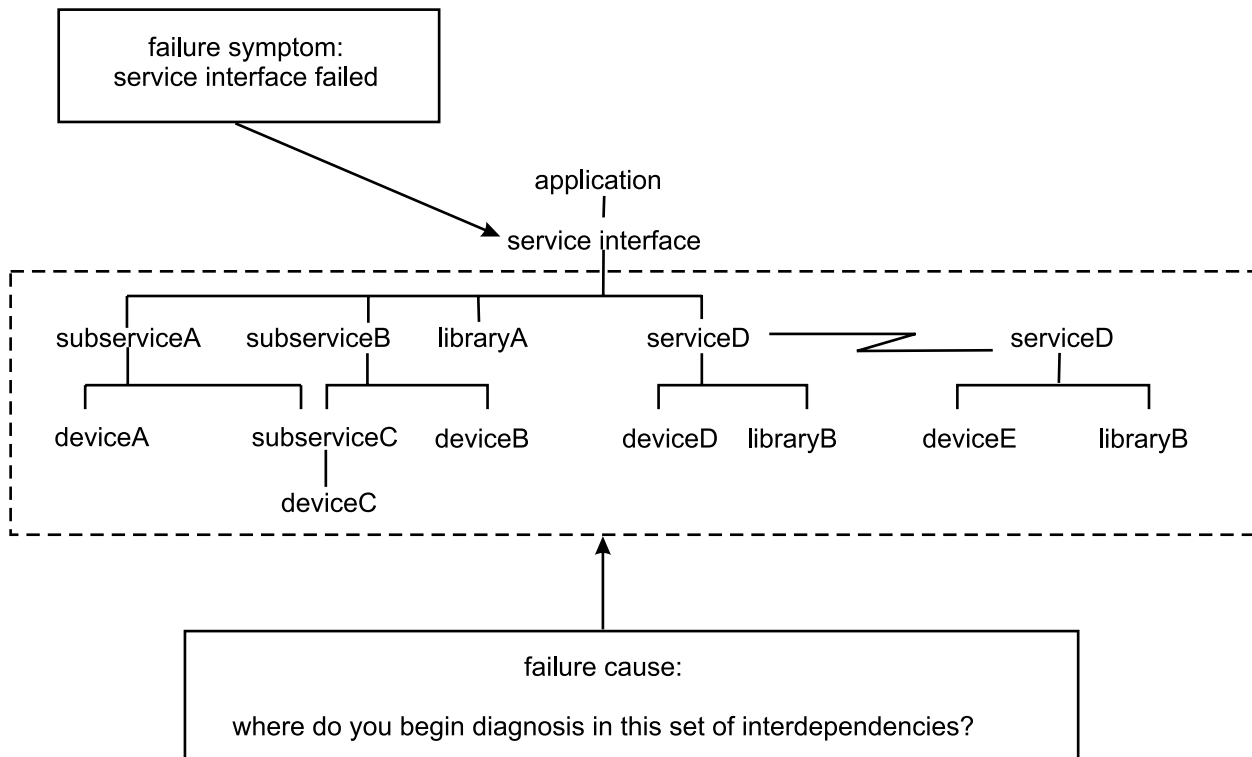


Figure 2. Diagnosing a Complex Failure Situation

How do you know whether to investigate subserviceA, subserviceB, libraryA, or serviceD to determine where the failure was caused, especially if a number of them have recorded failure records? If deviceB is known to be experiencing problems, can it be safely assumed that the application's failure is directly related to deviceB's problems? How does the problem investigator know if Node Beta might be involved in the failure, and whether or not to check that node for failure information? Judgements are made as to which component might be the most likely source of the failure, but these judgements are still guesses. Customer service representatives may spend hours or longer investigating a specific failure path and implement a repair for a problem discovered there, only to find that the repair has no effect on the end user's failure condition.

Another method often used is problem re-creation. Specialized traces are put in place to monitor the service's execution, in the hopes that the failure condition will occur again and leave a trail in the trace information. Unfortunately, activating a trace often degrades overall system performance, steals resources from other vital tasks, and often changes the conditions under which the service executes. Since such traces change the overall operational conditions of the service, one cannot guarantee that the service will experience the same conditions that caused the original failure condition.

Clearly, a better alternative is needed. The components of a distributed system clearly know their interdependencies. A mechanism should be available that permits these components to both record their failure conditions to persistent storage, and for these records to be somehow associated so that the causes and effects of failures can be understood. Problem resolution must evolve from the guessing game stage to a more sophisticated and reliable methodology.

First Failure Data Capture (FFDC) provides this capability. Building upon AIX failure recording facilities and providing a new facility, FFDC permits software to establish links between related failure conditions. Using this capability, a software developer can link effects to previously experienced failures, establishing a direct relationship between failures. When the failure condition is finally detected at the application level, the application can now have a direct reference to the start of a failure trail. The reference can be tracked through the various interdependencies of the service's components, leading the problem investigator directly to the root cause of the problem. Guessing is no longer required. Trial and error repairs are no longer necessary. FFDC leads you towards the root cause, even if that cause is on a different node in the system. You can simply repair the root cause and continue. The following illustration shows how FFDC is related to determining the root cause of a failure.

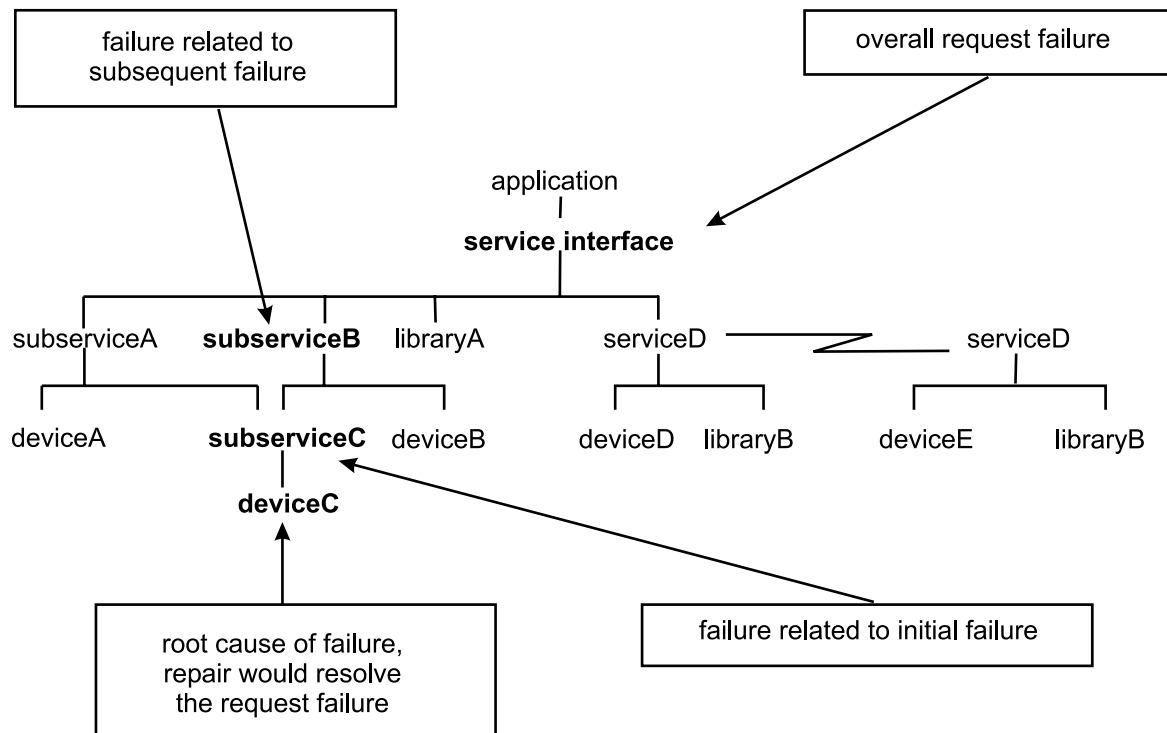


Figure 3. Determining Root Cause of a failure with FFDC

FFDC is provided to developers of distributed applications and to service developers. It is engineered to link related failures between nodes and between different software components. When all components support FFDC, a complete trail of related failures can be established from the root cause all the way to the application's failure symptom. Even if some components do not support FFDC, enough of a trail can be

constructed so that investigation paths can be easily eliminated from the problem determination effort. The following illustration depicts the use of FFDC in tracking failures.

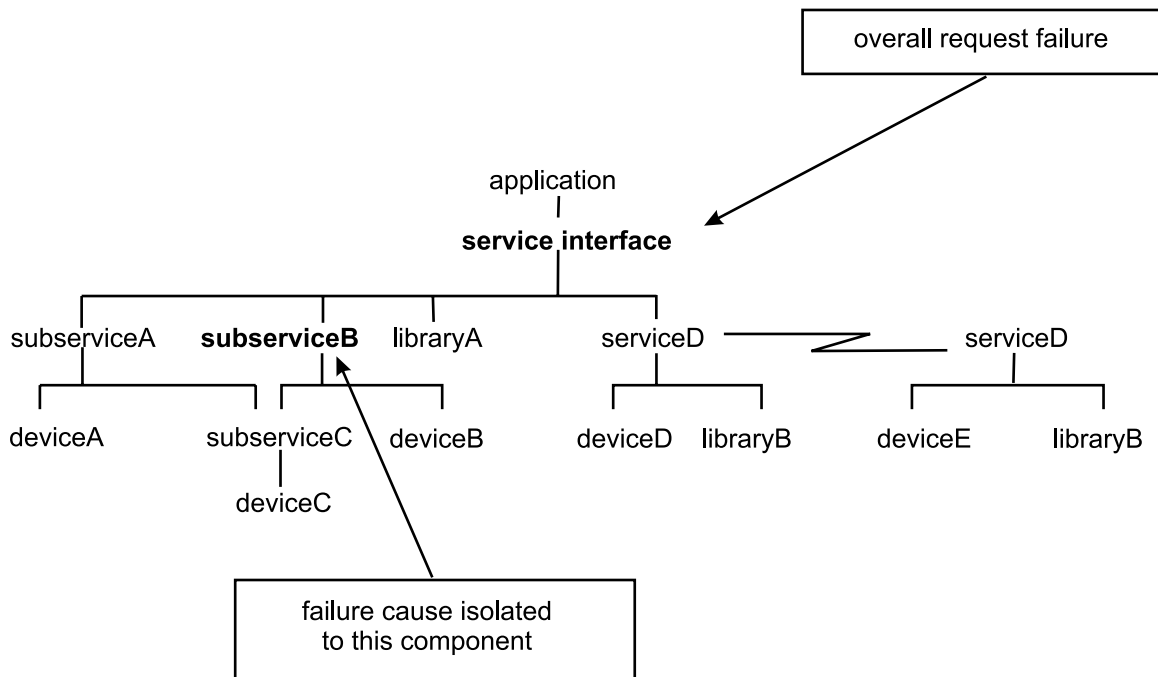


Figure 4. Tracking Failures with FFDC

FFDC utilities have been provided for these languages and environments:

- C language programs (also supports use by C++ programs)
- Script environments: Bourne, Korn, and C shell languages supported

FFDC Environment

The FFDC utilities provide two process execution environments and two levels of function:

- A basic level, the FFDC Environment, in which the FFDC client only makes use of the AIX Error Log and the BSD System Log, and
- An FFDC Error Stack Environment, which establishes an FFDC Error Stack for use by the software and allows the software to use the AIX Error Log and the BSD System Log.

An FFDC Error Stack is most useful when a series of processes or threads is executing as part of a common task, and these processes or threads are executing on the same system. These processes or threads must have a parent-child relationship between them. Information is recorded to the FFDC Error Stack to assist the application developer in finding problems in the application. Anything that needs to be brought to the system administrator's attention should be recorded to the AIX Error Log as well; the system administrator won't necessarily be monitoring the contents of the various FFDC Error Stacks.

FFDC Error Stack Environments can be created or inherited, depending on whether the software always wants an FFDC Error Stack to be in use. If an FFDC Error Stack is always desired, the software creates the FFDC Error Stack Environment. If the software is supporting the FFDC Error Stack only when an ancestor or client is using one, then the software inherits the FFDC Error Stack Environment. Without establishing an FFDC Error Stack Environment, whether it be by creating one or inheriting one, attempts to record to the FFDC Error Stack will not function.

The decision of whether or not to establish an FFDC Error Stack Environment is left to writers of applications and daemons. Library designers should not consider establishing an FFDC Error Stack Environment; they should leave that responsibility to the client. Libraries need to test for the presence of an FFDC Error Stack Environment before attempting to record information to it. See “fcteststk/fc_test_stack” on page 42.

The FFDC Environment is established through the **fcinit** command or the **fc_init** library routine. Each has options that permit the software to establish the specific FFDC Environment desired:

Table 1. Methods for Establishing FFDC Environments

FFDC Environment Requested	bsh/ksh Command	csh Command	Library Routine
Basic Environment - permits recording to the AIX Error Log and the BSD System Log	. fcinit.sh -1	source fcinit.csh -1	fc_init(FC_LOG)
FFDC Error Stack Environment - always present (created) - establishes an FFDC Error Stack for use, permits access to AIX Error Log and BSD System Log	. fcinit.sh -sc	source fcinit.csh -sc	fc_init(FC_STACK_CREAT)
FFDC Error Stack Environment - supported only if previously established by ancestor process (inherited)	. fcinit.sh -si	source fcinit.csh -si	fc_init(FC_STACK_INHERIT)

Notice that when used in shell script environments, the commands must be executed as part of the script's environment, not as a separate subcommand. Some shell script languages call this “sourcing” a command. In Bourne and Korn shells, a command is “sourced” by placing a period and a space before the command (.), as shown in the table above. C shells use the source built-in command to achieve the same result. These commands set environmental values, and therefore must be “sourced” from the calling script. If they were not “sourced”, the commands would set process environment variables within their own separate process, which would be lost as soon as the command terminated.

Creating a Basic FFDC Environment

A basic FFDC Environment can only be created by the process. If a process does not create a basic FFDC Environment, the FFDC utilities can still be used to record information to the AIX Error Log and the BSD System Log. The software loses the capability of linking together related failures if a basic FFDC Environment is not established.

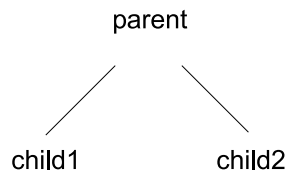
Basic FFDC Environments are always created whenever an FFDC Error Stack Environment is established. No special instruction is required to create it.

Unlike the FFDC Error Stack Environment, software cannot test for the presence or absence of a basic FFDC Environment. If the software is designed to record information to the AIX Error Log via the FFDC interfaces, the software must take responsibility for establishing the basic FFDC Environment itself. Application programs will need to establish the basic FFDC Environment themselves. Library software will have to rely on the client establishing the basic FFDC Environment, and should document this assumption in their user documentation.

Creating an FFDC Error Stack Environment

Software creates an FFDC Error Stack Environment if the software always wishes an FFDC Error Stack to be set aside for the software's use. Each time the software is executed, an FFDC Error Stack will be reserved for use.

Consider the “Parent/Child Source-Code Examples” on page 111. In these examples, the parent, child1, and child2 applications were created first, and the grandp application added at a later time. This discussion will consider the original design of the parent, child1, and child2 applications.



The parent application wishes for an FFDC Error Stack to be used whenever it is executed, so that failures in any of its child processes can be traced, and so that the effects of these failures to the parent application itself can be understood. Because an FFDC Error Stack is always desired by parent, parent creates an FFDC Error Stack Environment. See the mainline code of “The parent.sh Script” on page 111 for the call made by the script to set up this environment:

```
. fcinit.sh -sc
```

This sets up an FFDC Error Stack Environment for the parent process. Special process environment values are established for this process, and are passed along to any child processes created by parent.

What if an FFDC Error Stack Environment had been previously established by one of parent’s ancestors? This possibility will be addressed in the section: “Creating an FFDC Error Stack Environment Instead of Inheriting” on page 7.

Inheriting an FFDC Error Stack Environment

Now consider the child1 application in this example. When child1 is created, the environment values established for its parent application are provided to child1 through normal Unix semantics. The environmental values used to establish the FFDC Error Stack Environment exist, but they contain the same values used by parent. In other words, the FFDC Error Stack Environment still thinks it is set up for parent, not for child1. child1 does not automatically receive an FFDC Error Stack Environment simply because parent established one.

When child1’s execution begins, the process has the capability to inherit the FFDC Error Stack Environment. However, child1 must explicitly execute an FFDC instruction to inherit the environment. See the main() routine in the section “The child1.c Program” on page 119 for the instruction used by child1 to inherit the FFDC Error Stack Environment:

```
rc = fc_init(FC_STACK_INHERIT);
```

This FFDC call checks if an FFDC Error Stack Environment is available to be inherited. The utilities discover that an FFDC Error Stack Environment is available, but not tailored to child1 yet. **fc_init** tailors the environmental settings so that child1 can also make use of the FFDC Error Stack Environment, granting child1 access to the previously reserved FFDC Error Stack. child1 successfully inherits an FFDC Error Stack Environment.

Suppose that child1 had not inherited the FFDC Error Stack Environment. What would happen if child1 used the FFDC utilities to record information to the FFDC Error Stack? The FFDC utilities would find that child1 had not established an FFDC Error Stack Environment for itself, and would not record information to the FFDC Error Stack Environment. Even though its parent process established an FFDC Error Stack Environment, *child1* had not established one for itself.

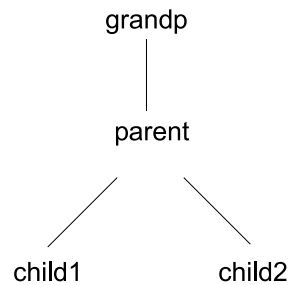
Now suppose that no FFDC Error Stack had been created by the parent process of *child1*. Such a case would exist if *child1* were invoked directly from the command line:



In this case, no FFDC Error Stack would be in use by *child1*. The **fc_init** call would inform *child1* that an FFDC Error Stack Environment did not exist, so it was not inherited. *child1* could then "remember" that an FFDC Error Stack Environment did not exist and not bother to collect or record failure information to the FFDC Error Stack. In the example used in the appendix, *child1* does not retain this knowledge but instead checks whether or not the FFDC Error Stack Environment exists each time it wants to record information to it. The choice of which approach to use is left to the software designer.

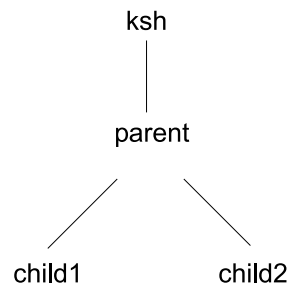
Creating an FFDC Error Stack Environment Instead of Inheriting

Recall that in the example, the parent application is creating an FFDC Error Stack Environment for itself and its descendants. At a later time, the grandp application is written as a parent application to parent itself. Like *parent*, grandp wants to make use of an FFDC Error Stack every time it is executed, to make it easier to trace failures in its descendants and understand their impacts.



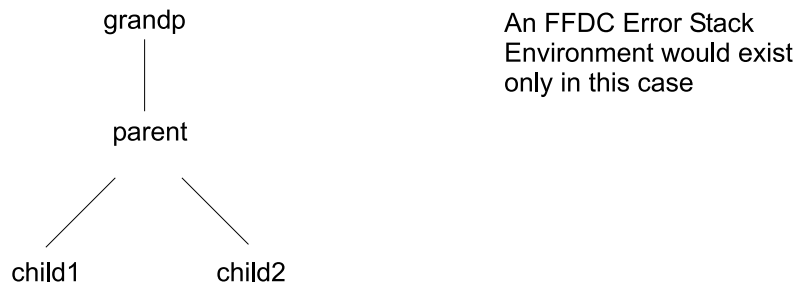
Recall that *parent* is going to *create* an FFDC Error Stack Environment every time it begins execution. However, grandp has already created an FFDC Error Stack Environment for use by its descendants. What happens in this case?

In this case, the *parent* application should *not* be changed to *inherit* the FFDC Error Stack Environment. Why? When *parent* is executed by some other process other than grandp, the *parent* application developer still wants an FFDC Error Stack reserved for the process and its children. If *parent* were changed to *inherit* an FFDC Error Stack Environment instead of always *creating* it, an FFDC Error Stack would not be used in the case where *parent* was executed by a process other than grandp:



No FFDC Error Stack Environment would exist in this case

The only time an FFDC Error Stack Environment would exist for an execution of parent is when it was invoked directly from grandp:



To ensure that an FFDC Error Stack Environment always exists whenever parent is executed, parent continues to *create* an FFDC Error Stack Environment whenever it is executed. But what does this do as far as grandp is concerned?

In these cases, the FFDC utilities convert the *create* request made by the process into an *inherit* request, if there is an FFDC Error Stack Environment capable of being inherited. When the FFDC utilities are used to *create* an FFDC Error Stack Environment, they perform the same check that they do when a process is attempting to *inherit* an FFDC Error Stack Environment. If an FFDC Error Stack Environment is capable of being inherited, the utilities *inherit* the environment instead of *creating* it. If no FFDC Error Stack Environment is capable of being inherited, the utilities *create* the FFDC Error Stack Environment.

This specific behavior of the FFDC Error Stack Environment is intentional. In cases where no FFDC Error Stack Environment exists, the *create* request established one. In cases where the FFDC Error Stack Environment can be inherited, the process "plugs into" the existing FFDC Error Stack Environment. This permits failures in parent and any of its descendents to be tracked as part of an FFDC Error Stack any time it is used by a process that creates an FFDC Error Stack for its own use, whether it be grandp or any other process. If parent did not *inherit* the existing environment in these cases, parent's failures and those of its descendents would be recorded to a separate stack, and the calling process would have no record of the failures or their impacts, limiting the usefulness of the FFDC Error Stack.

The FFDC utilities inform the application if the *create* request became an *inherit* request. Notice that "The parent.sh Script" on page 111 is written to accommodate the case when its creation request winds up being an inherit request.

When the FFDC Error Stack Is Most Useful

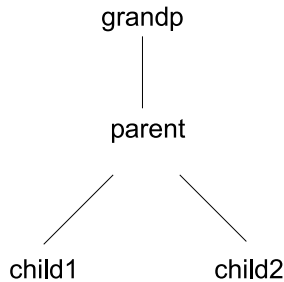
An FFDC Error Stack is most useful when a series of processes or threads is executing as part of a common task, *and* these processes or threads are executing on the same system. These processes or threads *must* have a parent-child relationship between them. Information is recorded to the FFDC Error Stack to assist the application developer in finding problems in the application. Anything that needs to be brought to the system administrator's attention should be recorded to the AIX Error Log as well; the system administrator won't necessarily be monitoring the contents of the various FFDC Error Stacks.

All processes or threads would write failure information to the FFDC Error Stack. If the overall task fails, the FFDC Error Stack should have all the failure information about the failure, including the root cause of the failure, plus all the effects that the failure had on the other processes or threads. Instead of beginning the problem determination effort with the problem symptom, the FFDC Error Stack can help the problem investigator find the root cause quickly.

An FFDC Error Stack can also be useful when a single application is making use of multiple libraries in accomplishing its task. If a library routine encountered a failure or some significant condition, it would record information about it to the FFDC Error Stack. If the overall application fails, the application writer can consult the FFDC Error Stack to see what incidents occurred, and what the effects of those incidents were at various levels in the application.

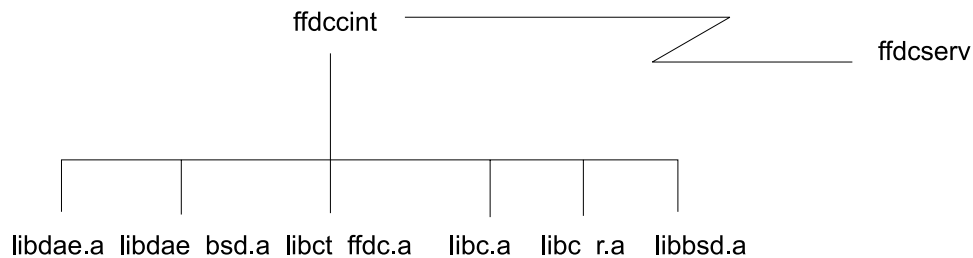
Note: The FFDC Error Stack is not to be considered a replacement for a tracing facility. If an application does not fall into one of the two previous classes, use of an FFDC Error Stack is not practical. The FFDC Error Stack does not span across nodes of a distributed system, so using an FFDC Error Stack with a **dsh** or **rsh** application may not be practical.

An example of an application that can make use of use of an FFDC Errpr Stack can be found in the appendix, "Parent/Child Source-Code Examples" on page 111.. In this case, a series of processes are executing on the same node to accomplish a particular task:



These processes are all started from a common ancestor, grandp. Therefore, they all share a common execution environment, one that was established by grandp and inherited through normal Unix conventions by parent, child1, and child2. An FFDC Error Stack is suitable because this parent-child relationship exists, the processes are performing parts of a single common task, and all processes are executing on the same system.

Another source code example is given in "Daemon Source-Code Example" on page 148.. In this case, a client process makes requests of a server process that may execute on a remote node:

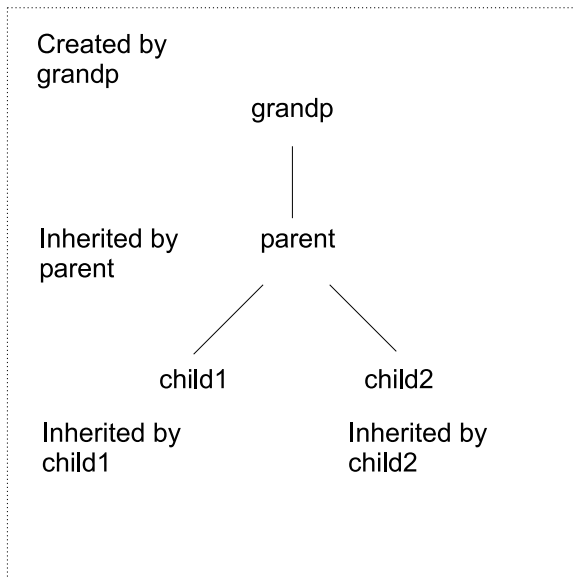


The ffdcclnt process and the ffdcsvr process can be executing on separate nodes, so attempting to use an FFDC Error Stack to capture failure information for both processes at once is not possible. Each process could establish its own FFDC Error Stack to capture information about failures encountered by any child processes or threads created by them. In this case, neither create children or separate threads. Finally, the processes could establish an FFDC Error Stack to trace failures that are detected by the libraries they use, to trace the failures from their symptoms to their causes. For the sake of this example, assume that most of the libraries used by the ffdcclnt application do not exploit FFDC. For these reasons, the ffdcclnt application decided not to make use of an FFDC Error Stack. Instead, failures and other significant incidents are recorded to the AIX Error Log, and the application uses the FFDC Failure Identifiers to establish links between failures and their effects.

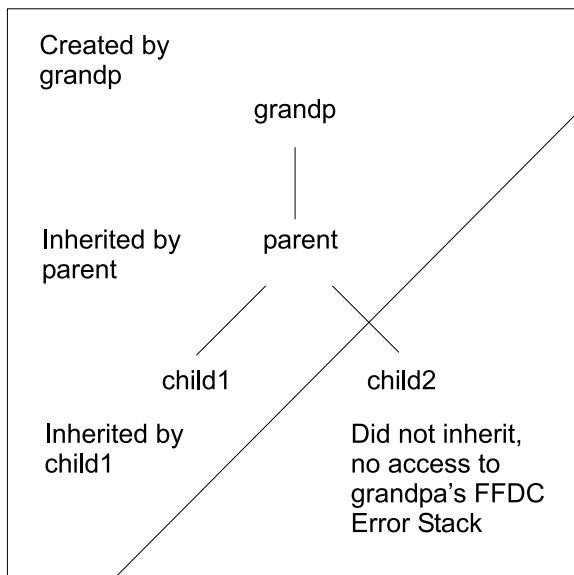
What Is The Scope of an FFDC Error Stack?

An FFDC Error Stack can contain failure information about the process that *creates* the FFDC Error Stack Environment, and any processes or threads that *inherit* the FFDC Error Stack. Looking at the example in Appendix A, the grandp process creates the FFDC Error Stack Environment, and all of its descendents

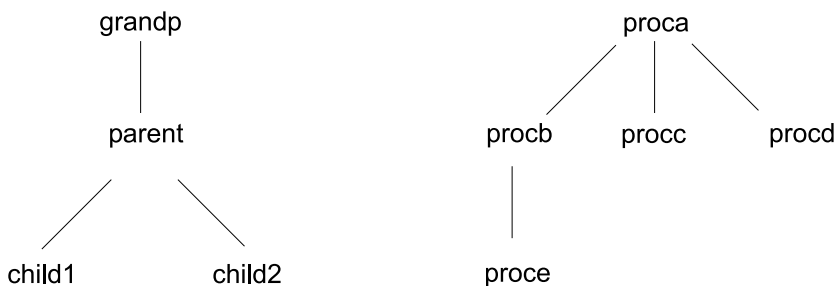
either inherit the environment or "plug into" it.



Suppose that child2 did not inherit the FFDC Error Stack Environment created by grandp? The child2 process does not have access to the FFDC Error Stack, and is outside of its scope. This is discussed in greater detail in the section: "Inheriting an FFDC Error Stack Environment" on page 6.

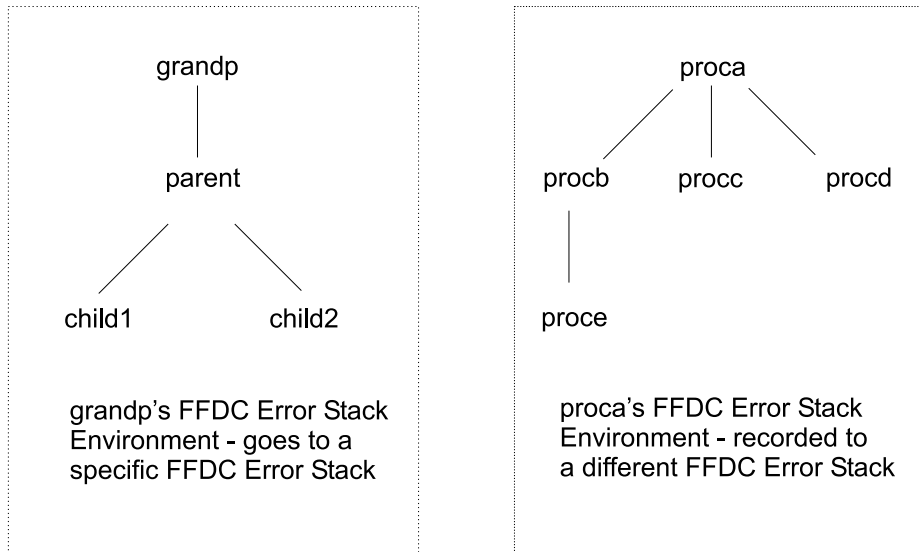


Now suppose that multiple processes that create FFDC Error Stack Environments execute at the same time on a specific system:



For this example, grandp and proca both create FFDC Error Stacks to track failures and their impacts. Do both sets of processes contribute to the same FFDC Error Stack? No.

There is one FFDC Error Stack for each successfully created FFDC Error Stack Environment. Since the grandp and proca processes are not related, each creates its own individual FFDC Error Stack Environment. The FFDC Error Stack is not the same as the AIX Error Log in this case, where only one AIX Error Log exists on a system.



Should the Software Support Possible Use of an FFDC Error Stack?

Perhaps the software itself doesn't find an FFDC Error Stack to be useful in all situations. For example, a library provider might decide that an FFDC Error Stack is not useful for the library because the library does not create any subprocesses or subthreads and does not invoke other libraries. An application might decide that an FFDC Error Stack is not practical if the application performs only one task and is more commonly issued by itself than as part of a series of tasks. In these cases, the software does not create an FFDC Error Stack for its own use.

But such software might be used by another application that wishes to use an FFDC Error Stack Environment. The library may be used by an application that uses other libraries and subprocesses to accomplish a task and wants an FFDC Error Stack to assist in problem determination. Because the stand-alone application may be invoked as part of a shell script and might impact the overall success or failure of the script, the script would want to use an FFDC Error Stack. If these software components do not support use of the FFDC Error Stack, it is not possible for them to record information on any failures or incidents they detect. The purpose of the FFDC Error Stack is then defeated, and when a problem occurs in the client software, problem determination for the client is at a disadvantage.

FFDC Error Stack Support For Libraries

Libraries run in the execution environment of their client. It is the responsibility of the client to decide whether an FFDC Error Stack is useful. The library issues FFDC calls when an incident occurs to determine if an FFDC Error Stack Environment exists, and if so, issues the FFDC calls to record information to the FFDC Error Stack. If the FFDC Error Stack Environment does not exist, the library proceeds normally.

FFDC Error Stack Support For Applications

Applications are a different case. For such cases, the FFDC utilities allow an FFDC Error Stack Environment to be *inherited* as well as *created*. The application in the current example would not *create* an FFDC Error Stack Environment to use an FFDC Error Stack, because that would cause an FFDC Error Stack to be started for every instance of the application. However, the application can *inherit* an FFDC Error Stack Environment. *Inheritance* works like this:

- If an FFDC Error Stack Environment was established by an ancestor of this process, then *inheritance* allows the process to make use of the existing environment. The process essentially "plugs in" to the existing environment and uses the existing FFDC Error Stack.
- If an FFDC Error Stack Environment were not established by an ancestor of this process, then there is no environment to *inherit*. If the process attempts to inherit the environment, the FFDC utilities inform it that no environment is available. The process continues as it would normally. Any calls to FFDC utilities do not result in any data being recorded to an FFDC Error Stack.

To *inherit* the environment, the process must ask to inherit it. The ancestor process can make the FFDC Environment available, but this process must "sign up" to use the environment. Otherwise, any attempts to make use of the FFDC Environment are no-ops.

The FFDC Error Stack Environment is established by using special variables in the process execution environment. This makes *inheritance* a simple concept to implement—the environmental variables are passed down from ancestor to child. A descendant process *inherits* an FFDC Error Stack Environment by issuing an FFDC call to tailor these environment values to the process. This is the reason why an ancestral or parent-child relationship is required to use the FFDC Error Stack.

Recording Failure Information

Software is still responsible for employing defensive programming techniques and detecting failures and other noteworthy incidents on its own. The FFDC utilities are not capable of deciding what warrants a "failure" for a particular piece of software. One's failure condition is another's expected result. Therefore, the software becomes ultimately responsible for deciding what constitutes a failure, and taking steps to detect them and prevent them from cascading into other failures.

Once a failure or significant incident is detected, the software decides whether or not to record information about it.

Table 2. Recording Failure Information

Condition Detected	Basic FFDC Environment Established	FFDC Error Stack Environment Established
Condition that should be brought to the system administrator's attention - application failures and major status changes	Record to AIX Error Log using a template designed specifically for the failure	Record to AIX Error Log using a template designed specifically for the failure, and record description to FFDC Error Stack to aid in application problem resolution
Condition that should be brought to the application developer's attention	Make a record to the AIX Error Log <i>only if</i> the application support services should be notified of the condition	Record to FFDC Error Stack
Checkpoint used in application debugging	Record only to trace log, do not record to AIX Error Log	Record to FFDC Error Stack and to trace log

Recording in a Basic FFDC Environment

In a basic FFDC Environment, the software can make recordings to the AIX Error Log using the `fc_log_error` library routine or the `fclogerr` command. Neither command requires that a basic FFDC Environment be established at the time, but having one is recommended. Without a basic FFDC Environment, the FFDC utilities will not be able to generate an *FFDC Failure Identifier* for the recording. Consult the section *Using the AIX Error Log Via the FFDC Utilities* for help on recording to this facility.

When the FFDC utilities are used to record to the AIX Error Log *and* a basic FFDC Environment exists, the FFDC utilities generate a token for the record. This token is called the *FFDC Failure Identifier*. With this identifier, later problem determination efforts can locate the *exact* record made to report this condition.

After calling the **fclogerr** or the **fc_log_error** routines, the software is provided with an *FFDC Failure Identifier* for its record. The software is recommended to return this identifier to its own client as part of its failure information. The reason is simple: the software's failure may lead to a related failure in the software's client. If the client will also fail, the client should be able to establish a persistent link between its failure and the cause of the failure. By returning the *FFDC Failure Identifier* to the client, the software permits the client to make this link. The link assists later problem determination efforts in bypassing problem symptoms and isolating the root cause of the problem in a short amount of time.

Recording an Initial (Unrelated) Failure

Consider "Daemon Source-Code Example" on page 148. In this example, the **ffdclnt** daemon communicates to its client via a file. Through this file, request results and error information is provided. **ffdclnt** can encounter a number of failure conditions that can result in records being made to the AIX Error Log. Whenever **ffdclnt** makes such recordings, it retains the *FFDC Failure Identifier* from the **fc_log_error** routine, and passes this token back to the client through the shared file by using the **WRITE_CLIENT_RESULTS** macro. The client has to know that this information is being returned in error cases, so it can understand what information is being returned and how it can be used.

Recording an Associated (Related) Failure

Examine the **ffdclnt** example source code again. In the **recv_server_reply()** function, it is possible that the **ffdcserv** server has indicated that it encountered a failure which prohibited it from fulfilling the request made by **ffdclnt**. As a direct result of **ffdcserv**'s failure, **ffdclnt** will also fail. The *FFDC Failure Identifier* from **ffdcserv** is extracted from the response information and returned to the **main()** routine. The **main()** routine then records **ffdclnt**'s failure to its own AIX Error Log. In addition to the failure information for **ffdclnt**, **main()** also provides the *FFDC Failure Identifier* from **ffdcserv** as an *associated failure identifier* to the **fc_log_error** routine. This permits the *FFDC* utilities to establish a persistent link between the root cause of the failure—**ffdcserv**'s failure—and the effect of that failure in **ffdclnt**.

Later problem determination efforts will detect that **ffdclnt**'s failure was caused externally by **ffdcserv**, and will obtain the token for that failure from **ffdclnt**'s failure report. Problem determination efforts move directly to **ffdcserv**, instead of wasting time debugging non-existent problems in **ffdclnt**.

Recording in an FFDC Error Stack Environment

Unlike the basic *FFDC* Environment, an *FFDC Error Stack Environment* must exist before software can record information to an *FFDC Error Stack*. The *FFDC* utilities are designed so that they return control back to their clients if they are invoked when an *FFDC Error Stack Environment* does not exist, so no ill effect occur if software makes use of the utilities in these cases.

FFDC provides commands and subroutines that detect whether an *FFDC Error Stack Environment* exists. These interfaces are provided for applications that cannot know or cannot retain the knowledge that an *FFDC Error Stack Environment* exists. Libraries are examples of such applications. Software can make use of the **fc_test_stack** library routine or the **fc_teststk** command to determine if an *FFDC Error Stack Environment* was established. These tests can be used before collecting failure information for the report.

The **fcpushstk** command and the **fc_push_stack** routine permit *FFDC* clients to record information to the *FFDC Error Stack*. Both interfaces provide the client with an *FFDC Failure Identifier* for its record. The software is recommended to return this identifier to its own client as part of its failure information. The reason is simple: the software's failure may lead to a related failure in the software's client. If the client will also fail, the client should be able to establish a persistent link between its failure and the cause of the failure. By returning the *FFDC Failure Identifier* to the client, the software permits the client to make this link. The link assists later problem determination efforts in bypassing problem symptoms and isolating the root cause of the problem in a short amount of time.

Recording an Initial (Unrelated) Failure

In "The **child2.sh** Script" on page 125, the mainline code verifies that the software was executed in the proper environment. If not, **child2** uses **fc_teststk** to see if an *FFDC Error Stack Environment* exists for the script, and makes a recording to the *FFDC Error Stack* if such an environment was established. A

corresponding record is not made to the AIX Error Log because the application developer does not believe it to be a condition worth reporting to the system administrator; it is only of interest to the application developer. The FFDC Failure Identifier from the `fcpushstk` command is obtained to be returned to the script's client.

Recording an Associated (Related) Failure

To see how an initial failure can lead to an associated failure in a client, return to "The parent.sh Script" on page 111. Recall that parent started the child2 process to accomplish a segment of an overall task. Because of the failure in child2 to accomplish its task, parent will also fail to perform its task. This means that child2's failure is *associated* to parent's failure, or that parent's failure is *related* to child2's failure. parent extracts child2's FFDC Failure Identifier from the failure information it provided, and prepares to record an entry for its own failure. Along with parent's failure information, parent also provides child2's FFDC Failure Identifier to the `fcpushstk` command as an *associated failure*. This establishes a persistent link between the root cause in child2 and the effect in parent.

Reporting Results

Software must inform its client as to whether the software succeeded or failed. If the software fails, it must inform its clients of the failure by a means that the client can test. The client must not be falsely led to believe that the software succeeded when it actually failed. The reverse is also true.

When software fails and makes a recording for the failure using the FFDC utilities, the FFDC Failure Identifier generated by the FFDC utilities should be returned to the client of the software as part of the failure information if possible: certain interfaces, such as those defined by POSIX, ANSI-C, or X/Open standards, may not permit additional mechanisms for passing additional error data. The FFDC Failure Identifier permits the client of the software to establish a persistent link between the failure of the client—the "effect"—and the "root cause" failure of the software. See the section "Informing the Client of the Failure Information" for assistance on returning information to the client.

Informing the Client of the Failure Information

Before a failure is reported, the failure must be understood by the software. Reporting a failure without understanding the failure leads to failure reports with little or no information that is helpful either to the client, the system administrator, or the problem analyzer. Without this understanding, the software can only report the symptom that it encountered. The report will not assist anyone using the report to understand what caused the failure to occur and what can be done to repair or bypass the failure condition.

Informing the Client At Execution Time

The most common mechanisms used by AIX-based software to convey the results of its execution to its client were mentioned earlier and are discussed below:

- Exit status codes from commands or child processes
- Return codes from library routines
- **errno** settings from library routines
- Pass-by-reference data areas provided for storing error information, passed as arguments to library routines
- Result information provided to a client through IPC mechanisms such as files, pipes, semaphores, shared memory, and message queues
- Result information received through asynchronous mechanisms such as signals and sockets
- Text messages directed to a standard error or standard output device

Software can use several of these mechanisms at the same time. For instance, a library routine can set a return code value, an **errno** value, and provide additional error information in a pass-by-reference field all at the same time. Applications can generate both an exit status code and a text message to indicate the results of the execution.

When the execution of software encounters a failure, the software must use mechanisms such as those listed above to alert the client to the failure condition. The software uses whatever mechanisms make sense for its execution context (for example, a library routine would not use an exit status code), and can combine the use of more than one mechanism to convey the failure notification. These mechanisms should be chosen so that all of the following information is conveyed in the notification:

- Overall success or failure of the software function
- Description of the failure condition
- Most probable cause or causes of the failure condition
- Actions to take in response to the failure condition, if possible

The combination of mechanisms used by the software should provide a unique set of information for any unique set of the above items. A different combination of values should be used for one specific failure than for another: no two failure conditions should use the same failure notification information, if it can be helped.

Consider the case of a hypothetical library routine. This library routine expects to be used by an application with a controlling terminal. The following possible failure conditions and their causes have been determined:

- Reception of invalid parameters, which is an external failure.
- Invoked by a client with insufficient privilege, which is an external failure.
- Failure to attach to an existing shared memory segment. This can indicate a resource failure in either the file used to generate an IPC key, a resource failure if the shared memory does not exist, or an assumed resource failure if the shared memory permissions are not the same as those this software assumes.
- Failure to read data from the shared memory segment. This can indicate an external failure in the external application using the shared memory along with this application if the data is invalid, and external failure if the shared memory segment is removed through the `ipcrm` command, a resource failure with the shared memory segment if the data is corrupted, or an internal failure if the shared memory permissions do not allow the application to read data from this segment (this condition should have been caught earlier).
- Failure to write data to the shared memory segment. This can indicate a resource failure with the shared memory segment if the segment is suddenly removed by the `ipcrm` command, an internal failure if the routine attempts to write data to an invalid offset in the shared memory segment, or an internal failure if the shared memory permissions do not allow the application to write data to this segment (this condition should have been caught earlier).
- Failure to dynamically allocate memory, which is assumed to be a resource failure, but may also be an internal failure to the application if memory is not being released after it is used.
- Segmentation violation in accessing a specific offset of dynamically allocated memory or shared memory, which is an internal failure.

This library routine has been designed to detect these conditions, with exception of the segmentation violation; in that case, the `SIGSEGV` signal interrupts the library's application and generates a core dump to terminate the application. When the library routine encounters the failure conditions listed above, it attempts to understand the failure from the notifications it receives, from the context in which the failure occurs, and the failure information it receives from the software it uses.

Once the library routine understands the failure to the best of its ability, the library routine will set a return code of -1 and an **errno** value to identify the failure:

- When invalid parameters are detected, **errno** is set to `EINVAL`. The user documentation for the interface indicates that this error value is provided when invalid parameters are specified and recommends that the application programmer correct the parameters provided to the interface.

- When the interface is invoked by a client with insufficient privilege, **errno** is set to EPERM. The documentation describes when this is set as an error value and recommends that the client obtain proper privilege or that the client's privileges be verified with the security subsystem.
- A number of **errno** settings can come from the failure to attach an existing shared memory segment. A value of ENOENT indicates that the file used to generate the IPC key is missing. A value of ENODEV indicates that the shared memory did not exist when the library attempted to attach it to the application. A value of EACCES indicates that the shared memory's permissions are not the same as the settings used by the software. The documentation describes the conditions that cause each error value and gives recommendations of how to react to each condition.

The combination of the return code and the errno value have been selected by the library designer to uniquely identify the failure condition and its possible causes. The library routine's user documentation will describe these failure indications, its impacts to the application using the library routine, and what the application can do in response to this failure. Had the combination of errno value and the failure return code not been enough to uniquely identify the failure, the library routine would need to provide another item of information to uniquely identify the failure, such as providing additional failure information in a pass-by-reference parameter received from the caller.

Consider another case, using a command this time. This command can be executed from the command line, but may also be invoked by a script. The command therefore expects to have access to standard input and output devices. For sake of this example, the command performs a relatively simple function of formatting and displaying the contents of a named file, and requires no special client privileges. The following failure conditions and their causes have been identified:

- Invalid options provided on the command line, which is an external failure.
- Failure to locate the named file, which can either an external failure if the wrong name was provided to the command, or it can be a resource failure if the file is expected to exist on the system at all times.
- Failure to open the named file for input, which is a resource failure concerned with the permissions on the file itself. However, this can also be an external failure if the software's client provided an incorrect file name on the command line.
- Failure to allocate memory for a buffer to store a segment of the data read from the named file, which is assumed to be a resource failure in the virtual memory since the command has a limited life span and allocates memory only once.
- Failure to read data from the file. This can indicate a resource failure in the file itself if the file has no data stored in it or if the data is corrupted. It can also be an external failure if the command's client specified the wrong file on the command line.
- Failure to write the reformatted data to standard output. This can indicate a resource failure when standard output is being redirected to a file, and the file becomes too large or the filesystem storing the file runs out of space.

Once again, the software must choose a method of indicating the failure back to the software's client so that the client will understand whether or not a failure occurred, what the failure was, what could have caused the failure, and what can be done about the failure condition. Because this command expects to be invoked either from the command line or from a script, the command chooses to convey this information through the use of NLS compatible error messages to the standard error device and through uniquely chosen exit status codes. The messages will assist command line users of the command by providing a complete description of the error and the suggested actions to take in response to the failure. The exit status codes can be used by scripts to verify the successful completion of the command:

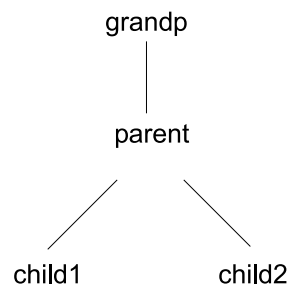
- An exit status of 0 is used when the command successfully completes.
- An exit status of 100 is used when invalid options are specified to the command. This exit status is documented in the command's user guide, as well as the recommendation to correct the usage of the command and to try the command again.

- An exit status of 101 is used when the named file cannot be located. The command's documentation describes this code. The documentation also recommends that the file name should be checked to make sure it is correct, and that the file should be checked to make sure it exists.
- An exit status of 102 is used when the named file cannot be opened for input. This exit status code is also documented. The documentation recommends that the client verify that the proper file name was provided, that the file's permissions be checked to make sure that any process can read the contents, and that the permissions be changed if they do not permit this access.
- An exit status of 103 is used when the command cannot allocate memory for its internal buffer. The documentation for the command contains the description of this exit code value, and recommends that processes hoarding system memory be identified and removed from the system.
- An exit status of 104 is used when the named file is found to have no data stored in it. The documentation recommends that the client verify that the proper file name was provided, and that the file be checked for problems if the correct file name was provided.
- An exit status of 105 is used when corrupt data is detected in the named file. The documentation recommends that the client verify that the proper file name was provided, and that the file be checked for problems if the correct file name was provided.
- An exit status of 106 is used when the command cannot write its output because the client's file size limit has been met or exceeded. The documentation recommends that the client either extend its file size limit, redirect the output to a paging device, or not redirect the output of the command.
- An exit status of 107 is used when the command cannot write its output because the file system containing the output file has run out of space. The documentation recommends that the file system be examined for unused or out-of-date files that can be removed, that the file system's size be extended, that the command's output be redirected to a file in a different file system or a paging device instead of a file, or that the command's output not be redirected.

The command has taken steps so that its clients, whether they be human or software, can detect whether the command succeeds or fails. The mechanisms used by the command permit either client to determine what failure condition occurred, what might have caused the failure condition, and what actions can be taken in response to the failure condition.

Returning an FFDC Failure Identifier in a Message

In specific cases, the software may only be able to return the FFDC Failure Identifier through a message. The source code examples in "Parent/Child Source-Code Examples" on page 111 show several examples of this:



In this example, the child1 and child2 programs have no private communication mechanism between themselves and their client, parent. The FFDC Failure Identifier is too lengthy to be contained in an exit status code, so the exit status can be used only to indicate success or failure to parent. child1 and child2 have little choice but to either establish a private communication mechanism between themselves and parent, or to return the FFDC Failure Identifier in a message.

To keep the amount of code to a minimum, both child1 and child2 choose to return this information in a message. The FFDC routine **fc_display_fid** and command **fcdispid** are used to write this information to the standard error device. However, this is not enough for parent to obtain this information. Both child1

and child2 have to document that the failure information is returned via a message to standard error, and parent has to be designed to capture this information for possible use later in failure recording.

Now consider the case of `ffdccInt` in “Daemon Source-Code Example” on page 148. In this case, it is not practical for `ffdccInt` to generate a message to report the FFDC Failure Identifier to its client; there is no terminal available to `ffdccInt`. `ffdccInt` is therefore forced to develop a private communication mechanism between itself and its client application. The shared file used by `ffdccInt` is developed for this purpose. Instead of using the `fc_display_fid` or `fcdispfid` interfaces, `ffdccInt` records the FFDC Failure Identifier directly to the shared file. The client is instructed to expect this information in failure cases, most likely through `ffdccInt`'s user documentation.

Chapter 2. Installing and Maintaining FFDC

The First Failure Data Capture utilities are designed to operate with a minimal amount of administrative overhead. There are some aspects of these utilities that may occasionally require maintenance by system administrators. This chapter discusses these aspects.

First Failure Data Capture Installation Media

The First Failure Data Capture utilities are part of the RSCT core utilities software packages. The **rsct.core.utils** installation package contains this software. Install this software in accordance to the instructions given for this package on all nodes in the cluster where you wish to make use of the FFDC utilities (See the *Packaging and Installation* document for instructions.)

First Failure Data Capture Directory

When **rsct.core.utils** is installed on a node, a directory is created on that node for use by the First Failure Data Capture utilities. This directory resides by default in the **/var** file system: **/var/adm/ffdc**.

Permissions are set on this directory so that any user can create file within this directory. Within this directory, two other directories reside:

stacks

The directory where the FFDC utilities store FFDC Error Stacks. Because these stacks are implemented as binary AIX files, they cannot be viewed by using a normal text editor. Each file is owned by the AIX user that created the FFDC Error Stack.

dumps

The directory where detail data files used by the **fcpushstk** and **fclogerr** commands, and their equivalent programming interfaces, are copied. FFDC imposes no format on these files. Each file is owned by the AIX user that submitted the file to the FFDC utilities.

Permissions on the **stacks** and **dumps** directories permit any AIX user to create files in these directories. Only the owner of a file or the **root** superuser, may remove files from these directories. Only the **root** superuser may remove FFDC Error Stack or detail data files owned by another AIX user.

fcclear is the only supported means for removing any files from either of these directories. Do not use the **rm** command to remove files from within **/var/adm/ffdc**.

Preventing Full Consumption of the /var File System

The First Failure Data Capture utilities are implemented with fail-safes, designed to prevent these utilities from consuming all available space in the **/var** file system. Because **/var** is used by the AIX operating system for administrative purposes, it is vital that some space always be available in this file system.

FFDC ensures that it will always leave at least 5% of the space in the **/var** file system available when an FFDC Error Stack is created. By default, the FFDC utilities allocate a 32KB file to store an FFDC Error Stack in **/var/adm/ffdc/stacks** when a process creates an FFDC Error Stack. If reserving a 32KB file would not leave 5% of the space in **/var** available, the utilities attempt to allocate a smaller file, decreasing the amount of space in the file by 2KB in each attempt. The utilities repeatedly attempt to reserve an FFDC Error Stack of smaller and smaller size until a file can be reserved that allows 5% free space to be retained as a cushion, or until an attempt fails to allocate an 8KB file. If an 8KB file cannot be allocated, the utilities report that an FFDC Error Stack cannot be allocated.

First Failure Data Capture Error Stack files are allocated when a process first attempts to record information to the FFDC Error Stack. Once allocated, the size of the FFDC Error Stack is set; it does not

increase or decrease. This ensures that a process has sufficient space to store a meaningful amount of data. It also ensures that an FFDC Error Stack file will not grow to consume all available space in the file system.

The FFDC utilities are designed so that they do not consume all available space in the file system containing the `/var/adm/ffdc` directory. However, not all utilities that record information to the `/var` file system are designed in such a manner. Even though the FFDC utilities ensure that they will not consume all available space in `/var`, available space in that file system may be consumed by other applications. Conditions may arise where FFDC is incapable of creating FFDC Error Stacks because other processes have consumed too much space in the `/var` file system. Fortunately, this problem can be rectified with minimal effort.

Because the FFDC utilities record information to their own directory, this directory can be mounted onto a separate file system. When you mount `/var/adm/ffdc` onto another file system, you ensure that FFDC does not consume space in `/var` that is needed by other applications. Similarly, other applications do not consume space that could be used to store FFDC Error Stack and detail data information.

Maintenance of the First Failure Data Capture Directory

First Failure Data Capture Error Stack files are only created if a process actually records data to the FFDC Error Stack. In other words, the `fcinit` command and the `fc_init` subroutine do not allocate an FFDC Error Stack file. The files come into being when `fcpushstk` or `fc_push_stack` are used for the first time to record failure information after an FFDC Error Stack Environment has been established.

Because the FFDC Error Stack is designed to be a persistent storage device, the file that contains the FFDC Error Stack contents remains on the system after the process is completed. The file will remain until the file is explicitly removed, either by the process owner or by the `root` superuser. This can pose a problem if an FFDC Error Stack user neglects to remove FFDC Error Stacks that are no longer needed, or misunderstands the intended use of the FFDC Error Stack and creates one every time the process executes. Each FFDC Error Stack file can be anywhere from 8KB to 32KB in size; if too many of these files exist on a node, sooner or later, a process that needs an FFDC Error Stack will be unable to allocate one.

Administrators should set a policy for removing the contents of the FFDC directory on a regular basis. Set this policy in accordance to your own particular needs. Using the `fcclear` command in conjunction with an automated administration command such as `cron` should be sufficient for clearing out the contents of the FFDC directory.

Note: Do not remove file in the FFDC directories using the `rm` command. `fcclear` is the only supported means for removing files from the FFDC directory. `fcclear` provides all the necessary options for specifying the files to remove or retain. See “`fcclear` Command” on page 105 for details on this command.

Disabling First Failure Data Capture Error Stack Usage

A control is provided to disable the creation of First Failure Data Capture Error Stacks on a node. The scope of this control depends on how the control is set, but at most, it can only disable FFDC Error Stack use on a node. The control does not operate on multiple nodes at the same time. To disable FFDC Error Stack use on multiple nodes, the control must be set on each node individually.

The `FFDCSDISABLE` environment variable governs whether the FFDC utilities can make use of an FFDC Error Stack. By default, this environment variable is not set when the FFDC utilities are installed. To disable FFDC Error Stacks, define this environment variable and set it to any non-null value. For example:

```
prompt> export FFDCSDISABLE=y
```

An environment variable was chosen to provide a simple and easy mechanism for disabling FFDC Error Stack use, and provide the control with sufficient flexibility. This control can disable FFDC Error Stacks for the entire system, certain users, or specific shells, depending on how it is defined.

To disable FFDC Error Stack use for an entire node, the **FFDCSDISABLE** environment variable can be defined in the **/etc/environment** file. Any process created on the system will inherit this environment variable, and will be unable to create an FFDC Error Stack. This control can be set when a specific file system for the FFDC directory cannot be created, and space in the **/var** file system is at a premium.

To prevent specific users from creating FFDC Error Stacks, insert a definition of the **FFDCSDISABLE** environment variable in the **.profile** of these users. Any process created by the user will inherit this environment variable, and will be unable to create an FFDC Error Stack. Set this control if specific users seem to be abusing the use of FFDC Error Stacks.

Users can set the **FFDCSDISABLE** environment variable in specific shells to prevent any process created within that shell from creating an FFDC Error Stack. This control can be set in shell scripts to prevent commands executed by these scripts from creating FFDC Error Stacks. Such a use would be permitted if a command executed from a script were expected to fail, and the script user did not intend on using the failure information from the command.

Diagnosing Common First Failure Data Capture Utility Problems

First Failure Data Capture provides the **fccheck** command to perform basic tests to determine if there are any system conditions that would prevent the FFDC utilities from functioning. See “fccheck Command” on page 108 for specifics on the tests performed and on how to correctly invoke this command.

Chapter 3. Constructing Persistent Records of Failures

When a failure occurs on a customer's system, the customer and the application's support services use the following information to understand the failure:

- Failure symptom experienced by the customer
- Failure notification from the software, if any
- Record generated by the failing software to persistent storage, if any

It is important that each item of information be as descriptive as possible. This chapter concentrates on making effective use of persistent storage to leave behind meaningful information of a failure for use by problem investigators, be they the customer or the application support service.

What Failure Recording Facility Should Be Used?

This section will concentrate on the use of two persistent storage mechanisms:

- AIX Error Log
- First Failure Data Capture (FFDC) Error Stack

This section also deals with failures that the software is capable of detecting.

When the AIX Error Log Is Appropriate

The AIX Error Log should be used whenever software needs to report a condition that requires system administrator attention, or to inform the system administrator of specific system status. Any software that does not have a control terminal or terminal access—daemon processes, kernel extensions, and device drivers—should use these devices to report failures, since they have no other way of bringing these conditions to the administrator's attention.

The AIX Error Log is not meant to be used as a trace facility. It is oriented towards the system administrator. Information about failures and noteworthy incidents go here, not debugging information from applications. Debugging information should be recorded to a different device, such as the FFDC Error Stack or a trace file, or both.

First Failure Data Capture provides an interface that can be used by any process running outside of the AIX kernel. Commands, libraries, GUIs, and daemon processes can use this interface to record failure information to the AIX Error Log. No FFDC interface is provided for kernel extensions; this software must use the **errsave** kernel service to record information to the AIX Error Log.

When the FFDC Error Stack Is Appropriate

The FFDC Error Stack is used to trace a failure in a top-level software component to the root cause of the failure in the bottom-level or mid-level software components. It can be used whenever multiple processes are started to carry out a specific function, and these processes have a parent-child or ancestral relationship to one another. The FFDC Error Stack can also be used by stand-alone processes that invoke multiple threads, levels of libraries or subroutines to accomplish its tasks.

The AIX Error Log is used when the system administrator should be informed of an incident. The audience for FFDC Error Stacks is different: it is oriented towards the application developer, trying to locate problems in an application where multiple processes, threads, or libraries are involved. It is safe to record debugging information to an FFDC Error Stack, since that would assist in isolating problems in the software. Remember that it is oriented towards problem determination, so reserve its use for recording information about failures, unexpected incidents, or abnormal execution path processing. If a failure should be brought to the system administrator's attention as well, then also record information to the AIX Error

Log in addition to the FFDC Error Stack, because system administrators expect this type of information in the AIX Error Log. Record trace information to a facility designed for tracing purposes, not the FFDC Error Stack or the AIX Error Log.

If a parent-child relationship does not exist between processes, a single FFDC Error Stack cannot be constructed to trace failure conditions between the processes¹. However, FFDC Error Stacks can be linked to associate failures in one of the processes with a failure in the other process. While the FFDC Error Stack works for situations such as the grandp and parent scenario mentioned earlier, it would not work in the case where an application makes a request of an already running service, such as an Event Management application making requests of RSCT Event Management.

Programming libraries, applications, daemons, and scripts can make use of the FFDC Error Stack. The FFDC Error Stack is not useful for kernel extensions, device drivers, or applications that will execute on software platforms other than AIX.

When AIX Error Log and FFDC Error Stack Are Used

The FFDC utilities are designed so that a process can use both the AIX Error Log and the FFDC Error Stack at the same time. Because the FFDC interfaces will generate FFDC Error Identifiers whenever either location is used, the AIX Error Log entries can reference FFDC Error Stack entries and vice versa. The question someone might rightfully ask is, "when would I even consider using both at the same time"?

A likely case is when the software performs some administrative task and is also invoked as a child process or a parent that uses the FFDC Error Stack. In these cases, the software should record failures to the AIX Error Log, because that is where system administrators will expect to find any record of failure. The software is also being used by a process that is using an FFDC Error Stack, so the software should also record information to the stack as well. This does amount to some degree of extra work, but it supports problem determination from two perspectives: the system administrator's perspective, and the parent process user's perspective. If the software were to choose to support only one of these devices, either the system administrator or the parent process user would be unable to trace the failure condition. By using both, either may trace the failure condition.

Another case where both might be used is in a system administration script. If a utility is designed to start a number of subprocesses and the subprocesses, in turn, may start other subprocesses, the parent control script should create an FFDC Error Stack so that any failures detected by the subprocesses can be captured in the same error stack, and the problem symptom can be traced down to the root cause. However, the parent script may also record an indication of overall success or failure to the AIX Error Log, where the system administrator would expect to find records of system failures. The script may choose to do this in case the script is invoked automatically, or from some parent other than the command line, where the command's output might not be readily available or noticeable.

Which Storage Works Best For My Software Type?

Here are some rules of thumb for deciding which persistent failure information device to use. These suggestions are organized by software application types, to give a different point of view on the selection process.

Device Drivers

Device drivers would use the AIX Error Log exclusively as the instrument for recording failure information. Device drivers execute in kernel mode, and would use the **errsave** kernel service to record this information to the AIX Error Log. The FFDC Error Stack would not be used, since this facility would be unavailable in kernel mode. The command interfaces and library interfaces that

1. These are situations where a failure condition in one process impacts the functioning of the other process, but these processes are not related to one another. The processes were started by separate parents, and possibly execute on separate systems. In these cases, an FFDC Error Stack cannot capture failure information from both processes into a single stack.

invoke the device driver on behalf of a client process would consider supporting the FFDC Error Stack, since these interfaces can execute within the client process's context.

Information on internal code failures and failures to use needed resources should be logged to the AIX Error Log and reported to the device driver client. These are failures that require the attention of a system administrator to resolve. The client must understand that the failure is a result of conditions beyond the client's control, but reporting the failure condition to the client will not result in the information reaching the system administrator. The AIX Error Log record serves as a notification to the system administrator.

Failures caused by environmental conditions, invalid usage or incorrect input from clients should not be logged to the AIX Error Log. The device driver should instead inform the client of the failure. Administrator intervention is not required in these cases.

Daemons

Daemons would use the AIX Error Log as the primary instrument to record information about detected failure conditions. Daemons that use libraries that support FFDC Error Stack use can also create an FFDC Error Stack Environment, so that failures from the underlying libraries and child processes can be captured along with the daemon's failure information.

Daemons should record information about status changes to the AIX Error Log, to leave an audit trail of the daemon's activities. At a minimum, daemon startup and shutdown should be logged using informational templates (type INFO). Status changes to tracing or debug mode and back to normal mode should be logged if possible. Status changes should not be reported to the FFDC Error Stack.

Information on internal code failures and failures to use needed resources should be logged to both the AIX Error Log and the FFDC Error Stack, if both are being used. These are failures that require the attention of a system administrator to resolve. Reporting the failure condition to the client does not result in the information reaching the system administrator. The AIX Error Log record serves as a notification to the system administrator. The FFDC Error Stack entry allows the problem investigator to understand how the failures recorded in the AIX Error Log related to the failure in the daemon itself.

Failures caused by environmental conditions, invalid usage, or incorrect input from clients should not be logged to the AIX Error Log or the FFDC Error Stack. The daemon should instead inform the client of the failure. Administrator intervention is not required in these cases.

Libraries

Libraries would use the FFDC Error Stack as the primary instrument for recording information about failures. The library would not create the FFDC Error Stack Environment, and this task should be left to the application using the library. Only in rare cases would a library interface consider making a recording to the AIX Error Log, and only in conditions where system administrator attention is required. Normally, library interfaces would leave the decision of whether the failure warrants operator intervention to the application using the library.

The rare cases where libraries would consider recording failure information to the AIX Error Log involve internal code failures and failures to use needed resources, both being conditions where administrator attention is required. The library interface would record information about internal failures or resource failures to the FFDC Error Stack and the AIX Error Log. By recording this information in the FFDC Error Stack, a problem investigator can determine how the failure in the library interface impacted those applications dependent upon it. Recording the incident to the AIX Error Log will allow the condition to come to the system administrator's attention.

Failures caused by environmental conditions, invalid usage or incorrect input from clients should not be logged to either the FFDC Error Stack or the AIX Error Log. Instead, the library interface should inform the client of the failure.

Applications

Applications would use the FFDC Error Stack as the primary instrument for recording information about failures. Applications would be responsible for either creating the FFDC Error Stack Environment, or for inheriting the existing FFDC Error Stack Environment:

- Applications that do not create child processes or threads or make use of libraries that support FFDC Error Stacks would not normally create an FFDC Error Stack Environment except for developmental debugging purposes. Instead, these applications would inherit the current FFDC Error Stack Environment, if it exists.
- Applications that expect to be invoked as part of a long series of commands from some other source (such as a shell script) would also not create an FFDC Error Stack Environment. Instead, these applications would inherit the current FFDC Error Stack Environment, if it exists.
- Applications that create child processes or threads or make use of libraries that support FFDC Error Stacks would consider creating an FFDC Error Stack Environment if it does not exist.

Applications that perform system administrator functions, or that change the status or availability of system resources, should also consider making use of the AIX Error Log to record information about failures or status changes in system resources. These applications are meant for system administrator use, and system administrators expect to find failure information for system resources in the AIX Error Log. Applications that do not perform these functions would not record failure information to the AIX Error Log unless the application has no access to a terminal to report failure information.

Information on internal code failures and failures to use needed resources should be logged to the FFDC Error Stack, and to the AIX Error Log if it is also being used. These are failures that will require further investigation to resolve. The client cannot be expected to perform this investigation, since the client may be automated software. The FFDC Error Stack entry will allow the problem investigator to understand how the failures recorded in the Error Stack related to the failure in the command itself.

Failures caused by environmental conditions, invalid usage or incorrect input from clients should not be logged to either the FFDC Error Stack or the AIX Error Log. Instead, the application should inform the client of the failure.

Scripts

Scripts use the same recommendations as **Applications**.

To make the most effective use of the AIX Error Log, use error logging templates that are specific to the failure condition. These templates should not only report the failure condition detected (such as a failure in the read system call), but what the failure means to the software (such as a failure to obtain the configuration information for the software, preventing the software from providing any function) and what the appropriate response to the failure is.

Constructing Meaningful Templates for AIX Error Logging

The appropriate method for using the AIX Error Logging facilities is to create a unique template for each possible failure condition. *Unique failure condition* means any failure with the following in common:

- A common failure nature or description
- Common potential causes of the failure condition
- Common impacts to the software
- Common actions to take in response to the condition

This information must be incorporated within the AIX Error Logging template itself. The software must not attempt to use the Detail Data section of the template to convey this information.

Consider the following **errpt** report executed on an SP node:

IDENTIFIER	TIMESTAMP	T	C	RESOURCE_NAME	DESCRIPTION
12081DC6	0622133998	P	S	grpsvcs	SOFTWARE PROGRAM ERROR
9EEBC3C4	0622162298	I	0	Eclock	Eclock command issued by user
:	:	:	:	:	:
EC771C6B	0619172798	T	S	css	Switch daemon SDR communications failed

The initial entry is an entry that reports a failure by using a general-purpose template. However, the next two entries in the example give better information in the brief report. The second entry is an informational message to the system administrator to report that the **Eclock** command had to be invoked by an end user. Enough information is provided in this report for the administrator to understand what is being reported by that specific AIX Error Log entry. The entry may indicate a problem (why would anyone issue an **Eclock** command on their own?), but since the message is informational, it does not report a condition that requires immediate attention. The final entry reports a failure of the Switch software to communicate with the System Data Repository. Once again, enough information is presented in the brief report for the system administrator to understand the nature of the failure recorded in that entry without requiring a fully detailed report to determine this. From this report, the system administrator may decide that the final entry is more severe than the second entry.

The recommended methodology for using the AIX Error Log does not necessarily mean that software will need to generate one unique error logging template for each line of code that can possibly fail; that takes the recommendation to the extreme. However, the software must not create a general-purpose template for recording failures. The template itself must convey all the information necessary for the system administrator to understand what the problem is and how to react to the problem *without having to examine the detail data to determine this*.

Consider the above example again. The second and final entries are templates created by the Communications Subsystem for PSSP: the Switch software. The final template is used to report failures that can occur in several locations within the Switch software. The failure conditions are slightly different: in one place, one system call fails, in another place, a different system call fails. However, all these failures have a common nature, common potential causes, common impacts on the Switch software, and common actions for the administrator to take in response to the failure:

```
-LABEL: SP_SW_SDR_FAIL_RE
IDENTIFIER: EC771C6B
```

```
Date/Time: Mon Jun 22 16:36:17
Sequence Number: 959
Machine Id: 000084208900
Node Id: k21n01
Class: S
Type: TEMP
Resource Name: css
```

```
Description
Switch daemon SDR communications failed
```

```
Probable Causes
Ethernet overloaded
Excessive SDR traffic
SDR daemon or control workstation down
Software Error
Failure Causes
Excessive ethernet traffic
SDR daemon not running
```

```
Recommended Actions
Check if SDR daemon is up
Call software service if problem persists
Detail Data
```

```
Software ID String
LPP=PSSP,Fn=set_node_info.c,SID=1.9,L#=469,
Failure cause
switchResponds saw an API failure
```

The template reports the nature of the failure, the potential causes, the severity of the failure (it is classified as a TEMPorary failure, not a PERManent one), and the actions the administrator should take in response to the failure. The Detail Data only indicates what specific failure was detected and where it was detected. This template conveys all the necessary information to the administrator *within the template itself*, instead of relying upon the Detail Data to convey this information. This is only one of many templates installed by the Switch software. This software package does not create a single template for every single possible failure condition, but it does provide an unique template for failures that have the same nature, the same possible causes, the same impacts, and the same suggested response actions.

AIX Error Log Template Now Support XPG/4 Message Catalogs

Starting with AIX version 4.3.2, software developers can create an error logging template using messages from an XPG/4 message catalog. This permits software developers to extend their own message catalogs to include meaningful "code points" to be used to construct more detailed and descriptive error templates. There is a "catch" to using messages, which will be addressed shortly. For more information, consult the **errupdate** command documentation.

The "Catch" in Using Messages

Messages can be used in creating AIX Error Log templates, so long as the templates do not need to be alertable. Alertable templates must be built from AIX Error Log template code points.

Selecting An Appropriate Log Event Type

The information provided as detail data for a failure is selected by the software to provide the greatest assistance to debuggers. The AIX Error Logging templates provide 230 bytes for containing detailed data, which can consist of up to 12 separately labelled items of information. To understand how this section is used in the error logging template, consult the command documentation for **errupdate**.

Before constructing the template, the severity of the failure to be recorded in the template must be known. First Failure Data Capture defines 6 categories, known as *log event types*. These types translate into specific codes to be used in the construction of the AIX Error Logging template, and also codes used to report failures in the BSD System Log when that facility is used on other UNIX-based platforms. The type definitions have been chosen to match those used in the former **Error Logging Policy**, to assist conversion efforts to First Failure Data Capture.

Table 3. First Failure Data Capture Log Event Types

Hierarchy	Log Event Type	Definition
1	FFDC_EMERG	A log event that is a severe failure and the node is in danger of coming offline. This information is required by the system administrator to bring the node back online.
2	FFDC_ERROR	A permanent failure that will persist until repaired. The node is not in danger of coming offline, and remains bootable.
3	FFDC_STATE	An event of some significance has occurred. This is not a failure condition.
4	FFDC_PERF	A condition that can or will degrade system performance below an acceptable level. The node is not in danger of coming offline, but performance may be unacceptable and causing random failures in end-user applications (such as timeouts).

Table 3. First Failure Data Capture Log Event Types (continued)

Hierarchy	Log Event Type	Definition
5	FFDC_TRACE	The name and location of a trace file generated by the software. Used when a tracing option has been set within the software itself. Software should not create an entry of this type unless a tracing option is active.
6	FFDC_RECOV	A recovery action has been successfully completed by the software in response to an FFDC_EMERG or an FFDC_ERROR condition. Software creates these entries only after an FFDC_EMERG or and FFDC_ERROR condition as been detected and an automated recovery action has been completed successfully.
7	FFDC_DEBUG	A failure condition was detected. The node is not in danger of coming offline, and remains bootable. The failure is not permanent, or the software can continue with the failure condition present.

These event log types map to the following values for the AIX Error Log and the BSD System Log:

Table 4. Mapping FFDC Types to AIX Error Log and BSD System Log Types

FFDC Event Type	AIX Error Log Type	AIX Error Log Description	BSD System Log Priority	BSD System Log Description
FFDC_EMERG	PEND	The loss of availability of a device is imminent	LOG_EMERG	Emergency condition, system is unusable
FFDC_ERROR	PERM	An unrecoverable condition, a permanent failure	LOG_ERR	Error condition
FFDC_STATE	INFO	Informational message to administrator	LOG_NOTICE	Normal but significant condition
FFDC_PERF	PERF	Performance degraded below acceptable level	LOG_WARNING	Abnormal but recoverable condition
FFDC_TRACE	UNKN	Not possible to determine the severity of the error	LOG_INFO	Informational message for system administrator
FFDC_RECOV	TEMP	Condition was recovered after a number of unsuccessful attempts	LOG_DEBUG	Debugging message
FFDC_DEBUG	UNKN	Not possible to determine the severity of the error	LOG_DEBUG	Debugging message

The following table gives advice for selecting the error log type appropriate for the incident being reported by the software:

Table 5. Mapping Software Incidents to FFDC Event Types

Software Incident	Example of Failure	FFDC Event Type	AIX Error Log Type
Status	Software service being started; Software service being taken offline or shutting down in a normal fashion	FFDC_STATE	INFO
Status	Trace file being opened or closed - name of trace file recorded in Detail Data section	FFDC_TRACE	UNKN
External Failure	Configuration error is detected that prevents the software from executing at all; Software detects an unsupported environment that prevents it from executing.	FFDC_ERROR	PERM

Table 5. Mapping Software Incidents to FFDC Event Types (continued)

Software Incident	Example of Failure	FFDC Event Type	AIX Error Log Type
External Failure	Configuration error is detected but software is able to continue or uses defaults to continue; Software detects an unsupported environment but is able to continue; Invalid options are bypassed or defaults used.	FFDC_DEBUG	UNKN
Resource Failure	Resource is unavailable and software is unable to continue; Resource access is denied although a valid access attempt was made.	FFDC_ERROR	TEMP
Resource Failure	Primary resource was unavailable, software is proceeding without it or a backup resource was used - a notification is being made so that the system administrator can investigate the failure in acquiring the resource at a later time.	FFDC_RECOV	TEMP
Resource Failure	Resource appears to be performing at a degraded level (network is congested, response time from the resource is slow, requests time out and have to be retried), but software can continue to make use of the resource - a notification is being made so that the system administrator has some record of the performance problem.	FFDC_PERF	PERF
Internal Failure	An internal problem is detected in the mainline code path; An internal problem is detected in a code path that cannot be disabled internally by the software.	FFDC_ERROR	PERM
Internal Failure	An internal problem is detected in a secondary code path that can be bypassed or disabled by the software (and the software sets controls to disable this code path); An internal problem is detected but the software continues to execute.	FFDC_DEBUG	UNKN

Carefully Describe the Failure Condition

When using an XPG/4 message to describe the error, construct the description carefully. Most system administrators will be making use of the standard **errpt** command to examine the contents of the AIX Error Log. This command truncates the error description to 38 characters. Recall the earlier example of **errpt**'s default output:

```
IDENTIFIER  TIMESTAMP  T  C  RESOURCE_NAME  DESCRIPTION
12081DC6   0622133998  P  S  grpsvcs        SOFTWARE PROGRAM ERROR
9EEBC3C4   0622162298  I  0  Eclock         Eclock command issued by user
EC771C6B   0619172798  T  S  css            Switch daemon SDR communications failed
```

The message must therefore convey the most meaningful information possible in the first few words of the message. A message such as:

```
FFDC server detected a failure in peer on remote node
```

would not be practical, since the most significant information is at the tail end of the message, and is likely to be truncated off the message by the **errpt** command:

```
FFDC server detected a failure in p
```

In spite of the effort to make the system administrator's task easier, this message once again forces the system administrator to obtain a detailed report to determine what the record is really reporting.

Don't list the software application name in the actual description, unless the *resource name* will not identify the software. The earlier message could drop the "FFDC server detected" phrase, and use a *resource name* of FFDC to identify the software making the record.

Do not be myopic in the error description. Avoid phrasing the failure in terms of a problem symptom such as a system call or a module name. Phrase the failure in terms of the software's function or logical structure. In the earlier message, it wouldn't not have been appropriate to describe the failure as: `select()` timed out or `write()` failed on socket. These are the indicators of the problem or details of the problem situation, but not the actual problem itself. In this case, the actual problem is a failure to communicate to the peer on a remote node. The failure should be phrased in those terms, not the specific failure symptom or indicator. Use the specific symptoms instead in the various *cause* fields of the AIX Error Log template, or in the *detail data* fields of the template.

Place the most important information first in the message. This is the information the system administrator wants to see in a brief report, and will be used by the administrator to decide whether further investigation of the incident is warranted. Placing the information first will prevent it from being truncated. The earlier message could have been phrased as: Peer failure on remote node.

If the software application is terminating, or a software service is rejecting a service request as a result of the incident, indicate this in the message. Place this information after the most significant information in the message; this information can be truncated from the message without significant loss of information. If the `ffdcserv` daemon were terminating because of the failure of its peer on the remote node, the message could be structured as Peer failure on remote node - exiting.

Organize the Causes and Their Associated Actions

The AIX Error Log template provides four *cause* fields. Three of these *cause* fields have associated *action* fields:

- Probable Cause
- Failure Cause and Failure Action
- Install Cause and Install Action
- User Cause and User Action

A *probable cause* **must** be provided in the template, or the **errupdate** command will not process the template source code. List the actual symptom detected by the software in this field. For instance, if this template reports a peer's failure on a remote node because the communication link was lost, list the communication failure as the probable cause.

Up to four *probable causes* can be listed. This permits a template to be used for a failure condition that can be noticed in several locations within the source code, where they may have slightly differing causes. For example, the software can detect a failure in the peer in several ways, including a failure to receive a response within an expected period of time, or a failure to send information to the peer through a previously established connection. The template can be used to report this failure both when the timeout occurs or the transmission failure occurs, provided both causes are listed. When providing multiple causes, place the most likely causes first, followed by the less likely causes. The purpose is to lead the system administrator or the problem investigator to the cause in the majority of the cases, so the most likely causes are always listed first.

Use the *failure cause*, *install cause*, and *user cause* fields to indicate possible sources of the failure condition. The *probable cause* indicated the actual symptom, or the "what" of the failure. In the remaining *cause* fields, the template attempts to indicate the reasons "why" a failure might have occurred. One of these failure fields must be provided in a template, but any or all can be provided.

Use these guides when determining what other *causes* to use:

- Use *failure causes* to report internal failures and resource failures.
- Use *install causes* to report failures resulting from improper installation, configuration, or setup of the software.
- Use *user causes* to report causes that the user or system administrator might have induced.

For some failures, multiple sources of failure may be possible. For example, the software may be failing because the software cannot access a needed resource that is identified in the software's configuration. The failure may have been induced because the software configuration failed or did not complete correctly, so an *install cause* should be identified. However, the software's configuration may be valid but the resource itself is unavailable or failing, so a *failure cause* should be identified. If the software is unable to state for sure which condition induced the problem, *both* causes should be listed.

For each other *cause* type, up to four causes can be listed. Once again, these items should be listed starting with the most likely, and ending with the least likely.

Whenever any other *cause* type is listed in the AIX Error Log template, a corresponding *action* needs to be listed, or **errupdate** refuses to budge. Only one corresponding *action* needs to be listed, even if all four possible *causes* were listed. These *actions* are instructions on how to deal with the associated *causes*. If a configuration failure was listed in *install causes*, the *install actions* would indicate how to address the misconfiguration, or where to look for assistance on resolving the problem. If a usage error was described in the *user causes*, the *user actions* would indicate how this misuse can be corrected, or direct the user to the appropriate location where this information can be found. If a code bug was listed in *failure causes*, the *failure action* would give advice on collecting information and contacting the application's support services.

Often, the software developer will identify a source of a problem that can be addressed through user action. For instance, a resource may not be available, but the system administrator can address this by making a backup resource available. The natural reaction would be to list the resource failure as a *failure cause*, but list the system administrator's workaround as a *user action*. Unfortunately, the **errupdate** command does not permit this. The system administrator workaround must be listed as a *failure action*, or a new *user cause* has to be added to the template.

Do Not Place Descriptive Information in Detail Data

The Detail Data section is not translated by the **errpt** command. The information is recorded to the AIX Error Log using the locale where the software itself is executing. Since this information cannot be translated and might be recorded in the "wrong" locale, Detail Data should not be used for descriptive information. Instead, the template itself has to be constructed to adequately describe the incident.

Making Correct Use of Detail Data

Detail Data is the *only* portion of the AIX Error Log template that is not modifiable at execution time. This area is reserved for details on the failure condition. It is here where the software will record items such as variable values, failing routine names and return code, or small dumps of data areas. The area is intended mostly for use by problem investigators, IBM Customer Support, and software development.

Three fields are reserved for use by the FFDC utilities. These fields must be defined as the first three fields in any template that is used through the **fclogerr** command or the **fc_log_err** subroutine:

```
Detail_Data = 46, 00A2, ALPHA
Detail_Data = 42, EB2B, ALPHA
Detail_Data = 42, 0030, ALPHA
```

The first field will have the message "DETECTING MODULE" associated with it, and will be used to indicate the licensed program product name, the source code file name, the SCCS version number of the source code module, and the line of code location where the failure is being reported.

The first field stores this information by using the following format:

```
<lpp_name>,<file_basename>,<SCCSID>,<line_of_code_num>
```

The second field has the message ERROR ID associated with it, and is used by the First Failure Data Capture utilities to store the FFDC Failure Identifier for this specific log entry.

The third field has the message REFERENCE CODE associated with it and is used by the First Failure Data Capture utilities to store the *FFDC Failure Identifier* of a previously recorded failure that caused the failure recorded in this entry to occur.

The fourth through twelfth *detail data* fields can be constructed to best assist the software developer to record pertinent information about the failure condition itself for isolation and debugging purposes. At the very least, one more field should be provided to contain specific failure information:

```
Detail_Data = 100, 001E, <whatever_makes_sense>
```

This will associate the message "DIAGNOSTIC EXPLANATION" to the fourth field, with the data taking the form specified by the <whatever_makes_sense> option. However, the remaining detail data fields should be selected to associate the best possible messages to the data, so that the software doesn't have to consume what little detail data room is left to store descriptions of the data itself. For example, examine the following Detail Data definition:

```
Detail_Data = 46, 00A2, ALPHA
Detail_Data = 42, EB2B, ALPHA
Detail_Data = 42, 0030, ALPHA
Detail_Data = 16, 0027, ALPHA
Detail_Data = 4, 8183, DEC
Detail_Data = 4, 8015, DEC
Detail_Data = 60, EB00, ALPHA
```

These definitions give a Detail Data section to the AIX Error Logging Template that reads as follows:

```
Detail Data
DETECTING MODULE
ERROR ID
REFERENCE CODE
FUNCTION
RETURN CODE
ERROR CODE AS DEFINED IN sys/errno.h
FILE NAME
```

The Detail Data section is constructed so that space is not consumed in the detailed data itself to say things like function %s failed: rc = %d, errno = %d, details dumped to file %s. Instead of using space to describe the data, the field descriptions describe the data, leaving the space available to store the actual failure information:

```
Detail Data
DETECTING MODULE
PSSP,harmld,1.16,408
ERROR ID
.IyzQgyNxiCr.CH41xRXQ7.....
REFERENCE CODE
.eI02I/9wiCr.4Vc1xRXQ7.....
FUNCTION
shmget
RETURN CODE
-1
ERROR CODE AS DEFINED IN sys/errno.h
17
FILE NAME
/var/adm/ffdc/dumps/harmld.10335.1001103298
```

Detail Data fields can also be labeled using XPG/4 cataloged messages as well as the standard AIX Error Log code points. Consult the **errupdate** command documentation for instructions.

When creating AIX Error Log templates, software developers often ask for a way to have substitutional parameters in the *error description* or the various *cause* fields. For instance, the software may wish to report a missing file resource to the AIX Error Log. However, the software deals with multiple files, any one of which may not be accessible. Software developers often ask for a substitutional parameter in the *error description* or *cause* fields, which will permit them to list the actual file. The belief is that without this

capability, individual templates will have to be made for each possible file. If these files can be dynamically defined, then building a proper template seems impossible.

For cases like this, it is best to use the Detail Data to contain the "substitutional" information. The *probable causes* and the other *causes* should indicate the failure in terms as specific as can be managed. One of the listed *causes* should then indicate a specific field in the Detail Data section that contains the "substitutional" information. Using the previous example, a template could be constructed to report the failing file:

```
Err_Desc = {1, 10, "Configuration File Access Failure - Exiting"}
Prob_Causes = {1, 53, "Cannot open configuration file in read-only mode"}, \
              {1, 54, "Consult Detail Data for configuration file name"}
Fail_Causes = {1, 60, "Configuration file does not exist"}, \
              {1, 61, "File permissions do not permit read-only access"}, \
              {1, 62, "One or more directory components do not permit search access"}, \
              {1, 63, "Directory does not exist or is not mounted"}
Fail_Actions = {1, 101, "Recreate configuration file"}, \
               {1, 102, "Verify and reset permissions of the configuration file"}, \
               {1, 103, "Verify and reset permissions on file directory path"}, \
               {1, 104, "Verify or mount directory"}
Inst_Causes = {1, 64, "Configuration failed"}
Inst_Actions = {1, 105, "Verify the software configuration"}, \
               {1, 106, "Consult the user documentation for assistance"}
User_Causes = {1, 65, "Configuration file renamed or removed"}
User_Actions = {1, 107, "Replace or recreate configuration file"}
Detail_Data = 46, 00A2, ALPHA
Detail_Data = 42, EB2B, ALPHA
Detail_Data = 42, 0030, ALPHA
Detail_Data = 100, {1, 201, "Configuration File Name"}, ALPHA
```

By using the Detail Data to convey the actual file name, a separate template is not needed for each file, and substitutional parameters are not needed within the other fields of the AIX Error Log template.

Other Rules Governing the Contents of an AIX Error Log Template

Use these rules in constructing the AIX Error Logging template, once the appropriate *error logging type* has been chosen from the tables.

- The error label used by the source code referring to the error template must contain a trailing **_XX** in the label name. The value of the *XX* is the first two letters of the First Failure Data Capture event type (after the *FFDC_* prefix). These event types are described in Table 1 of the Interface Semantics chapter. For example, if a template is being constructed to report a severe failure in the *XYZZ* software that will bring the node down shortly (*FFDC_EMERG*), the template name might be *XYZZ_TERM_EM*. The resulting name, including the trailing **_XX** characters, must be 16 characters or less.
- The *comment* field used in the source file to create the error logging template must contain a string that describes the failure condition where this template will be used.
- The *logging* field must be set to **true**.
- The *report* field must be set to **true**.
- The *alert* field should be **false**, unless the following conditions are true:
 - Any recording of this failure should be viewable by a remote monitoring application, such as NetView.
 - The error logging template is constructed using AIX Error Log code points (viewable through the **errmsg -w** command), and not from XPG/4 messages.
 - The code points are Generic SNA alerts, and fall outside the value range of E800 through EFFF.

If you want a failure to be alertable, please read "Error Logging and Alerts" in the chapter, "Error Logging Facility." of the *AIX Version 4.3 Problem Solving Guide and Reference*. Templates must conform to specific rules in order for the failure to be alertable.

- The *class* field must be supplied. For software applications, the class will be **S** or **O**. Device drivers should record failures in the driver as **S** and failures in the hardware it is supporting as **H**.
- The *error description* field must be provided.
- The *error type* field must use the AIX Error Logging event type that matches the First Failure Data Capture logging type.
- At least one *probable cause* must be identified. This should be the most likely cause of a incident being reported, or the most obvious reason for the incident.
The *failure cause*, *install cause*, and *user cause* fields are used to list more specific causes. The template must contain at least one cause type in this group, and can contain all of them. However, the template *must* contain a *probable cause*.
- At least one other *cause* field needs to be identified. Any and all of the *failure*, *install*, and *user* causes can be provided, but at least one must be provided.
For any additional *cause* fields that are provided, the associated *action* fields must also be provided.
- At least **four** *detail data* fields must be provided. The first three detail data fields are standard for all templates, and are used to contain failure location and identification information. Use of these fields was described in the earlier section. See “Making Correct Use of Detail Data” on page 32.

Using the AIX Error Log via the FFDC Utilities

The AIX Error Log and the BSD System Log exist on any AIX platform. This log records information about failure conditions and other noteworthy conditions that should be brought to the system administrator’s attention. The log is the failure reporting mechanism of choice for kernel extensions and daemon processes, and any other software process that does not have a direct parent process or a terminal to report failure information.

First Failure Data Capture does *not* replace the AIX Error Log with the FFDC Error Stack. In fact, FFDC *recommends* the continued use of the AIX Error Log where it is sensible to do so.

FFDC provides the capability to record to both the AIX Error Log and the BSD System Log at the same time with a single **fclogerr** or **fc_log_error** call. Recordings to the BSD System Log are made for compatibility reasons; the AIX Error Log is the definitive source for failure information on AIX platforms.

The apparent drawback to using the AIX Error Log versus the FFDC Error Stack is that it does not inherently provide the ability to tie root causes of failures in low-level software to the problem symptoms seen by users in the higher-level software components. Not all software that cooperates to complete a specific task operates in a parent-child relationship, so the FFDC Error Stack cannot be used. However, there is still a cause-and-effect relationship between these failures, and this information is just as essential in understanding the cause of the failure as it is in the parent-child relationship handled by the FFDC Error Stack. First Failure Data Capture removes this apparent drawback through the use of the **fclogerr** and **fc_log_error** interfaces and the *FFDC Failure Identifier*.

The FFDC Failure Identifier token: the Link Between Related Failures

While imitating the function of the underlying **errlog** interface, the **fclogerr** and **fc_log_error** interfaces provide a key additional feature. Callers to the **fclogerr** and **fc_log_error** interfaces receive a token from the interface as a response. This token contains enough information to locate the AIX Error Log entry made by the interface - the token contains a code to identify the system, AIX Error Log template, and date used in the record. The token is also stored in the Detail Data of the AIX Error Log entry. By itself, this 42-character token does not appear to be very useful. But by using this token, a problem analyst can locate the entry made by this software to report the failure. This avoids the need to search blindly through the AIX Error Log in search of the specific entry. The token is called the *FFDC Failure Identifier (fid)*.

The FFDC interface user receives this token as an indicator of where the failure report is recorded. Now, what is the caller supposed to do with this token? The intent of providing this token back to the caller is so that the caller can provide this token back to its own client as part of the failure information. The software’s

client can then record this token in its own error information, to establish a link between the failure it experiences because of this software's failure, and this software's failure report.

Required Information In Error Log Templates Used By FFDC Clients

The FFDC **fclogerr** and **fc_log_error** interfaces record information to the AIX Error Log and the BSD System Log. When these interfaces report information to the AIX Error Log, they provide several key items of information in the Detail Data section of the template:

- Name of the source code file where the failure was detected.
- Name of the licensed programming product which provides this software to the system.
- The SCCS version number, or SID, of the source code file detecting the failure.
- The (approximate) line of code position in the source code file where the failure was detected.
- The *FFDC Failure Identifier* of this report; this will be furnished by the **fclogerr** command or the **fc_log_error** subroutine.
- An *FFDC Failure Identifier* of a previously reported failure that likely caused this failure condition to occur; this may have been furnished to the software by a service it used and is recorded by the **fclogerr** command or the **fc_log_error** subroutine in the AIX Error Log to establish the link between these related failures. (See the section, "How the FFDC Failure Identifier Is Passed Back".)

The template must reserve space for this information and define fields in the AIX Error Log template source code to store this information.

AIX Error Logging templates used through the **fclogerr** and **fc_log_error** interfaces must provide at least four Detail Data fields; up to 12 are permitted. The first three fields are used to record this information, and must be defined as follows:

```
Detail_Data = 46, 00A2, ALPHA
Detail_Data = 42, EB2B, ALPHA
Detail_Data = 42, 0030, ALPHA
```

The first Detail Data field is labeled DETECTING MODULE and contains the source code file name, its SID version, the line of code position, and the LPP abbreviation. The second field is labeled ERROR ID and contains the *FFDC Failure Identifier* generated by the **fclogerr** or **fc_log_error** interface; this is the same FFDC Failure Identifier that **fclogerr** or **fc_log_error** will return to its caller. The third field is labeled REFERENCE CODE and contains an optional FFDC Failure Identifier that was reported previously by other software; this field establishes the link between this report and the associated failure condition in other software that caused it. The remaining nine Detail Data fields and 100 bytes of available space can be used as is best suited to report debugging information for this failure condition.

How the FFDC Failure Identifier Is Passed Back

Recall that the AIX Error Log is recommended to daemon processes. How would this particular versions of software provide the FFDC Error Identifier back to their callers?

Daemons implementing a distributed service already employ some sort of network protocol to relay information to one another. As part of this protocol, these daemons already employ some sort of mechanism for relaying requests from one daemon to another, and also for transmitting the results of these requests back to the originating daemon. The FFDC Error Identifier can be passed from a node back to the node originating a request through this protocol. This requires that the protocol be modified to carry additional information, but to avoid impacting performance of the distributed system, this information can be relayed only in a failure case.

Daemons that interface with external clients have a library interface designed for this interaction. Library routines are provided to make requests of a service. Some libraries provide a synchronous response to a query; these interfaces can be extended to include a pass-by-reference parameter that gets modified in a failure case to contain the FFDC Error Identifier of the failure record. Other libraries provide asynchronous notification through signals or through a socket connection. The socket notification can be extended to

include the FFDC Error Identifier as part of the data transmitted to the client through the socket, and can be retrieved using the library routines provided to obtain notification data from the socket.

Recording Information to the AIX Error Log and the BSD System Log

When recording information to the AIX Error Log and the BSD System Log, the caller must provide two key data items:

- An AIX Error Logging template identifier, indicating a template that is already installed on the system. This template must be designed for the specific failure condition being reported, not a general purpose template that will rely on information in the Detail Data section to explain the true error being reported.
- A character string to be recorded in the BSD System Log, which will describe the failure to that device.

At first glance, it looks like the software is repeating information in the call: it is specifying a template that describes the failure to the AIX Error Log, but it is also writing a string to describe the same error to the BSD System Log. Unfortunately, this is true. It is a necessary evil because an application such as the FFDC utilities cannot efficiently obtain an AIX Error Logging template from the system and extract the descriptive information from it to record in the BSD System Log. For this reason, the software must provide both items of information.

Recall that the software is instructed to use a template specific to the condition being recorded because the Detail Data is not translated into the user's locale by the `errpt` command. This information is considered by `errpt` to be debugging information only, not for customer use.

The `fclogerr` and `fc_log_error` FFDC interfaces are used to record information to the AIX Error Log and the BSD System Log. Unlike the FFDC Error Stack, an FFDC Error Stack Environment is not necessary to record information to the AIX Error Log and the BSD System Log. This implies that whenever `fclogerr` or `fc_log_error` is invoked, an entry is made in the AIX Error Log and the BSD System Log. These interfaces also generate a unique *FFDC Failure Identifier* for the report being made, and this 42-character token is recorded as part of the Detail Data of the report in the AIX Error Log. Upon completion, the FFDC Failure Identifier is returned to the caller. The caller is expected to capture this information and provide it as part of the failure notification to its own client or its peer. The client or peer can then reference the failure of this software in its own failure report.

Services invoked by the software may provide the software with an FFDC Failure Identifier as part of the service's failure information. If the service's failure also causes a failure in the software itself, the software must also make a report of the failure, reporting the failure condition in terms of the failure's impact to its own function. In this report, the software must make a reference to the service's failure, to establish the common link between this software's failure and its source in the service's failure. This is done by providing the service's FFDC Failure Identifier as part of the failure information to the FFDC interface. When this is done, the FFDC utilities record the service's failure token as a pointer to this failure's cause. This allows problem investigators to proceed to the cause of the failure, instead of guessing at the cause of the software's failure at this point.

The Programming Model

The `fclogerr` command and the `fc_log_error` subroutine are the interfaces that record information to both the AIX Error Log and the BSD System log.

Unlike their FFDC Error Stack brethren, `fclogerr` and `fc_log_error` always record information to the AIX Error Log and the BSD System Log if these operating-system services are active. If the AIX Error Logging template information appears invalid, `fclogerr` and `fc_log_error` record the failure information using a generic template created by FFDC, `FFDC_DEF_TPLT_TR` (AIX Error Log template identifier 2B4F5CAB). This is a general purpose template that conveys hardly more than a simple notification of "some event occurred" to the system administrator while retaining the links between this record and any related failure record, almost but not entirely defeating the purpose of this notification. To convey better information to the

administrator without forcing the administrator to obtain detailed reports from all nodes (and to understand the American variation of English), the caller is *strongly recommended* to use a template specific to the failure being reported.

The **fclogerr** command and **fc_log_error** subroutine record information about the source code file invoking them: what the file name is, what the line-of-code position is, the SCCS version of the file, and the licensed program product that shipped the file. For this reason, **fclogerr** and **fc_log_error** should be called inline, so that this information can be used to locate the position in the source code where the failure was detected. If the software decides to invoke a subroutine to report this information through **fclogerr** or **fc_log_error**, the failures may appear to be occurring in the failure-reporting subroutine unless the location information is passed to the reporting subroutine and used in the **fclogerr** or **fc_log_error** call. The software is required to duplicate this information through the Detail Data provided to **fclogerr** or **fc_log_error**, which would not be necessary if the routine were invoked inline.

The **fclogerr** command and **fc_log_error** subroutine return a 42-character FFDC Failure Identifier to its caller, even if the FFDC generic template had to be used to report the information. The caller is expected to capture this information and relay it back to its client or its peer as part of its failure notification.

Making Effective Use of the FFDC Error Stack

The FFDC Error Stack is a new utility introduced to capture failure information for software when multiple software utilities are invoked to perform a specific action. The FFDC Error Stack is oriented towards the application programmer, providing a utility to help debug failures in multiple related processes on the same system. The failure information is recorded so that failures in parent processes that are a result of failures in child processes are linked together and shown in sequence, so later problem investigation can determine the root cause of a failure. The concept is based on the Error Stack utility used within the Perspectives system administration tool, which captures failure information about the initial failure and the failures that occur as a result of this initial failure.

The FFDC Error Stack is similar in concept to the AIX Error Log: failure information is stored in a persistent location. The *intended use* of the FFDC Error Stack is *different*: where the AIX Error Log is used to report failures or events to the system administrator that requires the administrator's attention, the FFDC Error Stack is used to trace a failure in a top-level software component to the root cause of the failure in the bottom-level or mid-level software components. This is an important distinction, and is also the reason why software that does not currently record any persistent information about failures to the AIX Error Log will be recording persistent information about failures to the FFDC Error Stack.

FFDC Error Stacks can be used by applications that would normally record failure information to the AIX Error Log—such as daemon processes—to assist in tracing a failure symptom to its root cause. Both facilities can be used at the same time. Why use both facilities? If the application cannot determine exactly what a failure condition means, it can record the condition to the FFDC Error Stack. When the failure can finally be understood in terms of the daemon's function that was affected by the failure, an AIX Error Log entry can be made that references the FFDC Error Stack information.

The FFDC Error Stack Environment

The *FFDC Error Stack Environment* is a term given to a process's environment that has been explicitly enhanced to record failure information to the FFDC Error Stack. First Failure Data Capture utilities create this environment for a process when a process specifically requests it. By default, no FFDC Environment exists for a process. When an FFDC Error Stack Environment exists, failure information can be recorded to the FFDC Error Stack. When the environment does not exist, failure information will not be recorded.

The FFDC Error Stack Environment is optional for processes, and by default, an FFDC Error Stack Environment is not created for each process. This is to permit applications, commands, scripts, and libraries function normally in the usual execution environment. If the software detects a failure in the normal execution environment, the software reports the failure as it normally does, without the additional

recording of information to another location. After all, if a frequently used command fails because someone insists on using it inappropriately, the command should not fill up the FFDC Error Stack space unless someone explicitly wants this information traced.

FFDC Error Stack Environments are established by a process through the **fcinit** command or **fc_init** subroutine. Processes establish an FFDC Error Stack Environment for themselves when the process will create multiple child processes or threads that may invoke more child processes or threads to accomplish the necessary task. In this case, the parent process wants to be able to trace failures at the child or grandchild level or further, tracing these failures and the subsequent failures in higher-level software that occurs because of them. In this scenario, the parent process would create the FFDC Error Stack Environment, and the child processes or threads would detect the environment and record their failure information to the FFDC Error Stack in addition to reporting failures as they normally would.

The FFDC Error Identifier: the Link Between Related Failures

When software records failure information to the FFDC Error Stack through its interfaces, the software receives from FFDC a token as a response. This token contains enough information to locate the FFDC Error Stack entry made by this interface - the token contains a code to identify the cluster, node, i-node of the FFDC Error Stack file, and date used in the record. The token is also stored in the detail data of the FFDC Error Stack entry. Looking directly at the token, the 42-character value doesn't appear to be too meaningful. But after decoding this token, a problem analyst can locate the entry made by this software to report the failure. This avoids the need to search blindly through the multiple FFDC Error Stack files in search of the specific entry. The token is called the *FFDC Failure Identifier*.

The FFDC interface user receives this token as an indicator of where the failure report is recorded. Now, what is the caller supposed to do with this token? The intent of providing this token back to the caller is so that the caller can provide this token back to its own client as part of the failure information. The software's client can then record this token in its own error information, to establish a link between the failure it experiences because of this software's failure, and this software's failure report.

How the FFDC Error Identifier Is Passed Back

Recall that the FFDC Error Stack is useful for commands, scripts, and libraries. How would these particular versions of software provide the FFDC Failure Identifier back to their callers?

Library routines can provide the FFDC Failure Identifier through a function parameter. The caller would provide a pass-by-reference parameter, which the library would modify if it recorded failure information to the FFDC Error Stack. If no information was recorded, or if the FFDC Error Stack is not being used (which is determined by the FFDC interface providing a null-valued FFDC Failure Identifier as a response), this parameter would not be modified by the library routine. If an FFDC Failure Identifier is returned by the interface, the library routine would modify this parameter to reference the FFDC Failure Identifier value.

Applications and libraries do not have a similar facility. Applications and scripts have standard output, standard error, and an exit status code available for their use. Unfortunately, the choice of a 42-character token for the FFDC Failure Identifier removes the exit status from possible use. To make the use of FFDC consistent, FFDC provides an interface to report this identifier to standard error: **fcdispfid**. Commands and scripts invoke this interface to report the FFDC Failure Identifier to standard error. The interface will display the 42-character token using message number 2615-000. The application or script is then *strongly advised* to set a nonzero exit status before terminating to indicate to the caller that the command or script completed in error. Callers of the command or script can then detect the failure by examining the exit status and retrieve the FFDC Failure Identifier token by capturing standard error and searching the information for this specific message number and stripping the message number from the item.

Recording Information To the FFDC Error Stack

When issuing the First Failure Data Capture utility to record information to the FFDC Error Stack, the software must provide a description of the failure condition and the details about the failure condition. The

description is similar in purpose to the Error Description (Err_Desc) that would be used in an AIX Error Log template: it describes the nature of the failure condition. The detailed information is the same as the Detail Data that would be placed in the AIX Error Log.

The First Failure Data Capture utilities will supply information to identify the calling process and thread within the report. A unique *FFDC Error Identifier* is also recorded in the text of the failure description, and returned to the caller. The First Failure Data Capture utility user is expected to capture this information and to provide it as part of the failure information to its own client, so that the client can locate the failure report generated for this condition, and to possibly reference this failure report in its own failure report.

Services invoked by the software may provide the software with an FFDC Error Identifier as part of the service's failure information. If the service's failure will also cause a failure in the software itself, the software must also make a report of the failure, reporting the failure condition in terms of the failure's impact to its own function. In this report, the software must make a reference to the service's failure, to establish the common link between this software's failure and its source in the service's failure. This is done by providing the service's FFDC Error Identifier as part of the failure information to the FFDC interface. When this is done, the FFDC utilities will record the service's failure token as a pointer to this failure's cause. This will allow problem investigators to proceed to the cause of the failure, instead of guessing at the cause of the software's failure at this point.

The Programming Model

The routines that software uses to record information to the FFDC Error Stack are introduced in "Chapter 4. First Failure Data Capture Utilities" on page 41. The **fcinit** and **fcpushstk** interfaces allow software to use the FFDC Error Stack.

The **fcinit** interface allows software to explicitly create the FFDC Environment so that an FFDC Error Stack exists. **fcinit** also sets up software to make use of an FFDC Error Stack only if one of the ancestors of the software previously set up an FFDC Environment; this is called *inheriting*. The version of **fcinit** that is used is handled by specific options and parameters to the interface. **fcinit** is executed once by the application, and if it is executed from a threaded process, it must be executed before any threads are created. If an application requests to create an FFDC Environment when one already exists, the application inherits the existing environment.

The **fcpushstk** interface allows software to record information to the FFDC Error Stack. The software using this interface does not need to worry about checking for an active FFDC Error Stack or an FFDC Environment. **fcpushstk** detects whether or not an FFDC Environment exists and only records information if the environment does exist; if the environment does not exist, the interface does not report an error back to the client so that the client can continue in a normal fashion.

Chapter 4. First Failure Data Capture Utilities

The FFDC interfaces are provided in both command line interface (CLI) and application programming interface (API) format. FFDC is provided as part of the RSCT software, shipped as part of the **rsct.core.utils** file set. Software making use of these utilities must ensure that the **rsct.core.utils** file set is required by their software packaging files.

First Failure Data Capture commands are installed on each node. The commands are made available in the **/usr/bin** directory, though they are linked to this directory from their initial installation location of **/usr/sbin/rsct/bin**. The FFDC API is available through the **libct_ffdc.a** shared library and the **/usr/include/rsct/ct_ffdc.h** header file. The API supports both 32-bit and 64-bit operating environments.

Programming Interfaces

In the following sections, the names of the command line interfaces and their associated application programming interfaces are listed together. The description of these interfaces uses the command line interface name for simplicity. Any statements made about the function or requirements of the command line interface should be assumed by the reader to be true as well for the equivalent application programming interface.

fcinit/fc_init

This interface is called in order to use the FFDC utilities for recording failure information to persistent storage. Applications use this interface for one of the following reasons:

- An application may wish to record information to the AIX Error Log. Applications can use **fcinit** to establish a basic FFDC Environment.
- An application that considers itself to be a top-level application wants to have itself and any subroutines or descendant processes created by itself or its children to record failure information to the FFDC Error Stack. The application causes multiple lower-level applications to be created, and the success of the top-level application depends upon the success of these lower-level applications. When **fcinit** is used in this fashion, the process is said to *establish* or *create* the FFDC Error Stack Environment.
- The application uses the FFDC Error Stack only when the application is invoked by an ancestor process that wants failure information recorded to these devices. In all other cases, the application does not wish to use these devices. When **fcinit** is used in this fashion, the process is said to *inherit* the FFDC Error Stack Environment.

An *FFDC Error Stack Environment* is established by a process when that process wants to have failure information from itself, any subroutines or library routines it invokes, any threads it may create, and any descendant processes it may create to be recorded in an FFDC Error Stack. An *FFDC Error Stack Environment* is inherited by a process when that process wants to record failure information to an FFDC Error Stack file only when one of its ancestors has requested for processes to do so; in all other cases, the process does not record failure information to the FFDC Error Stack. Processes use **fcinit** to either establish or inherit the FFDC Error Stack Environment.

The FFDC Error Stack Environment reserves an FFDC Error Stack file, so that failure information is recorded to a file in the **/var/adm/ffdc/stacks** directory, named **script_name.PID.date**, where *script_name* is the name of the script itself, *PID* is the process identifier of the script, and *date* is the date and time when the script was executed. Whenever this script or children processes of this script record failure information to the FFDC Error Stack, it is recorded in this file.

A process must use the **fcpushstk** FFDC interface in order to record information in the FFDC Error Stack file, and the process also has to be operating within an established FFDC Error Stack Environment; otherwise, no information is recorded by that process in the FFDC Error Stack. This permits processes to

run in a normal or "silent" mode when failure debugging information is not wanted or needed but makes failure debugging information available when the process is invoked within a special environment for debugging.

Processes that use the **fclogerr** FFDC interface can use **fclogerr** independent of the existence of an *FFDC Environment*. Whenever **fclogerr** is used, failure information is recorded to the AIX Error Log and the BSD System Log, regardless of whether an *FFDC Environment* has been established. However, **fcinit** must be used by the application to set up process environment values required by the **fclogerr** interfaces.

If an FFDC Environment already exists when a process attempts to create one, the process inherits the existing FFDC Environment instead.

fcteststk/fc_test_stack

This interface is used by an application to test if an *FFDC Error Stack Environment* has been established for the application. This interface is useful for library routines that are not responsible for establishing an *FFDC Error Stack Environment*, but would record to an FFDC Error Stack if an *FFDC Error Stack Environment* existed. When a failure condition is detected, a library routine or application can issue **fcteststk** before attempting to collect information needed for an FFDC Error Stack entry. If **fcteststk** indicates that an *FFDC Error Stack Environment* has not been established, the application can avoid collecting failure information, because an FFDC Error Stack report is not made in this case anyway.

fcteststk is *not* intended for applications and library routines that record failure information to the AIX Error Log. These applications must assume that the AIX Error Log is always active and should collect failure information for any reports to be made to the AIX Error Log.

fclogerr/fc_log_error

This interface is used by an application program to record information to the AIX Error Log and the BSD System Log. Information is written to this device for use by the system administrator or operator to determine whether failure conditions or other noteworthy conditions that have occurred on the system require attention. Enough information is recorded in the AIX Error Log and the BSD System Log about a condition to determine the nature, impact, and response to the condition from the report without requiring the condition to be re-created. Any software that encounters permanent failure conditions that will persist until some type of direct intervention occurs, or encounters a condition that should be brought to the attention of the system administrator, should use **fclogerr** to record this information in the AIX Error Log and the BSD System Log.

Applications should establish a basic FFDC Environment or an FFDC Error Stack Environment before calling **fc_log_error**, either by creating or inheriting the environment. **fc_log_error** records information to the AIX Error Log and the BSD System Log even if these environments are not established, but the interface will not be capable of generating an FFDC Failure Identifier unless one of these environments exists.

Processes that use the FFDC Error Stack can also use the **fclogerr** interface and should use it if they encounter conditions that require administrator attention or intervention to resolve.

To ensure that a condition and the location at which it was encountered are properly identified, **fclogerr** should be called in-line in the source code module and invoked as soon as the condition is detected. **fclogerr** will record source code file name and line of code information to assist in identifying and locating the source code that encountered the condition. If **fclogerr** is not invoked directly, but indirectly through a subroutine call, this information must be passed to the subroutine. If the subroutine is not provided with this information, the failure report will not indicate the location within the code that the problem was discovered.

Although **fclogerr** reports information to both the AIX Error Log and the BSD System Log, different parameters must be provided to this interface for each recording device. The Detail Data information

recorded to the AIX Error Log is not recorded to the BSD System Log; BSD System Log information is provided in a different set of parameters. This may require the **fclogerr** user to duplicate some information in this call.

fclogerr expects the AIX Error Log template to have reserved sufficient space to record the name of the source code module experiencing the failure, the version of this file, the line-of-code position, the LPP name, the *FFDC Failure Identifier* for the report, and an optional identifier from a previously recorded report. Because of this, **fclogerr** requires any template used by the caller to reserve at least four Detail Data fields in the template. Three fields are used by **fclogerr**, leaving the fourth through twelfth fields to the caller for cataloging debug information about the failure:

```
Detail_Data = 46, 00A2, ALPHA
Detail_Data = 42, EB2B, ALPHA
Detail_Data = 42, 0030, ALPHA
```

The first Detail Data field is 46 characters long and is labeled DETECTING MODULE. It contains the source-code file name, its SCCS SID version, the line-of-code position, and the LPP abbreviation. The second field is labeled ERROR ID and contains the *FFDC Failure Identifier* generated by the **fclogerr** interface. This is the same FFDC Failure Identifier that **fclogerr** will return to its caller. The third field is labeled REFERENCE CODE and contains an optional FFDC Failure Identifier that was reported previously by other software; this field establishes the link between this report and the associated failure condition in other software that caused it. The remaining nine Detail Data fields and 100 bytes of available space can be used as is best suited to report debugging information for this failure condition.

Consult the chapter "Constructing Persistent Records of Failures" [LINK](#) to understand how to construct meaningful AIX Error Log templates and determine what information is required in each template.

fcpushstk/fc_push_stack

This interface is used by any application program that wishes to record information to the FFDC Error Stack. The information written to this device is intended for use as debugging information. The purpose of the FFDC Error Stack is to understand the failure conditions that occur within multiple layers of related software, and to understand how these failure conditions impact those software components dependent upon the software that failed. The FFDC Error Stack is best suited for use by an application that spawns one or more child processes or threads, which in turn may spawn more descendant processes. The FFDC Error Stack can also be used by applications that use multiple SP-developed libraries in a common task, even if the application is a standalone process. In these situations, the application can *establish* an FFDC Error Stack Environment using the **fcinit** interface. Once established, the application and any of its subroutines or descendant processes can make use of the FFDC Error Stack.

Not all software applications execute in such an environment under normal circumstances. However, these applications may be invoked in such an environment by other scripts or applications, such as those depicted in the examples in the previous chapter. In these cases, the scripts or applications invoking this software may wish to capture the failure information from this software, to analyze it along with other failure information from other software that is invoked, to discover any relationships or patterns in the failures. For this reason, software that ordinarily would not make use of the FFDC Error Stack under normal operational conditions should at least support the use of the FFDC Error Stack when it is used by any client invoking the software. This is accomplished by *inheriting* the FFDC Error Stack Environment from the parent process through the **fcinit** interface. Library routines need not inherit the FFDC Error Stack Environment explicitly; this task is performed by the application using the library.

Software services that execute as device-driver kernel extensions are not suited to using the FFDC Error Stack. Such software applications should use the AIX Error Log and the BSD System Log to record information about failure and noteworthy conditions.

fcpushstk is used by an application to record information about failures and noteworthy conditions to the FFDC Error Stack. In order for information to be recorded in the FFDC Error Stack by a process, the process has to be operating within an established *FFDC Error Stack Environment*. If an FFDC Error Stack

Environment does not exist, or if the **fcpushstk** interface is not used when an FFDC Error Stack Environment exists, no information is recorded by that process in the FFDC Error Stack. This permits processes to run in a normal or "silent" mode when failure debugging information is not wanted or needed but makes failure debugging information available when the process is invoked within a special environment for debugging.

To ensure that the condition and the location at which it was encountered are properly identified, **fcpushstk** should be called in-line in the source code module and invoked as soon as the condition is detected. **fcpushstk** will record source-code file-name and line-of-code information to assist in identifying and locating the source code that encountered the condition. If **fcpushstk** is not invoked directly, but indirectly through a subroutine call, this information must be passed to the subroutine. If the subroutine is not provided with this information, the failure report will not indicate the location within the code that the problem was discovered.

fcdispfid/fc_display_fid

This interface is used by an application to display an *FFDC Error Identifier* to the standard error device. FFDC provides an interface for this to ensure that all applications present this information in a consistent manner, so that any software can expect this information in a constant location and format.

fcfilter

This interface is used by a script to obtain an *FFDC Error Identifier* from the standard error of a child application or script.

End-User Commands

The following commands are provided for the end user to examine and maintain the information recorded by the First Failure Data Capture utilities.

fcdecode

This command accepts an *FFDC Error Identifier* as an argument. This identifier is generated by the **fclogerr** command and **fcpushstk** commands and by the **fc_log_error** and **fc_push_stack** application programming interfaces (APIs). **fcdecode** interprets the value provided and decodes the identifier into a more understandable format.

From the FFDC Error Identifier, **fcdecode** extracts the identity of the node where the failure is recorded, the time when the failure record was recorded, and the AIX Error Logging template identifier used (if any) to record the failure. This data is displayed to the standard output device. This permits the user or a script to decode the failure identifier generated by the FFDC utilities into options that can be passed to the **telnet**, **dsh**, and **errprt** commands. The script using the **fcdecode** command is expected to redirect standard output to either a file or a variable to capture this information, so that the data can be returned to the client of the script or be used in **dsh** or **errprt** commands.

fcreport

This command generates a list of the AIX Error Log and FFDC Error Stack recording associated with a specific failure. When an FFDC Error Identifier is provided, **fcreport** attempts to locate this error report and any other error reports associated with it. **fcreport** uses the associated FFDC Error Identifier from the initial report to locate the next failure report, pulls the associated FFDC Error Identifier from that report, obtains the next associated failure report, and so on until no more associated reports can be found. The full report is written to the standard output device.

fcstkrpt

This command generates a report of existing FFDC Error Stack files. The default action of the command is to provide a listing of available FFDC Error Stack files on the node, indicating the file name, the command

that created the file, the process identifier of the command, and the date and time at which the command was executed. Options are provided for the command to locate specific FFDC Error Stack files and display the contents of files that meet the selection criteria to the standard output device.

The **fcstkrpt** command provides options and arguments to select specific FFDC Error Stack files based on specific criteria. When specific files are located that match these criteria, the contents of the file are displayed to the standard output device. The file contents are presented in the order in which processes and threads were invoked.

fcclear

This command deletes FFDC Error Stack files from the node on which it executes. All FFDC Error Stack files that meet the selection criteria are deleted from the node. A selection criteria must be provided.

fcclear is modeled loosely after the **errclear** command, and, as with **errclear**, root authority is necessary to issue this command.

fccheck

This command performs basic problem determination on the First Failure Data Capture utilities and displays the results of these tests to standard output, unless the "quiet" option has been specified. The **fccheck** command sets an exist status value to indicate the most severe condition it detected during the execution of its tests.

FFDC Utilities Manual Pages

This section describes the manual pages for the FFDC header file, the programming interfaces, and the commands.

<rsct/ct_ffdc.h> Header File

Purpose

`/usr/include/rsct/ct_ffdc.h` - Provides data types, definitions, and interface prototypes for the First Failure Data Capture C language library interfaces.

Description

This header file must be included by any C and C++ language source code files that make use of the First Failure Data Capture C language interfaces. Contained in this file are the C language prototypes for the First Failure Data Capture interfaces, the symbolic constants used as return codes from these interfaces, and data type definitions needed by First Failure Data Capture C and C++ language clients.

C Language Interface Selection Control: This file provides the compiler definition **FC_VERSION**. This definition controls which version of the First Failure Data Capture interfaces should be used during compilation of the source code. Currently, only one version of the First Failure Data Capture interfaces are available, and the value of **FC_VERSION** is set to a default value of 1. Future versions of the First Failure Data Capture interfaces can be used during compilation—when they become available—by setting the value for **FC_VERSION** on the compilation command line. If this variable is not set during compilation, the value for **FC_VERSION** reverts to the default value of 1, and the initial version of the FFDC interfaces is used.

Data Types: The **fc_eid_t** data type defined by this module is used to store a First Failure Data Capture Failure Identifier. This identifier is created by the **fc_push_stack** and **fc_log_error** interfaces whenever these interfaces are successful in recording failure information. This identifier contains information in an encoded form to indicate the system on which the record was made, the time when the record was made, and the location of the record. First Failure Data Capture commands such as **fcreport** and **fcstkrpt** can be used at a later time to obtain the exact failure report for problem determination efforts.

FFDC Environment Establishment Codes: A First Failure Data Capture client application uses the **fc_init** interface to specify how the FFDC Environment should be established. The following selections are supported:

FC_LOG

A basic FFDC Environment is established, which permits the application to record failure information to the AIX Error Log and the BSD System Log. An FFDC Error Stack is not established for use by the application in this case, unless this value is combined with either the **FC_STACK_CREAT** or the **FC_STACK_INHERIT** options described below. This selection would be used by applications making use of the **fc_log_error** interface only.

FC_STACK_CREAT

Creates an FFDC Error Stack Environment if one was not previously created by an ancestor of this process, or inherits the FFDC Error Stack Environment if an ancestor previously established one. The FFDC Error Stack Environment permits the application to record information to the AIX Error Log, the BSD System Log, and the FFDC Error Stack. This selection is used by applications that wish to use the **fc_push_stack** interface as well as the **fc_log_error** interface. This selection is not to be combined with the **FC_STACK_INHERIT** option described next.

FC_STACK_INHERIT

Inherit an FFDC Error Stack Environment only if an ancestor process previously established an FFDC Error Stack Environment. If an ancestor did not establish such an environment, the application does not make use of an FFDC Error Stack, but may still make use of the AIX Error Log and the BSD System Log. Do not combine this selection with the **FC_STACK_CREAT** option specified previously.

Record Type Definitions: Seven FFDC Event Types are specified in this file. These event types are used to instruct the **fc_log_error** interface as to the severity of the condition being logged:

FFDC_EMERG

A severe failure has occurred, and the system is in danger of coming offline. This information is required by the system administrator to bring the system back online. The AIX Error Log type equivalent is PEND. The BSD System Log priority equivalent is LOG_EMERG.

FFDC_ERROR

A permanent failure has occurred, and the condition will persist until it is repaired. The system is not in danger of coming offline. The AIX Error Log type equivalent is PERM. The BSD System Log priority equivalent is LOG_ERR.

FFDC_STATE

An event of some significance has occurred, but the event does not constitute a failure condition. The AIX Error Log type equivalent is INFO. The BSD System Log priority equivalent is LOG_NOTICE.

FFDC_PERF

A condition has been noticed which can or will degrade the system's performance below acceptable levels. The system is not in danger of coming offline, but performance may be unacceptably slow, which can result in random failures in system applications, such as timeout conditions. The AIX Error Log type equivalent is PERF. The BSD System Log priority equivalent is LOG_WARNING.

FFDC_TRACE

This entry identifies the name and location of a trace file generated by an application or system component. Such an entry would be made when a trace has been enabled in an application or a component, to indicate where the trace file resides. The AIX Error Log type equivalent is UNKN. The BSD System Log priority equivalent is LOG_INFO.

FFDC_RECOV

A recovery action has been successfully completed by the system in response to an FFDC_EMERG, FFDC_ERROR, or FFDC_PERF condition. Such an entry would be created only after an FFDC_EMERG, FFDC_ERROR, or FFDC_PERF condition was detected, and a recovery action started in response to that condition completed successfully. The AIX Error Log type equivalent is TEMP. The BSD System Log priority equivalent is LOG_DEBUG.

FFDC_DEBUG

A failure condition was detected. Unlike the FFDC_ERROR case, the failure is not a permanent condition, or the system can continue successfully with the condition present. The AIX Error Log type equivalent is UNKN. The BSD System Log priority equivalent is LOG_DEBUG.

Library

/usr/include/rsct/ct_ffdc.h

/usr/sbin/rsct/include/ct_ffdc.h

Related Information

“fc_init Subroutine” on page 48, “fc_log_error Subroutine” on page 66, “fc_push_stack Subroutine” on page 59, “fc_display_fid Subroutine” on page 55, “fc_eid_init Subroutine” on page 52, “fc_eid_is_set Subroutine” on page 53, “fcdispfid Command” on page 95, “fcdecode Command” on page 97, “fcreport Command” on page 103, “fcstkrpt Command” on page 101

Examples

To make use of this file in a C or C++ language program, use the following line near the beginning of the source code module:

```
#include <rsct/ct_ffcd.h>
```

fc_init Subroutine

Purpose

Establishes a First Failure Data Capture execution environment

Library

First Failure Data Capture programming library (**libct_ffdc.a**)

Syntax

This subroutine is provided in the **<rsct/ct_ffdc.h>** header file.

```
#include <rsct/ct_ffdc.h>
int fc_init(int option_code)
```

Parameters

option_code

Is specified by specifying one or more symbolic values defined in **<rsct/ct_ffdc.h>**. These values can be ORed together where they make sense.

FC_LOG

Establish an FFDC Environment to make use of the AIX Error Log and the BSD System Log (a basic FFDC Environment). When *option_code* is set to this value only, a basic FFDC Environment is established without an FFDC Error Stack. This *option_code* would be used by applications wishing to use the **fc_log_error** interface only.

FC_STACK_CREAT

Create an FFDC Environment that makes use of the AIX Error Log, the BSD System Log, and the FFDC Error Stack. If an FFDC Environment containing an FFDC Error Stack already exists, this environment is inherited (the same as if the **FC_STACK_INHERIT** option had been specified). Use this option in applications that wish to use the **fc_push_stack** and possibly the **fc_log_error** interface. Do not combine this option with the **FC_STACK_INHERIT** option.

FC_STACK_INHERIT

Inherit an FFDC Environment that makes use of the AIX Error Log, the BSD System Log, and the FFDC Error Stack. If an FFDC Environment contain an FFDC Error Stack exists, it is inherited; otherwise, an FFDC Environment that uses the AIX Error Log and the BSD System Log only is established. Use this option in applications that wish to use the **fc_push_stack** interface only when an FFDC Error Stack has been established by an ancestor process. Do not combine this option with the **FC_STACK_CREAT** option.

Description

This interface must be called by any application program that wishes to use the FFDC library interfaces for recording information to the AIX Error Log, the BSD System Log, and the FFDC Error Stack .

Applications may wish to establish an FFDC Environment containing an FFDC Error Stack for one of the following reasons:

- The application may wish to record information to the AIX Error Log. Applications can use **fc_init** to establish a basic FFDC environment.
- The application wants to have itself and any descendant processes created by itself or its children to record failure information to the FFDC Error Stack. In this case, the application considers itself a "top-level" application that will cause multiple "lower-level" applications to be created, and the success of the "top-level" application depends upon the success of these "lower-level" applications. When using **fc_init** in this fashion, the process is said to *establish* or *create* the FFDC Error Stack Environment.

- The application uses the FFDC Error Stack only in those cases when the application is invoked by an ancestor process that wants failure information recorded to these devices. In all other cases, the application does not wish to use these devices. When using **fc_init** in this fashion, the process is said to *inherit* the FFDC Error Stack Environment.

Any process wishing to record information to the AIX Error Log or the BSD System Log through the FFDC interfaces must establish an *FFDC Environment*. If the process does not wish to make use of an FFDC Error Stack, a *basic FFDC Environment* that does not make use of an FFDC Error Stack can be established. An FFDC Environment containing an FFDC Error Stack is established by a process when that process wants to have failure information from itself, any threads it may create, and any descendant processes it may create to be recorded in an FFDC Error Stack. An FFDC Environment that contains an FFDC Error Stack is inherited by a process when that process wants to record failure information to an FFDC Error Stack file only when one of its ancestors has requested for processes to do so; in all other cases, the process will not record failure information to the FFDC Error Stack. When an FFDC Error Stack is used, the environment is said to be an *FFDC Error Stack Environment*.

The FFDC Environment that contains an FFDC Error Stack reserves an FFDC Error Stack file, so that failure information is recorded to a file in the **/var/adm/ffdc/stacks** directory. These files use the naming format **script_name.PID.date_and_time**, where *script_name* is the name of the script itself, *PID* is the process identifier of the script, and *date_and_time* is the date and time when the script was executed. Whenever this script or children processes of this script record failure information to the FFDC Error Stack, it will be recorded in this file.

In order for information to be recorded in the FFDC Error Stack by a process, the process must use the **fc_push_stack** FFDC interface, and the process has to be operating within an established FFDC Error Stack Environment. If an FFDC Error Stack Environment does not exist, or if the **fc_push_stack** interface is not used when an FFDC Error Stack Environment exists, no information is recorded by that process in the FFDC Error Stack. This function permits processes to run in a normal or "silent" mode when failure debugging information is not wanted or needed, but also permits this information to be available when the process is invoked within a special environment for debugging.

Processes that use the **fc_log_error** FFDC interface should establish a *basic FFDC Environment*. If the process only wishes to use the **fc_log_error** interface, the basic FFDC Environment can be established without an FFDC Error Stack. If a basic FFDC Environment is not established by a process, the process may still record failure information to the AIX Error Log using the **fc_log_error** interface. In these cases an *FFDC Failure Identifier* will not be generated for the report recorded to the AIX Error Log.

If an FFDC Environment already exists when a process attempts to create one, the process inherits the existing FFDC Environment instead of creating its own.

Return Values

fc_init returns the following integer codes upon completion:

FC_SUCCESS

FFDC Environment successfully established.

FC_INHERIT_SUCCESS

FFDC Environment successfully inherited.

FC_NO_FC_ENVIR

FFDC Environment not established or inherited. An FFDC Environment did not exist, and the FC_INHERIT option was specified.

fc_init returns the following integer codes upon detection of a failure:

FC_STACK_DISABLED

FFDC Error Stack Environment not established or inherited. Creation and use of FFDC Error Stacks has been disabled by the system administrator. Applications can establish basic FFDC Environments that make use of the AIX Error Log and the BSD System Log only.

FC_UNKNOWN_OPTION

FFDC Environment not established or inherited. Unknown function parameter provided.

FC_STACK_DIRECTORY

FFDC Error Stack Environment not established or inherited. Unable to reserve the FFDC Error Stack file for the calling process; the FFDC Error Stack directory does not exist or cannot be used.

FC_STACK_RESERVE

FFDC Error Stack Environment not established or inherited. Unable to reserve the FFDC Error Stack file for the calling process; the file already exists

FC_INHERIT_INV

FFDC Environment not established or inherited. Caller indicated that the FFDC Environment should be both created and inherited.

FC_ENV_EXIST

FFDC Environment not established in this call. The caller already has an FFDC Environment established for itself; this routine may have been executed multiple times.

FC_ENV_CORRUPT

FFDC Environment not established or inherited. The FFDC Environment appears to be corrupted and should be considered unusable.

FC_NO_MEMORY

FFDC Environment not established or inherited. The routine could not allocate the memory required to modify the client's process environment.

FC_ENV_FAILED

FFDC Environment not established or inherited. The client's process environment could not be modified by this routine.

FC_INTERNAL

FFDC Environment not established or inherited. An unexpected internal failure occurred within **fc_init**. This condition may require the attention of customer and application-support services.

Examples

To establish an FFDC Error Stack Environment that causes this process and any descendant process to record failure information to the FFDC Error Stack:

```
rc = fc_init(FC_STACK_CREAT);
switch (rc) {
    case FC_SUCCESS:           break;
    case FC_INHERIT_SUCCESS:   printf("Inheriting Environment\n");
                              break;
    case FC_STACK_DISABLED:    printf("Error Stacks not available\n");
                              break;
    case FC_UNKNOWN_OPTIONS:   printf("fc_init used with unknown options\n");
                              return(-1);
    :
    :
}
```

Note:

The FFDC client may receive an indication that an FFDC Error Stack Environment was inherited, instead of created by the **fc_init** call. This occurs when an FFDC Error Stack Environment was already established by one of the process's ancestors.

To inherit an FFDC Error Stack Environment from the process's parent process:

```
rc = fc_init(FC_STACK_INHERIT);
switch (rc) {
    case FC_INHERIT_SUCCESS:    printf("Inheriting Environment\n");
                              break;
    case FC_STACK_DISABLED:    printf("Error Stacks not available\n");
                              break;
    case FC_UNKNOWN_OPTIONS:   printf("fc_init_stack used with unknown option;
                              return(-1);
                              :
                              :
}
```

To establish a basic FFDC Environment that makes use of the AIX Error Log and the BSD System Log only (the process does not make use of an FFDC Error Stack):

```
rc = fc_init(FC_LOG);
switch (rc) {
    case FC_SUCCESS:          printf("Environment established\n");
                              break;
    case FC_UNKNOWN_OPTIONS:  printf("fc_init_stack used with unknown options\n");
                              return(-1);
                              :
                              :
}
```

Implementation Specifics

This subroutine is provided as part of the First Failure Data Capture option of RS Cluster Technology (RSCT).

Related Information

“fc_log_error Subroutine” on page 66, “fc_push_stack Subroutine” on page 59, “fc_test_stack Subroutine” on page 57,, “fcinit Command” on page 74, “fccheck Command” on page 108

fc_eid_init Subroutine

Purpose

Initializes or clears the contents of an FFDC Failure Identifier variable.

Library

First Failure Data Capture programming library (**libct_ffdc.a**)

Syntax

This subroutine is provided in the **<rsct/ct_ffdc.h>** header file.

```
#include <rsct/ct_ffdc.h>
int fc_eid_init(fc_eid_t fid)
```

Parameters

fid An *fc_eid_t* variable previously defined by the application. The actual variable is provided to the macro, not a pointer to the variable.

Description

This subroutine can be used by an application to initialize an FFDC Failure Identifier variable before passing it to the FFDC interfaces. The subroutine can also be used to clear out the contents of a previously used FFDC Failure Identifier value.

Examples

To initialize an FFDC Failure Identifier value before passing it to an FFDC interface:

```
fc_eid_t fid;
:
fc_eid_init(fid);
```

Implementation Specifics

This subroutine is provided as part of the First Failure Data Capture option of RS Cluster Technology (RSCT).

Related Information

"fc_eid_is_set Subroutine" on page 53

fc_eid_is_set Subroutine

Purpose

Tests if an FFDC Failure Identifier variable contains an FFDC Failure Identifier value.

Library

First Failure Data Capture programming library (**libct_ffdc.a**)

Syntax

This subroutine is provided in the **<rsct/ct_ffdc.h>** header file.

```
#include <rsct/ct_ffdc.h>
int = fc_eid_is_set(fc_eid_t fid);
```

Parameters

fid

A variable of type `fc_eid_t`, previously defined by the application, and used as a parameter to a previous call to the **fc_log_error** or **fc_push_stack** interfaces.

Description

This subroutine allows an application to determine if an FFDC Failure Identifier variable has been set by a previously called FFDC interface. The application must have first initialized or cleared the FFDC Failure Identifier variable with the **fc_eid_init** subroutine for this test to function properly. An application uses this subroutine to determine if a call to the **fc_display_fid** subroutine is necessary to display the FFDC Failure Identifier, or if an identifier exists to pass back to the application's client.

Return Values

0

The FFDC Failure Identifier variable has not been set by a previously executed FFDC interface.

1

The FFDC Failure Identifier variable has been set.

Examples

To determine if an FFDC Failure Identifier was successfully created for a recording made to the AIX Error Log before attempting to return it:

```
rc = fc_log_error(&fid, aid, "myprog", "PSSP", __FILE__, "%I%",
                __LINE__, (void*)NULL, FFDC_ERROR, ERRID_SP_FFDCXEMPL_ER,
                failure_details, 100, (char*)NULL,
                "Configuration Failure - Exiting");
if (FC_SUCCESS== rc) {
    fid_to_client = fid;
    close(filides);
    return(-1);
}
if (FC_SYSLOG_FAILED == rc) {
    if (fc_eid_is_set(fid)) {
        fid_to_client = fid;
    }
    return(-1);
}
```

Implementation Specifics

This subroutine is provided as part of the First Failure Data Capture option of RS Cluster Technology (RSCT).

Related Information

"fc_display_fid Subroutine" on page 55, "fc_eid_init Subroutine" on page 52

fc_display_fid Subroutine

Purpose

Displays the First Failure Data Capture Failure Identifier (FFDC Failure Identifier) to the standard error device.

Library

First Failure Data Capture programming library (**libct_ffdc.a**)

Syntax

```
#include <rsct/ct_ffdc.h>
int = fc_display_fid(fc_eid_t fid);
```

Parameters

fid

Specifies an FFDC Failure Identifier. This is an identifier returned from a previous call to **fc_push_stack** or **fc_log_error**, and indicates an entry made to report a failure encountered by the application. This identifier is written to the standard error device using an FFDC cataloged message.

Description

fc_display_fid is used by applications to display an FFDC Failure Identifier value to the standard error device. This interface is provided for applications that do not have a mechanism for passing data back to its client except through exit status codes, signals, standard output, and standard error (commands and scripts are typical examples of such applications). To accomplish the task of passing back an FFDC Failure Identifier from such an application to its client, **fc_display_fid** uses XPG/4 message number **2615-000** to display this information to the standard error device. Clients of the application can capture the standard error information, search for the specific message number, and obtain the FFDC Failure Identifier from the application.

Applications that use **fc_display_fid** must indicate that any FFDC Failure Identifiers generated by the application will be directed to the standard error device in the application's user documentation. The client cannot be expected to know this behavior by default.

If no FFDC Failure Identifier is provided, no output is directed to the standard error device.

Return Values

fc_display_fid returns the following integer codes upon completion:

FC_SUCCESS

FFDC Failure Identifier displayed to standard error.

FC_EID_INV

No information written to the standard error device. The *fid* parameter does not appear to be in the proper format.

Examples

To display an FFDC Failure Identifier to the client through the standard output device:

```
rc = fc_log_error(FFDC_ERROR, ERRID_SP_FFDCXEMPL_ER, "myprog",
                failure_details, 102, (char *) NULL, __FILE__, __LINE__, "%I%",
                "PSSP", aid, &fid, "Configuration Failure - Exiting");
if (FC_SUCCESS == rc) {
    rc = fc_display_fid(fid);
    return(-1);
}
```

Implementation Specifics

This subroutine is provided as part of the First Failure Data Capture option of RS Cluster Technology (RSCT).

Related Information

“fc_log_error Subroutine” on page 66, “fc_push_stack Subroutine” on page 59, “fc_eid_is_set Subroutine” on page 53, “fcdispfid Command” on page 95

fc_test_stack Subroutine

Purpose

Test for the presence of a First Failure Data Capture Error Stack Environment.

Library

First Failure Data Capture programming library (**libct_ffdc.a**)

Syntax

```
#include <rsct/ct_ffdc.h>
int fc_test_stack(void)
```

Description

This interface can be called by any application program that wishes to use the FFDC Error Stack to test whether an FFDC Error Stack Environment has been established for the process . By performing this test, applications can avoid the performance burden of collecting failure information in cases where an FFDC Error Stack Environment has not been established. This interface is provided primarily for use by library routines, which would not have any knowledge of whether their client application established or inherited an FFDC Error Stack Environment.

An FFDC Error Stack Environment is established by a process when that process wants to have failure information from itself, any threads it may create, and any descendant processes it may create to be recorded in an FFDC Error Stack. An FFDC Error Stack Environment is inherited by a process when that process wants to record failure information to an FFDC Error Stack file only when one of its ancestors has requested for processes to do so; in all other cases, the process will not record failure information to the FFDC Error Stack. Processes use **fc_init_stack** to either establish or inherit the FFDC Error Stack Environment.

The FFDC Error Stack Environment reserves an FFDC Error Stack file, so that failure information is recorded to a file in the **/var/adm/ffdc/stacks** directory. These files use the naming format **script_name.PID.date_and_time**, where *script_name* is the name of the script itself, *PID* is the process identifier of the script, and *date_and_time* is the date and time when the script was executed. Whenever this script or children processes of this script record failure information to the FFDC Error Stack, it will be recorded in this file.

Applications use the **fc_push_stack** interface to record failure information to the FFDC Error Stack. However, the application may need to collect this information from various locations before recording the information, and obtaining this information can impact the application's overall performance. The application should not need to collect this information if the FFDC Error Stack Environment was not established or inherited. To avoid this performance impact, the application can issue **fc_test_stack** to determine if an FFDC Error Stack Environment is available, and if so, begin collecting the failure information. If the FFDC Error Stack Environment does not exist, the application can avoid collecting this information.

Any application that records information using the **fc_log_error** interface must *always* collect the failure information and record it, regardless of whether an FFDC Error Stack is in use.

Return Values

fc_test_stack returns the following integer codes upon completion:

FC_SUCCESS

An FFDC Error Stack Environment exists that uses an FFDC Error Stack.

FC_NO_FC_ENVIR

FFDC Error Stack Environment has not been established or inherited by the client at this instant.

FC_ENV_CORRUPT

FFDC Error Stack Environment appears to be corrupted and should be considered unusable.

Examples

To test whether an FFDC Error Stack Environment exists for an application:

```
rc = fc_test_stack();
switch (rc) {
    case FC_SUCCESS:      /* Collect failure information */
                        :
                        :
                        /* Record Info via fc_push_stack */
                        :
                        :
                        break;
    case FC_NO_FC_ENVIR:  break;
                        break;
                        break;
}
```

Implementation Specifics

This subroutine is provided as part of the First Failure Data Capture option of RS Cluster Technology (RSCT).

Related Information

“fc_init Subroutine” on page 48, “fc_push_stack Subroutine” on page 59, “fcteststk Command” on page 78

fc_push_stack Subroutine

Purpose

Records information about failure or noteworthy conditions to the First Failure Data Capture Error Stack.

Library

First Failure Data Capture programming library (**libct_ffdc.a**)

Syntax

```
#include <rsct/ct_ffdc.h>
int fc_push_stack(fc_eid_t *fid, fc_eid_t assoc_fid, char *resource,
                 char *lpp_name, char *source_filename, char *sidlevel,
                 int line_of_code_pos, (void *) NULL, char *detail_data,
                 char *detail_data_file, char *catalog_name, int msg_set,
                 int msg_number, char *default_message, ...)
```

Parameters

fid

Pointer to a `fc_eid_t` data area previously defined or allocated by the caller. **fc_push_stack** returns the FFDC Failure Identifier for the record it made in the FFDC Error Stack through this parameter. This data area is not modified by this routine unless an FFDC Failure Identifier was successfully created by the routine.

assoc_fid

Contains the FFDC Failure Identifier for a failure condition reported by software used by this application which causes or influenced the condition being recorded at this time. This identifier should have been returned to this application as part of the software's result indication. The caller provides this identifier here so that the FFDC Error Stack can associate the failure report it is making at this time with the previously recorded failure report. This permits problem investigators to trace the cause of a failure from its various symptoms in this application and others to the root cause in the other software. If no other software failure is responsible for this condition, or if the other software did not return an FFDC Failure Identifier as part of its result information, this application should set this parameter to **(fc_eid_t) NULL**.

resource

Specifies the software component name. This is a symbolic name for the software making the report, and should be a name recognizable to both customer and application-support services.

lpp_name

An abbreviation of the name of the licensed programming product in which this software was shipped. This value should be recognizable to both customer and application-support services as an acceptable name for the LPP. Examples of such values are: PSSP, GPFS, LoadLeveler, and RSCT. If this value is not provided or appears invalid, the character string **PPS_PRODUCT** is used.

source_filename

The name of the source file containing the line of code that encountered the condition being reported. For source code built under SCCS control, this should be set to **__FILE__**. If this parameter is invalid or NULL, the character string **unknown_file** is used.

sidlevel

Indicates the SCCS version number of the source code module that detected the condition being recorded. For source code built under SCCS control, this should be set to "%I%" (the double-quotes are necessary). If this value is not provided or is invalid, the character string **unknown** is used.

line_of_code_pos

The line of code location within the source code module where the condition is being reported. To allow for proper identification and location of the condition, this value should be as close to the line of code that detected the condition as possible. For source code built under SCCS control, this should be set to **__LINE__**.

(void *) NULL

Reserved for future use.

detail_data

A NULL-terminated character string that provides detailed information on the condition, similar to the Detail Data concept used by the AIX Error Log. If details of the information are too lengthy, these details can be written to a file, and the name of that file provided as the *detail_data_file* parameter. If a detail data file name is provided, *detail_data* should be set to **(char *) NULL**. If neither the *detail_data* or the *detail_data_file* parameters are provided or appear valid, the detail data segment is left blank.

detail_data_file

Name of a file containing details about the condition being reported, similar to the Detail Data concept used by the AIX Error Log. This option is used when the details are too lengthy to record within the FFDC Error Stack itself, or when a utility exists that can analyze the detail information. The contents of this file are copied to the **/var/adm/ffdc/dumps** directory, and the file's new location is recorded as the Detail Data in the FFDC Error Stack. If a file containing details of the condition does not exist, this parameter must be set to **(char *) NULL**.

catalog_name

Specifies the name of an XPG/4-compliant message catalog that contains a message describing the incident being recorded. This is a message to be displayed to the problem investigator so that the problem investigator can understand the nature of the incident that occurred. It is not meant to convey details about the failure. This message is similar to the *Err_Desc* field of an AIX Error Log template. If the message catalog indicated by *catalog_name* cannot be located in the **NLSPATH** by the FFDC end-user interfaces, the information passed in the *default_message* parameter will be displayed.

msg_set

Specifies the specific message set within the message catalog named by *catalog_name* that contains a message describing the incident being recorded. This is a message to be displayed to the problem investigator so that the problem investigator can understand the nature of the incident that occurred. It is not meant to convey details about the failure. This message is similar to the *Err_Desc* field of an AIX Error Log template. If the message set indicated by *msg_set* cannot be located in the *catalog_name* message catalog by the FFDC end-user interfaces, the information passed in the *default_message* parameter will be displayed.

msg_number

Specifies the specific message within the *msg_set* message set of the *catalog_name* message catalog that describes the incident being recorded. This is a message to be displayed to the problem investigator so that the problem investigator can understand the nature of the incident that occurred. It is not meant to convey details about the failure. This message is similar to the *Err_Desc* field of an AIX Error Log template. Substitutional parameters are permitted within this message, and are provided as

variable parameters to **fc_push_stack**. If the message cannot be located by the FFDC end-user interfaces, the information passed in the *default_message* parameter will be displayed.

default_message

A null-terminated character string that will be used to describe the condition being recorded, in cases where the message catalog or the message ID cannot be located. Substitutional parameters are permitted within this message, and are provided as variable parameters to **fc_push_stack**. A default message string is *required*, and should be written to match the default translation of *msg_number*.

...

Specifies the substitutional parameters into the description message from the message catalog and the *default_message* parameter.

Description

fc_push_stack is used by an application to record failure information to the FFDC Error Stack. Applications record descriptive information and debugging data to the FFDC Error Stack for use in later problem determination efforts.

The FFDC Error Stack is used to help understand failure conditions that occur when multiple related processes or threads are executing together on a node to perform a common task. This device is best applied to an application that creates one or more threads or subprocesses, which, in turn, may also create threads or subprocesses themselves. To use the FFDC Error Stack, the application establishes an FFDC Error Stack Environment by using the **fc_init** interface. After this environment is established, the application and any of its descendants can make use of the FFDC Error Stack.

Not all software applications will establish an FFDC Error Stack Environment under normal circumstances. However, these applications may be invoked by other applications or scripts that establish FFDC Error Stack Environments. In these cases, the scripts or applications invoking this software may wish to capture the failure information from this software, to analyze it along with other failure information from other software it invokes to discover any relationships or patterns in the failures. For this reason, software that ordinarily would not make use of the FFDC Error Stack under normal operational conditions should at least support the use of the FFDC Error Stack when it is used by any client invoking the software. This is accomplished by inheriting the FFDC Environment from the parent process through the **fc_init** interface.

Software services that execute as daemon processes are not suited to using the FFDC Error Stack. Such software applications should use the AIX Error Log and the BSD System Log to record information about failure and noteworthy conditions.

fc_push_stack records descriptions and details about noteworthy incidents to the FFDC Error Stack. If an FFDC Error Stack Environment has not been established by the application by either creation or inheritance, **fc_push_stack** does not record any information and returns control back to the caller. This action permits processes to run in a normal or "silent" mode when debugging information is not requested but also permits the application to support the use of the FFDC Error Stack when debugging information is requested.

Applications must make explicit calls to **fc_push_stack** to record information to the FFDC Error Stack when an FFDC Error Stack Environment is established. Merely establishing the environment is not enough to result in failure data being recorded. The **fc_log_error** interface will not make any records to the FFDC Error Stack.

To ensure proper identification of the condition and the location at which it was encountered, **fc_push_stack** should be called in-line in the source code module and invoked as soon as the condition is detected. **fc_push_stack** will record source-code file name and line-of-code information to assist in identifying and locating the source code that encountered the condition. **fc_push_stack** can be invoked by a subroutine or asynchronous routine to record this information if this is necessary for performance

reasons, provided that all location information and necessary failure detail information is made available to this external routine. The external recording routine must record the true location where the incident was detected.

The maximum size of an FFDC Error Stack entry is given by the `FC_STACK_MAX` definition in the `<rsct/ct_ffdc.h>` header file. `FC_STACK_MAX` defines a length in bytes. This value should be used only as a rough guide, since this length includes data that will be used by `fc_push_stack` to record the detecting file information, description information, and FFDC Failure Identifier information. Any records longer than `FC_STACK_MAX` bytes will be truncated to fit within the `FC_STACK_MAX` limit.

Return Values

`fc_push_stack` returns the following integer codes upon successful completion:

FC_SUCCESS

FFDC Environment exists, and failure information successfully recorded in the FFDC Error Stack. *fid* contains the FFDC Failure Identifier of the FFDC Error Stack record.

`fc_push_stack` returns the following integer codes when a failure occurs:

FC_NO_FC_ENVIR

FFDC Error Stack Environment does not exist. No information recorded to the FFDC Error Stack. *fid* remains unchanged. This is the normal return code to the FFDC client when an FFDC Environment did not exist to be inherited via `fc_init`.

FC_STACK_ACCESS

No information recorded to the FFDC Error Stack, and *fid* remains unchanged. Unable to access the FFDC Error Stack file. The file may have been removed, or permissions on the file or its directory have been changed to prohibit access to the FFDC Error Stack.

FC_STACK_INV

No information recorded to the FFDC Error Stack, and *fid* remains unchanged. The FFDC Error Stack file name is set to a directory name. The FFDC Environment should be considered corrupted and unusable.

FC_STACK_DIRECTORY

No information recorded to the FFDC Error Stack, and *fid* remains unchanged. The FFDC Error Stack directory does not exist or cannot be used.

FC_STACK_LOCK

No information recorded to the FFDC Error Stack: the FFDC Error Stack file could not be locked for exclusive use by this interface. Repeated attempts had been made to lock this file, and all attempts failed. Another process may have locked the file and failed to release it, or the other process may be hung and is preventing other processes from using the FFDC Error Stack. *fid* is not modified.

FC_FILESYS_SPACE

No information recorded to the FFDC Error Stack - information could not be recorded in the FFDC Error Stack, or a dump file could not be copied to `/var/adm/ffdc/dumps`, or both. There is insufficient space in the file system containing the `/var/adm/ffdc` directory. The `fcclear` command should be used to remove unneeded FFDC Error Stacks and dump files, or the system administrator needs to add more space to the file system. *fid* is not modified.

FC_ENV_CORRUPT

No information recorded to the FFDC Error Stack, and *fid* remains unchanged. The FFDC Error Stack Environment appears to be corrupted and should be considered unusable.

FC_STACK_CORRUPT

No information recorded to the FFDC Error Stack, and *fid* remains unchanged. The FFDC Error Stack file appears to be corrupted. The client should consider the FFDC Error Stack Environment unusable.

FC_IO_FAILED

No information recorded to the FFDC Error Stack, and *fid* remains unchanged. A failure occurred when reading control information from the FFDC Error Stack or writing incident information to the FFDC Error Stack. The client should conclude that the entry was not recorded for this incident.

FC_INTERNAL

No information recorded to the FFDC Error Stack, and *fid* remains unchanged. An unexpected internal failure occurred in the **fc_push_stack** routine. This problem may require the attention of customer and application support services.

When **fc_push_stack** is provided with incomplete information, it substitutes default information for the missing information and attempts to make a record in the FFDC Error Stack. Warnings are generated in these cases. In cases where more than one warning condition was detected, the subroutine returns a warning for the condition it considered the most severe. The following integer codes are returned by **fc_push_stack** when warning conditions are detected:

FC_RESOURCE_INV

The name of the resource detecting the incident was not provided. The default resource name was substituted for the missing resource name.

FC_DEFAULTDESC_INV

No default message was provided to describe the nature of the incident. If the XPG/4 message catalog containing the description message cannot be found, no description for this condition will be displayed by the **fcstkrpt** command.

FC_DESC_INV

No message was provided to describe the nature of the incident, or a component of the XPG/4 information—catalog file name, message set number, message number—was not provided. No description for this condition can be displayed by the **fcstkrpt** command.

FC_DETECTFILE_INV

At least one component of the detecting application information—source code file name, source code file version, LPP name, line of code position—was not provided. Default information was substituted for the missing information.

FC_DETAILDATA_INV

No detailed information was provided for this incident. Later problem analysis may be difficult without these details to indicate specifics on the incident.

FC_COMBINED_OPTION

Both a detailed data string and a detail data file were provided to this routine. The routine chose the detail data string and ignored the detail data file.

FC_EID_INV

An invalid pointer was provided for the *fid* parameter. **fc_push_stack** could not return the FFDC Failure Identifier for the recording made on this incident.

FC_DATA_TRUNC

The information provided to this routine would have caused an FFDC Error Stack record to exceed the **FC_STACK_MAX** limit. The record was truncated to allow it to be recorded within the system limits. Important information about the failure may have been lost during the truncation process.

Modify the program to provide less information or to record the information to a detail data file and submit the detail data file name to this routine instead.

FC_EID_FAILED

An FFDC Error Identifier could not be constructed for the report created by this routine. The fid remains unchanged, but information on the incident was recorded to the FFDC Error Stack.

FC_COPY_FAILED

The file named in the *detail_data_file* parameter could not be copied to the */var/adm/ffdc/dumps* directory. The FFDC Error Stack entry cites the original version of this file. Do not discard the original copy of this file.

Examples

To record information about a failure to the FFDC Error Stack when the FFDC Environment is established or inherited by the process:

```
rc = some_library_call(param1);
if (-1 == rc) {
    memset((void *) buffer, (int) 0, sizeof(buffer));
    sprintf(buffer, "%s - library routine %s, rc = %d, errno = %d\n",
            "Failure in function func1", "some_library_call", rc, errno);
    rc = fc_push_stack(&fid, (fc_eid_t) NULL, "myprog", "PSSP", __FILE__, "%I%",
                    __LINE__, (void *) NULL, buffer, (char *) NULL, "ssp.cat",
                    setname, EMSG567,
                    "1234-567 Error - Cannot obtain privileges for %1$s\n",
                    param1);
    if (FC_SUCCESS == rc) {
        fc_display_fid(fid);
        return(1);
    }
    :
    :
}
```

To record information about a failure condition that is related to another failure condition previously recorded to the FFDC Error Stack by an application exploiting FFDC:

```
rc = make_request(param1, param2, &aid);
if (-1 == rc) {
    memset((void *) buffer, (int) 0, sizeof(buffer));
    sprintf(buffer, "%s\n%s %d\n",
            "Failure detected in func1, making request of Beefive software",
            "Routine Name: make_request\nReturn Code:", rc);
    rc = fc_push_stack(&fid, aid, "myprog", "PSSP", __FILE__, "%I%",
                    __LINE__, (void *) NULL, buffer, (char *) NULL, "ssp.cat",
                    setname, EMSG567,
                    "1234-567 Error - Cannot obtain privileges for %1$s\n",
                    param1);
    if (FC_SUCCESS == rc) {
        (*return_fid) = fid;
        return(1);
    }
    :
    :
}
```

Implementation Specifics

This subroutine is provided as part of the First Failure Data Capture option of RS Cluster Technology (RSCT).

Related Information

"fc_init Subroutine" on page 48, "fc_log_error Subroutine" on page 66, "fc_eid_init Subroutine" on page 52, "fc_eid_is_set Subroutine" on page 53, "fcpushstk Command" on page 80, "fcstkrpt Command" on page 101, "fcreport Command" on page 103

fc_log_error Subroutine

Purpose

Records information about failure or noteworthy conditions to the AIX Error Log and the BSD System Log.

Library

First Failure Data Capture programming library (**libct_ffdc.a**)

Syntax

```
#include <rsct/ct_ffdc.h>
#include "xxxx.h" /* Error Log template source file */
int fc_log_error(fc_eid_t *fid, fc_eid_t assoc_fid, char *resource,
                char *lpp_name, char *source_filename, char *sidlevel,
                int line_of_code_pos, (void *) NULL, int event,
                unsigned error_template_label, void *detail_data,
                size_t detail_data_len, char *detail_data_file,
                char *format, ...)
```

Parameters

fid

Pointer to an `fc_eid_t` data area previously defined or allocated by the caller. **fc_log_error** returns the FFDC Failure Identifier for the record it made in the AIX Error Log through this parameter. This data area is not modified by this routine unless an FFDC Failure Identifier was successfully created by the routine.

assoc_fid

Contains the FFDC Failure Identifier for a failure condition reported by software used by this application which causes or influenced the condition being recorded at this time. This identifier should have been returned to this application as part of the software's result indication. The caller provides this identifier here so that the FFDC Error Stack can associate the failure report it is making at this time with the previously recorded failure report. This permits problem investigators to trace the cause of a failure from its various symptoms in this application and others to the root cause in the other software. If no other software failure is responsible for this condition, or if the other software did not return an FFDC Failure Identifier as part of its result information, this application should set this parameter to **(fc_eid_t) NULL**.

resource

Specifies the software component name. This is a symbolic name for the software making the report, and should be a name recognizable to both customer and application-support services. This string length is limited to 16 characters.

lpp_name

An abbreviation of the name of the licensed programming product in which this software was shipped. This value should be recognizable to both the customer and to application-support services as an acceptable name for the LPP. Examples of such values are: PSSP, GPFS, LoadLeveler, and RSCT. If this value is not provided or appears invalid, the character string **PPS_PRODUCT** is used. The combined length of this character string and the character string provided in the *source_filename* parameter should not exceed 24 characters.

source_filename

The name of the source file containing the line of code that encountered the condition being reported. For source code under SCCS control, this should be set to **__FILE__**. If this parameter is invalid or

NULL, the character string **unknown_file** is used. The combined length of this character string and the character string provided in the *lpp_name* parameter should not exceed 24 characters.

sidlevel

Indicates the SCCS version number of the source code module that detected the condition being recorded. For source code under SCCS control, this should be set to "%I%" (the double-quotes are necessary). If this value is not provided or is invalid, the character string **unknown** is used.

line_of_code_pos

The line of code location within the source code module where the condition is being reported. To allow for proper identification and location of the condition, this value should be as close to the line of code that detected the condition as possible. For source code built under SCCS control, this should be set to **__LINE__**.

(void *) NULL

This parameter is ignored in the current implementation of this interface. It is provided to permit future extensions to this interface in subsequent releases of FFDC software.

event

Specifies the FFDC Log Event Type. Current valid values are FFDC_EMERG, FFDC_ERROR, FFDC_STATE, FFDC_TRACE, FFDC_RECOV, and FFDC_DEBUG. This code gives a general description of the type of event being logged (emergency condition, permanent condition, informational notification, debugging information, etc.) and the severity of the condition. This parameter must be specified.

error_template_label

Indicates the symbolic label given to the AIX Error Logging template in the error log repository. The **errupdate** command that builds error logging templates creates a macro that maps this label to an integer code. This label begins with the characters **ERRID_** and is a maximum of 19 characters.

detail_data

A data stream that provides detailed information on the condition, used to provide the Detail Data in the AIX Error Log entry. If details of the information are too lengthy, these details can be written to a file, and the name of that file provided as the *detail_data_file* parameter. If a detail data file name is provided, *detail_data* should be set to **(char *) NULL**. If neither the *detail_data* or the *detail_data_file* parameters are provided or appear valid, the detailed data information will be set to null.

detail_data_len

Length of the *detail_data* parameter in bytes. If *detail_data* is set to NULL, this parameter is ignored.

detail_data_file

Name of a file containing details about the condition being reported. This option is used when the details are too lengthy to record within the remaining 100 bytes of Detail Data information left to the application or when a utility exists that can analyze the detail information. The contents of this file are copied to the **/var/adm/ffdc/dumps** directory, and the file's new location is recorded as the Detail Data in the AIX Error Log entry. If a file containing details of the condition does not exist, this parameter must be set to **(char *) NULL**.

format

Specifies a format string, similar to a format string provided to the **vsprintf** library routine. This specifies the format to use when recording information to the BSD System Log.

. . .

Specifies the variable argument list (*va_list*) used to fill in the data format provided by the *format* parameter. This list contains the data to be recorded to the BSD System Log and is formatted according to the setting of the *format* parameter.

Description

This interface is used by any application program that wishes to record information to the AIX Error Log and the BSD System Log. The information written to this device is intended for use by the system administrator or operator to determine what failure conditions or other noteworthy conditions have occurred on the system that require attention. The purpose of the AIX Error Log and the BSD System Log is to record enough information about a condition so that the nature, impact, and response to the condition can be determined from the report, without requiring a recreation of the condition to detect what condition occurred and where. Any software that encounters permanent failure conditions that will persist until some type of direct intervention occurs, or encounters a condition that should be brought to the attention of the system administrator, should use **fc_log_error** to record this information in the AIX Error Log and the BSD System Log.

Applications should establish a basic FFDC Environment or an FFDC Error Stack Environment before calling **fc_log_error**, either by creating or inheriting the environment. **fc_log_error** records information to the AIX Error Log and the BSD System Log even if these environments are not established, but the interface will not be capable of generating an FFDC Failure Identifier unless one of these environments exists.

Processes designed to use the FFDC Error Stack can also make use of the **fc_log_error** interface, and should make use of it if they encounter conditions that require administrator attention or intervention to resolve.

To ensure proper identification of the condition and the location at which it was encountered, **fc_log_error** should be called in-line in the source code module and invoked as soon as the condition is detected. **fc_log_error** will record the source-code file name and line-of-code information to assist in identifying and locating the source code that encountered the condition. **fc_log_error** can be invoked by a subroutine or asynchronous routine to record this information if this is necessary for performance reasons, provided that all location information and necessary failure detail information is made available to this external routine. The external recording routine must record the true location where the incident was detected.

Although **fc_log_error** reports information to both the AIX Error Log and the BSD System Log, different parameters must be provided to this interface for each recording device. The Detail Data information recorded to the AIX Error Log is not also recorded to the BSD System Log; BSD System Log information is provided in a different set of parameters. This may require the **fc_log_error** user to duplicate some information in this call.

Return Values

fc_log_error returns the following integer codes upon successful completion:

FC_SUCCESS

Information successfully queued to be written to the AIX Error Log and the BSD System Log. *fid* contains the FFDC Failure Identifier of the AIX Error Log record.

On UNIX platforms other than AIX, **fc_log_error** returns the following integer codes when a failure occurs:

FC_SYSLOG_FAILED

A record could not be made in the BSD System Log for this incident. The System Log is experiencing a failure condition. On AIX systems, a report was recorded to the AIX Error Log; on other systems, this should be considered a failure.

When **fc_log_error** is provided with incomplete information, it substitutes default information for the missing information and attempts to make a record in the FFDC Error Stack. Warnings are generated in these cases. In cases where more than one warning condition was detected, the routine returns a warning for the condition it considered the most severe.

The following integer codes are returned by **fc_log_error** when warning conditions are detected:

FC_RESOURCE_INV

The name of the resource detecting the incident was not provided. The default resource name was substituted for the missing resource name.

FC_DETECTFILE_INV

At least one component of the detecting application information—source code file name, source code file version, LPP name, line of code position—was not provided. Default information was substituted for the missing information.

FC_LOGEVENT_INV

event is not a valid FFDC event type. The event type **FFDC_DEBUG** has been assigned to this incident record.

FC_LOGMSG_INV

A message was not supplied in the *format* parameter. As a result, a generic message was recorded to the BSD System Log for this incident.

FC_DETAILDATA_INV

No detailed information was provided for this incident. Later problem analysis may be difficult without these details to indicate specifics on the incident.

FC_DETAILLEN_INV

The length of the detail data string was greater than the capacity of the AIX Error Log entry limit. Detail data was truncated to fit in the available space. Some information on the incident may have been lost in this truncation.

FC_SYSLOG_FAILED

A record could not be made in the BSD System Log for this incident. The System Log may not be enabled or may be experiencing problems. On AIX systems, a report was recorded to the AIX Error Log; on other systems, this should be considered a failure.

FC_COMBINED_OPTION

Both a detailed data string and a detail data file were provided to this routine. The routine chose the detail data string and ignored the detail data file.

FC_EID_INV

An invalid pointer was provided for the *fid* parameter. **fc_log_error** could not return the FFDC Error Identifier for the recording made on this incident.

FC_EID_FAILED

An FFDC Error Identifier could not be constructed for the report created by this routine. *fid* remains unchanged, but information on the incident was recorded to the AIX Error Log and the BSD System Log.

FC_COPY_FAILED

The file named in the *detail_data_file* parameter could not be copied to the **/var/adm/ffdc/dumps** directory. The FFDC Error Stack entry cites the original version of this file. Do not discard the original copy of this file.

Examples

For this example, a C source code file attempts to access configuration information from a file. If this attempt fails, the code will record a failure to the AIX Error Log using the following template source code:

```
*! mymesgcat.cat
+ SP_FFDCXMPL_ER:
  Comment      = "Configuration Failed - Exiting"
  Class        = S
  Log          = true
  Report       = true
  Alert        = false
  Err_Type     = PERM
  Err_Desc     = {3, 10, "CONFIGURATION FAILURE - EXITING"}
  Prob_Causes  = E89B
  User_Causes  = E811
  User_Actions = 1056
  Fail_Causes  = E906, E915, F072, 108E
  Fail_Actions = {5, 14, "VERIFY USER HAS CORRECT PERMISSIONS TO ACCESS FILE"},
                 {5, 15, "VERIFY CONFIGURATION FILE"}
  Detail_Data  = 46, 00A2, ALPHA
  Detail_Data  = 42, EB2B, ALPHA
  Detail_Data  = 42, 0030, ALPHA
  Detail_Data  = 16, EB00, ALPHA
  Detail_Data  = 16, 0027, ALPHA
  Detail_Data  = 4, 8183, DEC
  Detail_Data  = 4, 8015, DEC
  Detail_Data  = 60, 8172, ALPHA
```

This definition yields the following AIX Error Logging Template:

```
LABEL:          ERRID_SP_FFDCXMPL_ER
IDENTIFIER:     <calculated by errupdate during source code build>

Date/Time:     <filled in by AIX Error Log subsystem>
Sequence Number: <filled in by AIX Error Log subsystem>
Machine Id:    <filled in by AIX Error Log subsystem>
Node Id:       <filled in by AIX Error Log subsystem>
Class:         S
Type:          PERM
Resource Name: <filled in by resource parameter to fc_log_error>
```

```
Description
CONFIGURATION FAILURE - EXITING
```

```
Probable Causes
COULD NOT ACCESS CONFIGURATION FILE
```

```
User Causes
USER CORRUPTED THE CONFIGURATION DATABASE OR METHOD
```

```
Recommended Actions
RE-CREATE FILE
```

```
Failure Causes
COULD NOT ACCESS CONFIGURATION FILE
PERMISSIONS ERROR ACCESSING CONFIGURATION DATABASE
FILE READ ERROR
FILE IS CORRUPT
```

```
Recommended Actions
VERIFY USER HAS CORRECT PERMISSIONS TO ACCESS FILE
VERIFY CONFIGURATION FILE
```

```
Detail Data
```

```

DETECTING MODULE
<filled in by fc_log_error parameters>
ERROR ID
<The FFDC Failure Identifier created by fc_log_error>
REFERENCE CODE
<The assoc_fid parameter value to fc_log_error>
FILE NAME
<Must be supplied as part of detail_data parameter to fc_log_error>
FUNCTION
<Must be supplied as part of detail_data parameter to fc_log_error>
RETURN CODE
<Must be supplied as part of detail_data parameter to fc_log_error>
ERROR CODE AS DEFINED IN sys/errno.h
<Must be supplied as part of detail_data parameter to fc_log_error>
USER ID
<Must be supplied as part of detail_data parameter to fc_log_error>

```

The first three Detail Data Fields are constructed by the **fc_log_error** routine from information passed in the parameters. The remaining Detail Data information must be supplied as a single stream of data by the **fc_log_error** client. To do this, the **fc_log_error** client must construct a data area of 100 bytes (the amount of the remaining space: 230 bytes for Detail Data minus the 130 bytes filled in automatically by **fc_log_error**) and copy binary data to this memory area. The example source code segment that follows demonstrates how this is done, and how **fc_log_error** is invoked to record the information in the AIX Error Log and the BSD System Log.

```

char    config_fname[16];
char    inbuf[80];
pid_t   myclient;
char    failure_details[100];
char    function_name[16];
void    *failure_field;
int     fildes;
int     rc;
int     ecode;
fc_eid_t fid;
:
myclient = getpid();
memset((void *) config_fname, (int) 0, (sizeof(char) * 16));
strcpy(config_fname, "/configfile.bin");
:
fildes = open(config_fname, O_RDONLY);
:
rc = read(fildes, inbuf, 80);
ecode = errno;
if (-1 == rc) {
    /*
     * Create Detail Data Memory Block for AIX Error Log Template
     * Need to know the EXACT structure of the Template to do this correctly.
     * Field 1 - filled in by fc_log_error
     * Field 2 - filled in by fc_log_error
     * Field 3 - filled in by fc_log_error
     * Field 4 - name of configuration file being used
     * Field 5 - name of function call that failed
     * Field 6 - return code from failing function
     * Field 7 - errno from failing function call
     * Field 8 - user ID using this software
     * This source code supplied fields 4 through 8 in the failure_details[]
     * array. Data must be copied into the array as binary data.
     */
    memset((void *) failure_details, (int) 0, (sizeof(char) * 102));
    memset((void *) function_name, (int) 0, (sizeof(char) * 16));
    strcpy(function_name, "read");
    failure_field = (void *) (failure_details);
    memcpy(failure_field, (void *) config_fname, (sizeof(char) * 16));
}

```

```

failure_field += (sizeof(char) * 16);
memcpy(failure_field, (void *) function_name, (sizeof(char) * 16));
failure_field += (sizeof(char) * 16);
memcpy(failure_field, (void *) &rc, sizeof(int));
failure_field += sizeof(int);
memcpy(failure_field, (void *) &ecode, sizeof(int));
failure_field += sizeof(int);
memcpy(failure_field, (void *) &myclient, sizeof(pid_t));
rc = fc_log_error(&fid, (fc_eid_t) NULL, "myprog", "PSSP", __FILE__, "%I%",
                __LINE__, (void*) NULL, FFDC_ERROR, ERRID_SP_FFDC_EXMPL_ER,
                failure_details, 100, (char*) NULL,
                "Configuration Failure - Exiting");
if (FC_SUCCESS == rc) {
    fid_to_client = fid;
    close(filides);
    return(-1);
}
:
}

```

Now consider a slight variation on the above example, using the same AIX Error Logging template, but this time using a library routine to obtain the configuration data from a file that this source code supplies. The library routine returns an FFDC Failure Identifier if it encounters any failure conditions:

```

char    config_fname[16];
char    inbuf[80];
pid_t   myclient;
char    failure_details[100];
char    function_name[16];
void    *failure_field;
int filides;
int rc;
int ecode;
fc_eid_t fid, aid;
:
myclient = getpid();
memset((void *) config_fname, (int) 0, (sizeof(char) * 16));
strcpy(config_fname, "/configfile.bin");
:
rc = get_config_data(config_fname, inbuf, 80, &aid);
ecode = errno;
if (-1 == rc) {
    /*
     * Create Detail Data Memory Block for AIX Error Log Template
     * Need to know the EXACT structure of the Template to do this correctly.
     * Field 1 - filled in by fc
     * Field 2 - filled in by fc_log_error
     * Field 3 - filled in by fc_log_error
     * Field 4 - name of configuration file being used
     * Field 5 - name of function call that failed
     * Field 6 - return code from failing function
     * Field 7 - errno from failing function call
     * Field 8 - user ID using this software
     * This source code supplied fields 4 through 8 in the failure_details[]
     * array. Data must be copied into the array as binary data.
     */
    memset((void *) failure_details, (int) 0, (sizeof(char) * 102));
    memset((void *) function_name, (int) 0, (sizeof(char) * 16));
    strcpy(function_name, "get_config_data");
    failure_field = (void *) (failure_details);
    memcpy(failure_field, (void *) config_fname, (sizeof(char) * 16));
    failure_field += (sizeof(char) * 16);
    memcpy(failure_field, (void *) function_name, (sizeof(char) * 16));
    failure_field += (sizeof(char) * 16);
    memcpy(failure_field, (void *) &rc, sizeof(int));
}

```

```

failure_field += sizeof(int);
memcpy(failure_field, (void *) &ecode, sizeof(int));
failure_field += sizeof(int);
memcpy(failure_field, (void *) &myclient, sizeof(pid_t));
rc = fc_log_error(&fid, aid, "myprog", "PSSP", __FILE__, "%I%",
                __LINE__, (void*)NULL, FFDC_ERROR, ERRID_SP_FFDCEXMPL_ER,
                failure_details, 100, (char*)NULL,
                "Configuration Failure - Exiting");
if (FC_SUCCESS == rc) {
    fid_to_client = fid;
    close(fildes);
    return(-1);
}
:
}

```

Implementation Specifics

This subroutine is provided as part of the First Failure Data Capture option of RS Cluster Technology (RSCT).

Related Information

"<rsct/ct_ffdc.h> Header File" on page 46, "fc_init Subroutine" on page 48, "fc_push_stack Subroutine" on page 59, "fc_eid_init Subroutine" on page 52, "fc_eid_is_set Subroutine" on page 53, "fclogerr Command" on page 87, "fcreport Command" on page 103

fcinit Command

Purpose

Establishes or inherits a First Failure Data Capture execution environment.

Syntax

For Bourne and Korn Shells:

```
. /usr/sbin/rsct/bin/fcinit.sh [ [-l] [-s{c|i}] ] | [-h]
```

For C Shells:

```
source /usr/sbin/rsct/bin/fcinit.csh [ [-l] [-s{c|i}] ] | [-h]
```

Flags

- h** Displays a help message to standard output and exits. No other processing is performed, regardless of the options specified.
- l** Indicates that the process wishes to make use of the AIX Error Log only. This option is not necessary when the **-s** option is specified, since use of the AIX Error Log is permitted within an FFDC Error Stack Environment.
- s** Indicates that an FFDC Error Stack Environment is to be established. Applications wishing to use the **fcpushstk** interface must specify this flag. Upon successful completion of this command, an FFDC Error Stack file is reserved for the script in the **/var/adm/ffdc/stacks** directory. This flag must be specified with one of two possible options:
 - c** Requests that the FFDC Error Stack Environment be *created*. If an FFDC Error Stack Environment was not created by an ancestor process, it will be created. If such an environment was previously created by an ancestor process, this process will *inherit* the FFDC Error Stack Environment as if the **i** option had been specified.
 - i** Specifies that an FFDC Error Stack Environment is to be *inherited* if it was previously established by an ancestor process. If an FFDC Error Stack Environment was not previously established by an ancestor process, an FFDC Error Stack Environment is not established for this process, and this process cannot make use of an FFDC Error Stack (although it may make use of the AIX Error Log and the BSD System Log).

Description

This interface must be used by a script program that wishes to use the FFDC interfaces for recording information to the AIX Error Log, the BSD System Log, or the FFDC Error Stack .

Applications may wish to establish an FFDC Environment for one of the following reasons:

- The script may wish to record information to the AIX Error Log. Scripts can use **fcinit** to establish a basic FFDC Environment
- The script wants to have itself and any descendant processes created by itself or its children to record failure information to the FFDC Error Stack. In this case, the script considers itself a "top-level" application that will cause multiple "lower-level" applications to be created, and the success of the "top-level" application depends upon the success of these "lower-level" applications. When using **fcinit** in this fashion, the process is said to *establish* or *create* the FFDC Error Stack Environment.
- The script uses the FFDC Error Stack or the FFDC Trace only in those cases when the script is invoked by an ancestor process that wants failure information or trace information recorded to these devices. In all other cases, the script does not wish to use these devices. When using **fcinit** in this fashion, the process is said to *inherit* the FFDC Error Stack Environment.

Any process wishing to record information to the AIX Error Log or the BSD System Log through the FFDC interfaces must establish an FFDC Environment. If the process does not wish to make use of an FFDC Error Stack, the process can establish a basic FFDC Environment that does not make use of an FFDC

Error Stack. An FFDC Error Stack Environment, which contains an FFDC Error Stack, is established by a process when that process wants to have failure information from itself, any threads it may create, and any descendant processes it may create to be recorded in an FFDC Error Stack. An *FFDC Error Stack Environment*, which contains an FFDC Error Stack, is inherited by a process when that process wants to record failure information to an FFDC Error Stack file only when one of its ancestors has requested for processes to do so; in all other cases, the process will not record failure information to the FFDC Error Stack.

The FFDC Error Stack Environment, which contains an FFDC Error Stack, reserves an FFDC Error Stack file, so that failure information is recorded to a file in the `/var/adm/ffdc/stacks` directory. These files use the naming format `script_name.PID.date_and_time`, where `script_name` is the name of the script itself, `PID` is the process identifier of the script, and `date_and_time` is the date and time when the script was executed. Whenever this script or children processes of this script record failure information to the FFDC Error Stack, it will be recorded in this file.

In order for information to be recorded in the FFDC Error Stack by a process, the process must use the `fcpushstk` FFDC interface, and the process has to be operating within an established FFDC Error Stack Environment. If an FFDC Error Stack Environment does not exist, or if the `fcpushstk` interface is not used when an FFDC Error Stack Environment exists, no information is recorded by that process in the FFDC Error Stack. This function permits processes to run in a normal or "silent" mode when failure debugging information is not wanted or needed, but also permits this information to be available when the process is invoked within a special environment for debugging.

`fcinit` must be executed within the FFDC client's process environment ("sourced") in order for the command to properly set the FFDC Environment for the script. Script-based FFDC clients using this command must "source" the command in order for `fcinit` to execute within the client's process image. If this is not done, the FFDC interface is executed within its own process image; any settings of the FFDC Environment are lost after the FFDC interface completes. To demonstrate how a script-based application would "source" the `fcinit` command, a Korn Shell program would issue the following instruction:

```
. fcinit.sh <options and arguments>
```

A C Shell script would do the following:

```
source fcinit.csh <options and arguments>
```

Processes that use the `fclogerr` FFDC interface must establish an *FFDC Environment*. If the process only wishes to use the `fclogerr` interface, the *FFDC Environment* can be established without an FFDC Error Stack.

If an FFDC Environment already exists when a script attempts to create one, the script inherits the existing FFDC Environment instead of creating its own.

Return Values

`fcinit` returns the following exit status codes upon completion:

- 0 FFDC Environment successfully established.
- 1 FFDC Environment successfully inherited.
- 2 Help information displayed and processing ended.

`fcinit` returns the following exit status codes upon detection of a failure:

12

FFDC Environment not established or inherited - Unknown function parameter provided.

13

FFDC Error Stack Environment not established or inherited - caller indicated that the FFDC Environment should be both created and inherited.

14

FFDC Environment not established in this call - the caller already has an FFDC Environment established for itself - this routine may have been executed multiple times.

15

FFDC Error Stack Environment not established or inherited - an FFDC Error Stack Environment did not exist, and the FC_INHERIT option was specified.

16

FFDC Environment not established or inherited - the client's process environment could not be modified by this routine.

17

FFDC Environment not established or inherited - the FFDC Environment appears to be corrupted and should be considered unusable.

18

FFDC Environment not established or inherited - the routine could not allocate the memory required to modify the client's process environment.

19

FFDC Error Stack Environment not established or inherited - Unable to reserve the FFDC Error Stack file for the calling process - the FFDC Error Stack directory does not exist or cannot be used.

21

FFDC Error Stack Environment not established or inherited - Unable to reserve the FFDC Error Stack file for the calling process - the file already exists

42

FFDC Error Stack Environment not established or inherited - creation and use of FFDC Error Stacks has been disabled by the system administrator. Scripts can establish only a basic FFDC Environment that makes use of the AIX Error Log and the BSD System Log.

99

FFDC Environment not established or inherited - an unexpected internal failure occurred within **fcinit**. This condition may require the attention of customer and application-support services.

Examples

For a Korn Shell script to establish a basic FFDC Environment for using the AIX Error Log and the BSD System Log only (an FFDC Error Stack is not to be used or reserved):

```
# Set up an FFDC Environment to use the AIX Error Log only. An FFDC Error
# Stack is not needed for this script.
. fcinit.sh -l
rc=$?
if ((rc != 0))
then
    print "fcinit failed with exit code of $rc"
    exit 1
fi
# Normal processing starts
```

For a Korn Shell script to establish an FFDC Error Stack Environment that causes the script and any descendant process to record failure information to the FFDC Error Stack:

```
# Set up FFDC Environment to record failure information to the FFDC Error
# Stack
. fcinit.sh -sc
rc=$?
if ((rc != 0))
then
    print "fcinit failed with a code of $rc"
    exit 1
fi
# Normal processing starts
```

Note: The FFDC client may receive an indication that an FFDC Error Stack Environment was inherited, instead of created by the **fcinit** call. This occurs when an FFDC Error Stack Environment was already established by one of the process's ancestors.

To inherit an FFDC Error Stack Environment from the process's parent process:

```
# Inherit an FFDC Environment from parent process if it exists - otherwise,
# operate in a normal "silent" mode
. fcinit.sh -si
rc=$?
if ((rc != 0))
then
    print "fcinit failed with a code of $rc"
    exit 1
fi
# Normal processing starts
```

Implementation Specifics

This command is provided as part of the First Failure Data Capture option of IBM RS Cluster Technology (RSCT).

Related Information

"fclogerr Command" on page 87, "fcpushstk Command" on page 80, "fcteststk Command" on page 78, "fccheck Command" on page 108, "fc_init Subroutine" on page 48

fcteststk Command

Purpose

Test for the presence of a First Failure Data Capture Error Stack environment

Syntax

```
/usr/sbin/rsct/bin/fcteststk [-q] | [-h]
```

Flags

-h Displays a usage message for this command. No further processing is performed.

-q

Suppresses output from this command that explains whether or not an FFDC Environment was established. The command user will be required to test the exit status from the command to determine whether an FFDC Environment is established for this process.

Description

fcteststk can be called by any application program that wishes to use the FFDC Error Stack to test if these facilities have been activated. By performing this test, applications can avoid the performance burden of collecting failure information in cases where an *FFDC Environment* has not been established. This interface is provided primarily for use by library routines, which would not have any knowledge of whether their client application established or inherited an *FFDC Environment*.

An *FFDC Error Stack Environment* is established by a process when that process wants to have failure information from itself, any threads it may create, and any descendant processes it may create to be recorded in an FFDC Error Stack. An *FFDC Error Stack Environment* is inherited by a process when that process wants to record failure information to an FFDC Error Stack file only when one of its ancestors has requested for processes to do so; in all other cases, the process will not record failure information to the FFDC Error Stack. Processes use **fcinit** to either establish or inherit the FFDC Error Stack Environment.

The FFDC Error Stack Environment reserves an FFDC Error Stack file, so that failure information is recorded to a file in the **/var/adm/ffdc/stacks** directory. These files use the naming format **script_name.PID.date_and_time**, where *script_name* is the name of the script itself, *PID* is the process identifier of the script, and *date_and_time* is the date and time when the script was executed. Whenever this script or children processes of this script record failure information to the FFDC Error Stack, it will be recorded in this file.

Applications use the **fcpushstk** interface to record failure information to the FFDC Error Stack. However, the application may need to collect this information from various locations before recording the information, and obtaining this information can impact the application's overall performance. The application should not need to collect this information if the *FFDC Error Stack Environment* was not established or inherited. To avoid this performance impact, the application can issue **fcteststk** to determine if an *FFDC Error Stack Environment* is available, and if so, begin collecting the failure information. If the *FFDC Error Stack Environment* does not exist, the application can avoid collecting this information.

Processes that use the **fclogerr** FFDC interface can use **fclogerr** when an *FFDC Environment* exists, whether or not an FFDC Error Stack is in use by the *FFDC Environment*. Whenever **fclogerr** is used, failure information is recorded to the AIX Error Log and the BSD System Log, regardless of whether an FFDC Error Stack was reserved. Any application that records information using the **fclogerr** interface must *always* collect the failure information and record it, regardless of whether an FFDC Error Stack is in use.

Return Values

- 0** An FFDC Error Stack Environment exists.
- 2** Help information displayed and processing ended.

- 12 No processing performed. An invalid option was specified.
- 15 FFDC Error Stack Environment has not been established or inherited by the client at this point in time.
- 17 FFDC Error Stack Environment appears to be corrupted and should be considered unusable.

Examples

To test whether an FFDC Error Stack Environment exists for an application:

```
fcteststk -q
if (($? == 0))
then
    # Collect failure information
    :
    :
    # Use fcpushstk to record failure info
    :
    :
fi
```

Implementation Specifics

This command is provided as part of the First Failure Data Capture option of IBM RS Cluster Technology (RSCT).

Related Information

“fcinit Command” on page 74, “fcpushstk Command” on page 80, “fc_test_stack Subroutine” on page 57

fcpushstk Command

Purpose

Records information about failure or noteworthy conditions to the First Failure Data Capture Error Stack.

Syntax

```
/usr/sbin/rsct/bin/fcpushstk { [-aassoc_fid] -cmessage_catalog_name  
-mmessage_set -nmessage_number [-omessage_param[,message_param,...]]  
-lpp_name -pline_of_code_pos -rresource -ssource_filename -vsidlevel  
{[-ddetail_data]|[-fdetail_data_file]} } default_message | -h
```

Flags

-a

Specifies an FFDC Failure Identifier for a failure condition reported by software used by this application which causes or influenced the condition being recorded at this time. This identifier should have been returned to this application as part of the software's result indication. The caller provides this identifier here so that the FFDC Error Stack can associate the failure report it is making at this time with the previously recorded failure report. This permits problem investigators to trace the cause of a failure from its various symptoms in this application and others to the root cause in the other software. If no other software failure is responsible for this condition, or if the other software did not return an FFDC Failure Identifier as part of its result information, the **-a** option should not be provided.

-c

Indicates the name of the XPG/4-compliant message catalog that contains a description of the failure being recorded. This name is relative to the **/usr/lib/nls/msg/\$LANG** directory. If the message catalog cannot be found, the *default_message* will be displayed to describe the failure. Note that the *default_message* will not be translated between locales.

-d

A character string that provides detailed information on the condition, similar to the Detail Data concept used by the AIX Error Log. If details of the information are too lengthy, these details can be written to a file, and the name of that file provided as an argument to the **-f** option. The **-d** and **-f** options cannot be specified at the same time. If neither the **-d** or the **-f** options are provided or appear valid, the character string **no detail data** is recorded.

-f

Specifies the name of a file containing details about the condition being reported, similar to the Detail Data concept used by the AIX Error Log. This option is used when the details are too lengthy to record within the FFDC Error Stack itself, or when a utility exists that can analyze the detail information. The contents of this file is copied to the **/var/adm/ffdc/dumps** directory, and the file's new location is recorded as the Detail Data in the FFDC Error Stack. If a file containing details of the condition does not exist, do not specify this option. The **-d** and **-f** options cannot be specified at the same time.

-h

Displays a help message to standard output and exits. No other processing is performed, regardless of the options specified.

-l

Specifies an abbreviation of the name of the licensed programming product in which this software was shipped. This value should be recognizable to both customer and application-support services

as an acceptable name for the LPP. Examples of such values are: PSSP, GPFS, LoadLeveler, and RSCT. If this option is not provided or appears invalid, the character string **PPS_PRODUCT** is used.

-m

Specifies the message set containing the message describing the failure in the message catalog file. If this message set cannot be located, the *default_message* will be displayed to describe the failure. Note that **default_message** will not be translated to the user's locale.

-n

Specifies the message number that describes the failure being recorded. If this message cannot be located, the *default_message* will be displayed to describe the failure. Note that *default_message* will not be translated to the user's locale.

-o

Specifies a list of substitution parameters within the message indicated by the **-n** option. **fcpushstk** only supports character strings as substitutional parameters (%s) due to the shell operating environment. If multiple substitutional parameters are provided, each one must be separated by a comma (.). If any of these substitution parameters contain imbedded white space, they must be enclosed in double quotes ("").

-q

Suppresses the generation of warning messages from the command. Warning are generated when the command must substitute default information for missing information, or when the command is unable to copy the *detail_data_file* to the **/var/adm/ffdc/dumps** directory.

-r

Specifies the software component name. This is a symbolic name for the software making the report, and should be a name recognizable to both customer and application-support services.

-p

Specifies the line of code location within the source code module where the condition is being reported. The value provided must be a valid integer value. To allow for proper identification and location of the condition, this value should be as close to the line of code that detected the condition as possible. Korn Shell scripts can use the value of **\$LINENO**. Script languages that do not provide a special line count variable can provide a symbolic value here that a developer can use to locate the spot in the source code where **fcpushstk** is being used. If this option is not valid or not provided, the value of **0** is used.

-s

Specifies the name of the source file containing the line of code that encountered the condition being reported. For Korn and Borne Shell scripts, the argument to this option should be set to **\$0**; C Shell scripts would set this argument to **\${0}**. If this option is not provided or not valid, the character string **unknown_file** is used.

-v

Indicates the SCCS version number of the source code module that detected the condition being recorded. For source code under SCCS control, this should be set to **"%I%"** (the double-quotes are necessary). If this option is not provided or is not valid, the character string **unknown** is used.

Parameters

default_message

Indicates a default message to be used as a description of the failure, when the information cannot be retrieved from the message catalog information supplied through the **-c**, **-m**, and **-n** options. If this string contains positional parameters, all positional parameters must be specified to

be character strings (%s). The message should be enclosed in double quotes (") if it contains any embedded white space. **fcpushstk** limits the overall length of this string to 72 characters.

Description

fcpushstk is used by scripts to record failure information to the FFDC Error Stack. Scripts record descriptive information and debugging data to the FFDC Error Stack for use in later problem determination efforts.

The FFDC Error Stack is used to help understand failure conditions that occur when multiple related processes or threads are executing together on a node to perform a common task. This device is best applied to an application that creates one or more threads or subprocesses, which in turn, may also create threads or subprocesses themselves. To use the FFDC Error Stack, the script establishes an *FFDC Error Stack Environment* using the **fcinit** interface. After this environment is established, the application and any of its descendants can make use of the FFDC Error Stack.

Not all software applications will establish an FFDC Error Stack Environment. However, these applications may be invoked by other applications or scripts that establish FFDC Error Stack Environments. In these cases, the scripts or applications invoking this software may wish to capture the failure information from this software, to analyze it along with other failure information from other software it invokes to discover any relationships or patterns in the failures. For this reason, software that ordinarily would not make use of the FFDC Error Stack under normal operational conditions should at least support the use of the FFDC Error Stack when it is used by any client invoking the software. This is accomplished by *inheriting* the FFDC Error Stack Environment from the parent process through the **fcinit** interface.

fcpushstk records descriptions and details about noteworthy conditions to the FFDC Error Stack. If an *FFDC Error Stack Environment* has not been established by the script, either by creation or inheritance, **fcpushstk** does not record any information and returns control back to the caller. This action permits the script to run in a normal "silent" mode when debugging information is not requested, but also permits the script to support the use of the FFDC Error Stack when debugging information is requested.

Scripts must make explicit calls to **fcpushstk** to record information to the FFDC Error Stack when an FFDC Error Stack Environment is established. Merely establishing the environment is not enough to result in failure data being recorded. The **fclogerr** command will not make any records to the FFDC Error Stack.

To ensure proper identification of the condition and the location at which it was encountered, **fcpushstk** should be called in-line in the script's source code module, invoked as soon as the condition is detected. **fcpushstk** will record source code file name and line of code information to assist in identifying and locating the source code that encountered the condition. **fcpushstk** can be invoked by a subroutine or autoloading routine to record this information if this is necessary, provided that all location information and necessary failure detail information is made available to this external routine. The external recording routine must record the true location where the incident was detected.

The maximum size of an FFDC Error Stack entry is given by the `FC_STACK_MAX` definition in the `<rsct/ct_ffdc.h>` header file. `FC_STACK_MAX` defines a length in bytes. This value should be used only as a rough guide, since this length includes data that will be used by **fcpushstk** to record the detecting file information, description information, and FFDC Failure Identifier information. Any records longer than `FC_STACK_MAX` bytes will be truncated to fit within the `FC_STACK_MAX` limit.

Return Values

fcpushstk returns the following exit status codes upon successful completion:

0

FFDC Error Stack Environment exists, and failure information successfully recorded in the FFDC Error Stack. An FFDC Failure Identifier for the record is displayed to standard output. The caller should capture standard output to obtain this value.

2

Help information displayed and processing ended.

fcpushstk returns the following exit status codes when a failure occurs:

11

No information recorded to the FFDC Error Stack, and no FFDC Failure Identifier is provided by this command. The client requested to use an option not supported in this release of the FFDC software

12

No information recorded to the FFDC Error Stack, and no FFDC Failure Identifier is provided by this command. Unknown function parameter provided to the interface.

15

FFDC Error Stack Environment does not exist. No information recorded to the FFDC Error Stack. No FFDC Failure Identifier is generated by this command. This is the normal return code to the FFDC client when an FFDC Error Stack Environment did not exist to be inherited via **fcinit**.

17

No information recorded to the FFDC Error Stack, and no FFDC Failure Identifier is provided by this command. The FFDC Error Stack Environment appears to be corrupted and should be considered unusable.

19

No information recorded to the FFDC Error Stack - the FFDC Error Stack directory does not exist or cannot be used. No FFDC Failure Identifier is provided by this command.

20

No information recorded to the FFDC Error Stack, and no FFDC Failure Identifier is provided by this command. Unable to access the FFDC Error Stack file. The file may have been removed, or permissions on the file or its directory have been changed to prohibit access to the FFDC Error Stack.

22

No information recorded to the FFDC Error Stack - the FFDC Error Stack file could not be locked for exclusive use by this interface. Repeated attempts had been made to lock this file, and all attempts failed. Another process may have locked the file and failed to release it, or the other process may be hung and is preventing other processes from using the FFDC Error Stack. No FFDC Failure Identifier is provided by this command.

24

No information recorded to the FFDC Error Stack, and no FFDC Failure Identifier is provided by this command. The FFDC Error Stack file appears to be corrupted. The client should consider the FFDC Error Stack Environment unusable.

25

No information recorded to the FFDC Error Stack, and no FFDC Failure Identifier is provided by this command. The FFDC Error Stack file name is set to a directory name. The FFDC Error Stack Environment should be considered corrupted and unusable.

32

A dump file could not be copied to the **/var/adm/ffdc/dumps** directory. There is insufficient space in the file system containing the **/var/adm/ffdc** directory. The **fcclear** command should be used to remove unneeded FFDC Error Stacks and dump files, or the system administrator needs to add more space to the file system. No FFDC Failure Identifier is provided by this command.

40

No information recorded to the FFDC Error Stack - information could not be recorded in the FFDC Error Stack. There is insufficient space in the file system containing the **/var/adm/ffdc** directory. The **fcclear** command should be used to remove unneeded FFDC Error Stacks and dump files, or the system administrator needs to add more space to the file system. No FFDC Failure Identifier is provided by this command.

41

No information recorded to the FFDC Error Stack, and no FFDC Failure Identifier is provided by this command. A failure occurred when reading control information from the FFDC Error Stack or writing incident information to the FFDC Error Stack. The client should conclude that the entry was not recorded for this incident.

99

No information recorded to the FFDC Error Stack, and no FFDC Failure Identifier is provided by this command. An unexpected internal failure occurred in the **fc_push_stack** routine. This problem may require the attention of application-support services.

When **fcpushstk** is provided with incomplete information, it substitutes default information for the missing information and attempts to make a record in the FFDC Error Stack. Warnings are generated in these cases, and warning messages are displayed to the standard error device unless the **-q** option has been specified. In cases where more than one warning condition was detected, the command generates an exit status code corresponding to the most severe warning condition it detected. The following exit status codes are returned by **fcpushstk** when warning conditions are detected:

26

Both a detailed data string and a detail data file were provided to this routine. The routine chose the detail data string and ignored the detail data file.

28

The name of the resource detecting the incident was not provided. The default resource name was substituted for the missing resource name.

29

At least one component of the detecting application information—source code file name, source code file version, LPP name, line of code position—was not provided. Default information was substituted for the missing information.

30

No default message was provided to describe the nature of the incident. If the XPG/4 message catalog containing the description message cannot be found, no description for this condition will be displayed by the **fcstkrpt** command.

31

No message was provided to describe the nature of the incident, or a component of the XPG/4 information—catalog file name, message set number, message number—was not provided. No description for this condition can be displayed by the **fcstkrpt** command.

32

The file named in the *detail_data_file* parameter could not be copied to the **/var/adm/ffdc/dumps** directory. The FFDC Error Stack entry cites the original version of this file. Do not discard the original copy of this file.

35

No detailed information was provided for this incident. Later problem analysis may be difficult without these details to indicate specifics on the incident.

37

An FFDC Failure Identifier could not be constructed for the report created by this routine. No FFDC Failure Identifier is provided by this command, but information on the incident was recorded to the FFDC Error Stack.

44

The information provided to this command would have caused an FFDC Error Stack record to exceed the FC_STACK_MAX limit. The record was truncated to allow it to be recorded within the system limits. Important information about the failure may have been lost during the truncation process. Modify the script to provide less information, or to record the information to a detail data file and submit the detail data file name to this command instead.

Examples

To record information about a failure to the FFDC Error Stack when the FFDC Environment is established or inherited by the process:

```
#!/bin/ksh
:
:
cp /tmp/workfile $FILENAME
RC=$?
if ((RC != 0))
then
  FFDCID=$(fcpushstk -c mymsg.cat -m2 -n10 -o$FILENAME -r myprog
    -d"cp exit status $RC - file being copied /tmp/workfile" -s$0
    -p$LINENO -v"%I%" -lPSSP "Cannot update configuration file %1$s")
  if (($? == 0))
  then
    fcdispfid $FFDCID
    return 1
  fi
fi
:
:
```

To make the same recording from a script language that does not have a line of code variable available:

```
#!/bin/bsh
:
:
CODESCTN=14          # Used to identify where in the script code we are
cp /tmp/workfile $FILENAME
RC=$?
if test $RC -ne 0
then
  FFDCID='fcpushstk -c mymsg.cat -m2 -n10 -o$FILENAME -r myprog
    -d"cp exit status $RC - file being copied /tmp/workfile" -s$0
    -p$CODESCTN -v"%I%" -lPSSP "Cannot update configuration file %1$s"'
  if test $? -eq 0
  then
    fcdispfid $FFDCID
    return 1
  fi
fi
CODESECTION=15      # New code section begins - a different task starts
:
:
```

To record information about a failure condition that is related to another failure condition previously recorded to the FFDC Error Stack by an application exploiting FFDC:

```
#!/bin/ksh
:
:
ASSOC_FID=$(/usr/lpp/ssp/bin/somecmd -a -b)
RC=$?if ((RC != 0))
then
    FFDCID=$(fcpushstk -a$ASSOC_FID -c mymsg.cat -m2 -n10 -o$FILENAME -r myprog
        -d"cp exit status $RC - file being copied /tmp/workfile" -s$0
        -p$LINENO -v"%I%" -lPSSP "Cannot update configuration file %1$s")
    if (($? == 0))
    then
        fcdispfid $FFDCID
        return 1
    fi
fi
:
:
```

Implementation Specifics

This command is provided as part of the First Failure Data Capture option of IBM RS Cluster Technology (RSCT).

Related Information

“fcdecode Command” on page 97, “fcdispfid Command” on page 95, “fcinit Command” on page 74, “fcreport Command” on page 103, “fcstkrpt Command” on page 101, “fcteststk Command” on page 78, “fc_push_stack Subroutine” on page 59

fclogerr Command

Purpose

Records information about failure or noteworthy conditions to the AIX Error Log and the BSD System Log.

Syntax

```
/usr/sbin/rsct/bin/fclogerr {-e event -t error_template_label  
-i error_template_headerfile -r resource -s source_filename  
-p line_of_code_pos -v sidlevel -l lpp_name -a assoc_fid  
{ [ -d detail_data_item[,detail_data_item,...]  
-x detail_data_type[,detail_data_type,...]  
-y detail_data_len[,detail_data_len,...] ] |  
[-f detail_data_file] } -b BSD_syslog_message_text } | -h
```

Flags

-a

Contains the FFDC Failure Identifier for a failure condition reported by software used by this application which causes or influenced the condition being recorded at this time. This identifier should have been returned to this application as part of the software's result indication. The caller provides this identifier here so that the FFDC Error Stack can associate the failure report it is making at this time with the previously recorded failure report. This permits problem investigators to trace the cause of a failure from its various symptoms in this application and others to the root cause in the other software. If no other software failure is responsible for this condition, or if the other software did not return an FFDC Failure Identifier as part of its result information, this option should be omitted.

-b

Specifies the text message to be written to the BSD System Log.

-d

One or more data items that provides detailed information on the condition, used to provide the Detail Data in the AIX Error Log entry. If details of the information are too lengthy, these details can be written to a file, and the name of that file provided as the *detail_data_file* parameter. If a detail data file name is provided, this option should be omitted. If neither the *detail_data* or the *detail_data_file* parameters are provided or appear valid, null information will be recorded for the detail data in the AIX Error Log.

More than one data item may be provided with this option. Each data item must be separated by commas (,) with no intervening white-space characters. If a data item has imbedded whitespace characters, the data item must be enclosed in double quotes ("). The data items themselves must not contain commas (,), as the command interprets commas as field separators.

This option *must* be accompanied by the **-x** and **-y** options.

-e

Specifies the FFDC Log Event Type. Current valid values are FFDC_EMERG, FFDC_ERROR, FFDC_STATE, FFDC_TRACE, FFDC_RECOV, and FFDC_DEBUG. This code gives a general description of the type of event being logged (emergency condition, permanent condition, informational notification, debugging information, etc.) and the severity of the condition. If this option is not specified, the event type FFDC_DEBUG is assigned to this incident record.

-f

Name of a file containing details about the condition being reported. This option is used when the details are too lengthy to record within the remaining 100 bytes of Detail Data information left to the application by **fclogerr**, or when a utility exists that can analyze the detail information. The

contents of this file is copied to the **/var/adm/ffdc/dumps** directory, and the file's new location is recorded as the Detail Data in the AIX Error Log entry.

-h

Displays a help message to standard output and exits. No other processing is performed, regardless of the options specified.

-i

Specifies the absolute path name of the header file (.h) that contains the error logging template identification number that corresponds to the *error_template_label* specified in the **-l** option. This template must also be found in the node's error logging template repository (**/var/adm/ras/errtmpl**). This header file was generated by the **errupdate** command as part of the source code's building procedures, and should have been included in the LPP's packaging to be installed on the node with the software. If this option is not specified or the header file cannot be found when the script is executed, **fclogerr** will record the failure information using its own default error template (label **FFDC_DEF_TPLT_TR**, identifier code **2B4F5CAB**).

-l

Specifies an abbreviation of the name of the licensed programming product in which this software was shipped. This value should be recognizable to both customer and application-support services as an acceptable name for the LPP. Examples of such values are: **PSSP**, **GPFS**, **LoadLeveler**, and **RSCT**. If this option is not provided or appears invalid, the character string **PPS_PRODUCT** is used.

-p

Specifies the line of code location within the source code module where the condition is being reported. The value provided must be a valid integer value. To allow for proper identification and location of the condition, this value should be as close to the line of code that detected the condition as possible. Korn Shell scripts can use the value of **\$LINENO**. Script languages that do not provide a special line count variable can provide a symbolic value here that a developer can use to locate the spot in the source code where **fclogerr** is being used. If this option is not valid or not provided, the value of **0** is used.

-q

Suppresses the generation of warning messages from the command. Warning are generated when the command must substitute default information for missing information, or when the command is unable to copy the *detail_data_file* to the **/var/adm/ffdc/dumps** directory.

-r

Specifies the software component name. This is a symbolic name for the software making the report and should be a name recognizable to both customer and application-support services. The character string is limited to 16 characters.

-s

Specifies the name of the source file containing the line of code that encountered the condition being reported. For Korn and Bourne Shell scripts, the argument to this option should be set to **\$0**; C Shell scripts would set this argument to **\${0}**. If this option is not provided or not valid, the character string **unknown_file** is used.

-t

Indicates the symbolic label given to the AIX Error Logging template in the error log repository. The **errupdate** command that builds error logging templates creates a macro that maps this label to an integer code. This label begins with the characters **ERRID_** and is a maximum of 19 characters. If this option is not specified or the header file cannot be found when the script is executed, **fclogerr** will invoke the **errlogger** to create a message in the AIX Error Log using the **OPMSG** template.

-v

Indicates the SCCS version number of the source code module that detected the condition being recorded. For source code built under SCCS control, this should be set to "%!%" (the double-quotes are necessary). If this option is not provided or is not valid, the character string **unknown** is used.

-x

Indicates how the data items specified by the **-d** option are to be interpreted when recording this information to the AIX Error Log. These types must agree with the corresponding fields of the AIX Error Logging template specified in the **-t** option. Each type indicates how the corresponding data item in the **-d** list is interpreted. Acceptable values for this option are ALPHA, HEX, and DEC. There must be a matching type listed in the **-x** argument for each argument in the **-d** list.

This option *must* be supplied if the **-d** option is provided.

-y

Indicates the length of the data items (in bytes) specified by the **-d** option. These lengths must agree with the corresponding fields of the AIX Error Logging template specified in the **-t** option. There must be a matching type listed in the **-y** argument for each argument in the **-d** list.

This option *must* be supplied if the **-d** option is provided.

Description

This interface is used by any script program that wishes to record information to the AIX Error Log and the BSD System Log. The information written to this device is intended for use by the system administrator or operator to determine what failure conditions or other noteworthy conditions have occurred on the system that require attention. The purpose of the AIX Error Log and the BSD System Log is to record enough information about a condition so that the nature, impact, and response to the condition can be determined from the report, without requiring a recreation of the condition to detect what condition occurred and where. Any software that encounters permanent failure conditions that will persist until some type of direct intervention occurs, or encounters a condition that should be brought to the attention of the system administrator, should use **fclogerr** to record this information in the AIX Error Log and the BSD System Log.

Scripts should establish a basic FFDC Environment or an FFDC Error Stack Environment before using **fclogerr**, either by creating or inheriting the environment. **fclogerr** records information to the AIX Error Log and the BSD System Log even if these environments are not established, but the interface will not be capable of generating an FFDC Failure Identifier unless one of these environments exists.

Processes designed to use the FFDC Error Stack can also make use of the **fclogerr** interface, and should make use of it if they encounter conditions that require administrator attention or intervention to resolve.

To ensure proper identification of the condition and the location at which it was encountered, the FFDC Policy recommends that **fclogerr** should be called in-line in the script's source code module and invoked as soon as the condition is detected. **fclogerr** will record source code file name and line of code information to assist in identifying and locating the source code that encountered the condition. **fclogerr** can be invoked by a subroutine or autoloading routine to record this information if this is necessary, provided that all location information and necessary failure detail information is made available to this external routine. The external recording routine must record the true location where incident was detected.

Although **fclogerr** reports information to both the AIX Error Log and the BSD System Log, different options must be provided to this interface for each recording device. The Detail Data information recorded to the AIX Error Log is not also recorded to the BSD System Log; BSD System Log information is provided through different command options. This may require the **fclogerr** user to duplicate some information in this call.

Return Values

fclogerr returns the following exit status codes upon successful completion:

0

Information successfully queued to be written to the AIX Error Log and the BSD System Log. An FFDC Failure Identifier for the record is displayed to standard output. The caller should capture standard output to obtain this value.

2

Help information displayed and processing ended.

12

No information recorded to the AIX Error Log, and no FFDC Failure Identifier is provided by the command. The command user provided an invalid option to this command.

On UNIX platforms other than AIX, **fclogerr** returns the following exit status codes when a failure occurs:

38

A record could not be made in the BSD System Log for this incident. The System Log is experiencing a failure condition. On AIX systems, a report was recorded to the AIX Error Log; on other systems, this should be considered a failure.

When **fclogerr** is provided with incomplete information, it substitutes default information for the missing information and attempts to make a record in the FFDC Error Stack. Warnings are generated in these cases, and warning messages are generated unless the **-q** option is specified. In cases where more than one warning condition was detected, the command returns an exit status code for the condition it considered the most severe. The following exit status codes are returned by **fclogerr** when warning conditions are detected:

10

The command user failed to provide the **-i** option to this command, or the header file named as the argument to the **-i** option could not be located. The command will record generic information to the AIX Error Log in this case, using the First Failure Data Capture default template (label FFDC_DEF_TPLT_TR, identifier code 2B4F5CAB).

26

Both a detailed data string and a detail data file were provided to this routine. The routine chose the detail data string and ignored the detail data file.

28

The name of the resource detecting the incident was not provided. The default resource name **ffdc** was substituted for the missing resource name.

29

At least one component of the detecting application information—source code file name, source code file version, LPP name, line of code position—was not provided. Default information was substituted for the missing information.

32

The file named in the *detail_data_file* parameter could not be copied to the **/var/adm/ffdc/dumps** directory. The FFDC Error Stack entry cites the original version of this file. Do not discard the original copy of this file.

33

The **-e** option was not specified, or did not specify a valid FFDC event type. The event type FFDC_DEBUG has been assigned to this incident record.

34

A message was not supplied in the *format* parameter. As a result, a generic message was recorded to the BSD System Log for this incident.

35

No detailed information was provided for this incident. Later problem analysis may be difficult without these details to indicate specifics on the incident.

36

The length of the detail data string was greater than the capacity of the AIX Error Log entry limit. Detail data was truncated to fit in the available space. Some information on the incident may have been lost in this truncation.

37

An FFDC Error Identifier could not be constructed for the report created by this routine. An FFDC Failure Identifier is not written to standard output, but information on the incident was recorded to the AIX Error Log and the BSD System Log.

38

A record could not be made in the BSD System Log for this incident. The System Log may not be enabled, or may be experiencing problems. On AIX systems, a report was recorded to the AIX Error Log; on other systems, this should be considered a failure.

Examples

For this example, a Korn Shell script attempts to access configuration information from a file. If this attempt fails, the code will record a failure to the AIX Error Log using the following template source code:

```
#!/mymesgcat.cat
+ SP_FFDCXEMPL_ER:
  Comment      = "Configuration Failed - Exiting"
  Class        = S
  Log          = true
  Report       = true
  Alert        = false
  Err_Type     = PERM
  Err_Desc     = {3, 10, "CONFIGURATION FAILURE - EXITING"}
  Prob_Causes  = E89B
  User_Causes  = E811
  User_Actions = 1056
  Fail_Causes  = E906, E915, F072, 108E
  Fail_Actions = {5, 14, "VERIFY USER HAS CORRECT PERMISSIONS TO ACCESS FILE"},
                 {5, 15, "VERIFY CONFIGURATION FILE"}
  Detail_Data  = 46, 00A2, ALPHA
  Detail_Data  = 42, EB2B, ALPHA
  Detail_Data  = 42, 0030, ALPHA
  Detail_Data  = 16, EB00, ALPHA
  Detail_Data  = 16, 0027, ALPHA
  Detail_Data  = 4, 8183, DEC
  Detail_Data  = 4, 8015, DEC
  Detail_Data  = 60, 8172, ALPHA
```

This definition yields the following AIX Error Logging Template:

```
LABEL:          ERRID_SP_FFDCXEMPL_ER
IDENTIFIER:     <calculated by errupdate during source code build>

Date/Time:     <filled in by AIX Error Log subsystem>
Sequence Number: <filled in by AIX Error Log subsystem>
Machine Id:    <filled in by AIX Error Log subsystem>
```

Node Id: <filled in by AIX Error Log subsystem>
Class: S
Type: PERM
Resource Name: <filled in by -r option to **fclogerr**>

Description
CONFIGURATION FAILURE - EXITING

Probable Causes
COULD NOT ACCESS CONFIGURATION FILE

User Causes
USER CORRUPTED THE CONFIGURATION DATABASE OR METHOD

Recommended Actions
RE-CREATE FILE

Failure Causes
COULD NOT ACCESS CONFIGURATION FILE
PERMISSIONS ERROR ACCESSING CONFIGURATION DATABASE
FILE READ ERROR
FILE IS CORRUPT

Recommended Actions
VERIFY USER HAS CORRECT PERMISSIONS TO ACCESS FILE
VERIFY CONFIGURATION FILE

Detail Data
DETECTING MODULE
<filled in by **fclogerr** options>
ERROR ID
<The FFDC Failure Identifier created by **fclogerr**>
REFERENCE CODE
<The -a option value to **fclogerr**>
FILE NAME
<Must be supplied as part of -d option list to **fclogerr**>
FUNCTION
<Must be supplied as part of -d option list to **fclogerr**>
RETURN CODE<Must be supplied as part of -d option list to **fclogerr**>
ERROR CODE AS DEFINED IN sys/errno.h
<Must be supplied as part of -d option list to **fclogerr**>
USER ID<Must be supplied as part of -d option list to **fclogerr**>

The first three Detail Data Fields are constructed by the **fclogerr** routine from information passed in the parameters. The remaining Detail Data must be supplied with the -d option, and the type of data supplied must be indicated by the -x option. The example source code segment below demonstrates how this is done, and how **fclogerr** is invoked to record the information in the AIX Error Log and the BSD System Log.

```
typeset CONFIG_FNAME
typeset INBUF
typeset MINUSDOPTS
typeset MINUSXOPTS
typeset MINUSYOPTS
typeset FID
integer MYCLIENT
integer RC
:
MYCLIENT=$$
CONFIG_FNAME="/configfile.bin"
exec 3< $CONFIG_FNAME
:
read -u3 INBUF
RC=$?
if ((RC != 0))
```

```

then
# Create Detail Data Memory Block for AIX Error Log Template
# Need to know the EXACT structure of the Template to do this correctly.
#   Field 1 - filled in by fc_log_error
#   Field 2 - filled in by fc_log_error
#   Field 3 - filled in by fc_log_error
#   Field 4 - name of configuration file being used - 16 bytes
#   Field 5 - name of function call that failed - 16 bytes
#   Field 6 - return code from failing function - 4 byte integer
#   Field 7 - errno from failing function call (unused) - 4 byte integer
#   Field 8 - user ID using this software - remaining space (62 bytes)
# This source code supplies fields 4 through 8 in the "-d" option, and
# describes the data types for each in the "-x" option.
MINUSDOPTS=$CONFIG_FNAME
MINUSXOPTS="ALPHA"
MINUSYOPTS="16"
MINUSDOPTS="$MINUSDOPTS,read"
MINUSXOPTS="$MINUSXOPTS,ALPHA"
MINUSYOPTS="$MINUSYOPTS,16"
MINUSDOPTS="$MINUSDOPTS,$RC"
MINUSXOPTS="$MINUSXOPTS,DEC"
MINUSYOPTS="$MINUSYOPTS,4"
MINUSDOPTS="$MINUSDOPTS,0"
MINUSXOPTS="$MINUSXOPTS,DEC"
MINUSYOPTS="$MINUSYOPTS,4"
MINUSDOPTS="$MINUSDOPTS,$MYCLIENT"
MINUSXOPTS="$MINUSXOPTS,DEC"
MINUSYOPTS="$MINUSYOPTS,60"
FID=$(fclogerr -e FFDC_ERROR -t ERRID_SP_FFDCXMPL_ER -i /usr/lpp/ssp/inc/
myprog.h -r myprog -s myprog.ksh -p $LINEPOS -v "%I%" -l PSSP -d $MINUSDOPTS -x
$MINUSXOPTS -y $MINUSYOPTS -b "myprog Configuration Failure - Exiting")
RC=$?
if ((RC == 0))
then
    fcdispfid $FID
    return 1
else
:
fi
fi

```

Now consider a slight variation on the above example, using the same AIX Error Logging template, but this time using an external command to obtain the configuration data from a file that this source code supplies. The command exits with a non-zero exit status and prints an FFDC Failure Identifier to standard output if it encounters any failure conditions. Also, to demonstrate the use of double-quotes in the **-d** list, the configuration file will have an embedded space in the name:

```

typeset CONFIG_FNAME
typeset INBUF
typeset MINUSDOPTS
typeset MINUSXOPTS
typeset MINUSYOPTS
typeset FID
typeset OUTPUT
integer MYCLIENT
integer RC
:
MYCLIENT=$$
CONFIG_FNAME="This is a test"
OUTPUT=$(configdabeast $CONFIG_FNAME)
RC=$?
if ((RC != 0))
then
# Create Detail Data Memory Block for AIX Error Log Template
# Need to know the EXACT structure of the Template to do this correctly.

```

```

# Field 1 - filled in by fc_log_error
# Field 2 - filled in by fc_log_error
# Field 3 - filled in by fc_log_error
# Field 4 - name of configuration file being used - 16 bytes
# Field 5 - name of function call that failed - 16 bytes
# Field 6 - return code from failing function - 4 byte integer
# Field 7 - errno from failing function call (unused) - 4 byte integer
# Field 8 - user ID using this software - remaining space (62 bytes)
# This source code supplied fields 4 through 8 in the "-d" option, and
# describes the data types for each in the "-x" option.
MINUSDOPTS="\\"$CONFIG_FNAME\\"
MINUSXOPTS="ALPHA"
MINUSYOPTS="16"
MINUSDOPTS="$MINUSDOPTS,configdabeast"
MINUSXOPTS="$MINUSXOPTS,ALPHA"
MINUSYOPTS="$MINUSYOPTS,16"
MINUSDOPTS="$MINUSDOPTS,$RC"
MINUSXOPTS="$MINUSXOPTS,DEC"
MINUSYOPTS="$MINUSYOPTS,4"
MINUSDOPTS="$MINUSDOPTS,0"
MINUSXOPTS="$MINUSXOPTS,DEC"
MINUSYOPTS="$MINUSYOPTS,4"
MINUSDOPTS="$MINUSDOPTS,$MYCLIENT"
MINUSXOPTS="$MINUSXOPTS,DEC"
MINUSYOPTS="$MINUSYOPTS,60"
FID=$(fclogerr -e FFDC_ERROR -t ERRID_SP_FFDCEMPL_ER -i /usr/lpp/ssp/inc/
myprog.h -r myprog -s myprog.ksh -p $LINEPOS -v "%I%" -l PSSP -d $MINUSDOPTS -x
$MINUSXOPTS -y $MINUSYOPTS -a $OUTPUT -b "myprog Configuration Failure - Exiting")
RC=$?
if ((RC == 0))
then
    fcdispfid $FID
    return 1
else
:
fi
fi

```

Implementation Specifics

This command is provided as part of the First Failure Data Capture option of IBM RS Cluster Technology (RSCT).

Related Information

"<rsct/ct_ffdc.h> Header File" on page 46, "fcinit Command" on page 74, "fcpushstk Command" on page 80, "fcdispfid Command" on page 95, "fcreport Command" on page 103, "fcdecode Command" on page 97, **errpt** command (part of Base AIX), "fc_log_error Subroutine" on page 66

fcdispfid Command

Purpose

Displays the First Failure Data Capture Failure Identifier (FFDC Failure Identifier) to the standard error device.

Syntax

```
/usr/sbin/rsct/bin/fcdispfid [-q] FFDC_Failure_ID | -h
```

Flags

-h Displays a help message to standard output and exits. No other processing is performed, regardless of the options specified.

-q

Suppresses warning messages from this command. If this option is not provided, this command will display messages when an invalid FFDC Failure Identifier is detected.

Parameters

FFDC_Failure_ID

Specifies an FFDC Failure Identifier. This is an identifier returned from a previous call to **fcpushstk** or **fclogerr**, and indicates an entry made to report a failure encountered by the script. This identifier is written to the standard error device using FFDC message **2615-000**.

Description

This command is used by scripts to display an FFDC Failure Identifier value to the standard error device. This interface is provided because script programs do not have a mechanism for passing data back to its client except through exit status codes, signals, standard output, and standard error. To accomplish the task of "passing back" an FFDC Failure Identifier to a client in such an environment, **fcdispfid** uses XPG/4 cataloged message number **2615-000** to display this information to the standard error device. Clients of the script can capture the standard error information, search for the specific message number, and obtain the FFDC Failure Identifier from the script.

The script must indicate that any FFDC Failure Identifiers generated by the script will be directed to the standard error device in the script's user documentation. The client cannot be expected to know this behavior by default.

Return Values

0

FFDC Failure Identifier displayed to standard error.

2

Help information displayed and processing ended.

12

No information written to the standard error device. An invalid option was specified.

27

No information written to the standard error device. The *FFDC_Failure_ID* argument does not appear to be in a valid format.

Examples

To display an FFDC Failure Identifier to the client through the standard output device:

```

FID=$(fclogerr -e FFDC_ERROR -t ERRID_SP_FFDCXEMPL_ER -i /usr/lpp/ssp/inc/
myprog.h -r myprog -s myprog.ksh -p $LINEPOS -v "%I%" -l PSSP -d $MINUSDOPTS -x
$MINUSXOPTS -y $MINUSYOPTS -b "myprog Configuration Failure - Exiting")
RC=$?
if ((RC == 0))
then
    fcdispfid $FID
    return 1
else
    :
fi

```

Implementation Specifics

This command is provided as part of the First Failure Data Capture option of IBM RS Cluster Technology (RSCT).

Related Information

“fcdecode Command” on page 97, “fcfilter Command” on page 99, “fclogerr Command” on page 87, “fcpushstk Command” on page 80, “fcreport Command” on page 103, “fcstkprt Command” on page 101, “fc_display_fid Subroutine” on page 55

fcdecode Command

Purpose

Translates a First Failure Data Capture (FFDC) Failure Identifier from its standard form into its component parts, displaying this information to the standard output device in human readable format.

Syntax

```
/usr/sbin/rsct/bin/fcdecode FFDC_Failure_ID[,FFDC_Failure_ID,...] | -h
```

Flags

-h Displays a help message to standard output and exits. No other processing is performed, regardless of the options specified.

Parameters

FFDC_Failure_ID

An FFDC Failure Identifier, returned from previous calls to the **fcpushstk** and **fclogerr** commands, or returned from previous calls to the **fc_push_stack** or **fc_log_error** programming interfaces. This identifier indicates an entry made to report a failure or other noteworthy incident. More than one FFDC Failure Identifier can be provided as an argument to this command, however, each identifier must be separated by a comma (,) with no intervening white space between the identifiers.

Description

fcdecode decodes the 42-character FFDC Failure Identifier into its component parts, and displays these parts in human readable format. The output of this command displays the following information, extracted from the FFDC Failure Identifier:

- The network address (in ASCII format) of the node where this report resides
- The time when this recording was made, expressed using the currently active time zone settings
- One of the following, depending on where the information is recorded:
- The AIX Error Log template ID used to make this recording, if the record was filed in the AIX Error Log on that node, or
- The name of the FFDC Error Stack file containing this recording, if the record was file in the FFDC Error Stack and the FFDC Error Stack resides on this node
- A suggested command that can be used to obtain the specific report associated with this FFDC Failure Identifier.

Return Values

fcdecode returns one of the following integer status codes upon completion:

0

FFDC Failure Identifier successfully decoded.

2

Help information displayed and processing ended.

10

An FFDC Failure Identifier was not provided as an argument to this command.

12

Invalid or unsupported option provided to this command.

27

No information written to the standard output device. The FFDC Failure Identifier argument was not valid.

Examples

The FFDC Failure Identifier is represented by a base-64 value, read from right to left. Each dot represents a leading zero. To decode the FFDC Failure Identifier **.3Iv04ZVVfvp.wtY0xRXQ7**.....into its component parts:

```
fcdecode .3Iv04ZVVfvp.wtY0xRXQ7.....
Information for First Failure Data Capture identifier
.3Iv04ZVVfvp.wtY0xRXQ7.....
Generated by the local system
Generated Thu Sep 3 11:40:17 1998 EDT
Recorded to the AIX Error Log using template 460bb505
To obtain the AIX Error Log information for this entry, issue
the following command on the local system:
TZ=EST5EDT errpt -a -j 460bb505 -s 0903114098 | more
Search this output for an AIX Error Log entry that contains
the following ERROR ID code:
.3Iv04ZVVfvp.wtY0xRXQ7.....
```

The same command run on a different node has the following results:

```
fcdecode .3Iv04ZVVfvp.wtY0xRXQ7.....
Information for First Failure Data Capture identifier
.3Iv04ZVVfvp.wtY0xRXQ7.....
Generated on a remote system with the following Internet address:
9.114.55.125
Generated Thu Sep 3 11:40:17 1998 EDT
Recorded to the AIX Error Log using template 460bb505
TZ=EST5EDT errpt -a -j 460bb505 -s 0903114098 | more
Search this output for an AIX Error Log entry that contains
the following ERROR ID code:
.3Iv04ZVVfvp.wtY0xRXQ7.....
```

Implementation Specifics

This command is provided as part of the First Failure Data Capture option of IBM RS Cluster Technology (RSCT).

Related Information

“fcdispfid Command” on page 95, “fcreport Command” on page 103, “fcstkrpt Command” on page 101

fcfilter Command

Purpose

Locates and displays any First Failure Data Capture (FFDC) Failure Identifiers in a file or in standard input. More than one file may be specified.

Syntax

```
/usr/sbin/rsct/bin/fcfilter [ file_name ] [ . . . ]
```

Parameters

file_name

The name of the file to be searched for an FFDC Failure Identifier. More than one file may be provided. If a file name is not provided, **fcfilter** reads from standard input.

Description

This command scans any files listed as arguments for First Failure Data Capture (FFDC) Failure Identifiers. If a file name is not provided as an argument, this command examines standard input for FFDC Failure Identifiers. If an FFDC Failure Identifier is detected, **fcfilter** displays the identifier to standard output on its own line.

fcfilter can be used by scripts to extract FFDC Failure Identifiers returned by child processes via the standard error device.

If **fcfilter** detects more than one FFDC Failure Identifier in the input, the command will display all FFDC Failure Identifiers found, each one on a separate output line.

Return Values

fcfilter returns the following integer status codes upon completion:

0

fcfilter completed its execution. This exit status does not necessarily mean that any FFDC Failure Identifiers were detected.

> 0

fcfilter was interrupted or stopped by a signal. The exit status is the integer value of the signal that stopped the command.

Examples

The FFDC Failure Identifier is represented by a base-64 value, read from right to left. Each dot represents a leading zero. To obtain the list of all FFDC Failure Identifiers generated by a run of the command *mycmd*:

```
mycmd 2> /tmp/errout
fcfilter /tmp/errout
/.00...JMr4r.p9E.xRXQ7.....
/.00...JMr4r.pMx.xRXQ7.....
```

To obtain the FFDC Failure Identifier from a child process in a parent script, the script can use the **fcfilter** command as follows:

```
RESULTS=$(mychild 2> /tmp/errout)
if (($? != 0))          # mychild ended in failure, get FFDC ID
then
```

```
        cat /tmp/errout | fcfilter | read FIRST_FFDCID
else
    rm -f /tmp/errout
fi
```

Implementation Specifics

This command is provided as part of the First Failure Data Capture option of IBM RS Cluster Technology (RSCT).

Related Information

“fcpushstk Command” on page 80, “fclogerr Command” on page 87, “fcdispfid Command” on page 95, “fcstkrpt Command” on page 101, “fcreport Command” on page 103, “fc_display_fid Subroutine” on page 55, “fc_log_error Subroutine” on page 66, “fc_push_stack Subroutine” on page 59

fcstkrpt Command

Purpose

Displays the contents of an FFDC Error Stack file.

Syntax

```
/usr/sbin/rsct/bin/fcstkrpt { [-a] [-p | -r]
                             {
                               -fFFDC_Failure_Identifier [-i] |
                               -sFFDC_Error_Stack_File_Name
                             }
                             } | [-h]
```

Flags

-a

Indicates that all information be displayed for entries in the FFDC Error Stack. The default action is to display the time stamp for the record and the description of the incident.

-f

Specifies the FFDC Failure Identifier to use in locating the FFDC Error Stack. **fcstkrpt** decodes the FFDC Failure Identifier, locates the FFDC Error Stack associated with that FFDC Failure Identifier, and processes the FFDC Error Stack. Only one FFDC Failure Identifier can be specified by this flag.

-h

Displays a help message to standard output and exits. No other processing is performed regardless of the options specified.

-i

Displays only the information associated with the specific failure report identified by the **-f** flag. By default, all records in the FFDC Error Stack are displayed.

-p

Displays information from the FFDC Error Stack by process orientation. The output is ordered so that it reflects the order in which the processes were created (parent-child process relationship). Child process information is shown first, followed by parent process information. This view is used to understand which incidents occurred first, and which incidents occurred later because of them.

-r

Displays information from the FFDC Error Stack by incident relationships. Incidents are presented along with those incidents that are related to them. This view is used to understand which incidents occurred because of the occurrence of other incidents. This is the default.

-s

Specifies the name of the FFDC Error Stack to be examined. This name may be either the absolute or relative path name of the FFDC Error Stack. Only one FFDC Error Stack file name can be specified by this flag. If a relative file name is used, the file is assumed to be located in the **/var/adm/ffdc/stacks** directory of the node where the file resides.

Description

fcstkrpt reads an existing FFDC Error Stack file and displays its contents to the standard output device. The FFDC Error Stack file is indicated either through the name of the file itself, or by using an FFDC Failure Identifier that references a specific record within that file.

Information from the FFDC Error Stack can be displayed in one of two formats: by *related failure conditions* (the default) or by *software layer*.

Return Values

fcstkprpt issues the following integer exit status codes upon completion:

0

FFDC Error Stack file successfully located, and contents displayed to the standard output device.

2

Help information displayed and processing ended.

12

An invalid option was specified.

14

No information written to the standard output device. The **-f** option was used and the *FFDC Error Identifier* argument was not valid.

20

No information written to the standard output device. The **-s** option was used and the *FFDC Error Stack File* argument was not found.

27

No information written to the standard output device. The caller provided a valid *FFDC Failure Identifier*, but the file referenced by the FFDC Failure Identifier was not recorded on this node. Use the **fcdecode** command to locate the node where this FFDC Error Stack resides.

81

No information written to the standard output device. A failure occurred while writing information to standard output. The application should conclude that standard output cannot accept output.

85

No information written to the standard output device. The caller provided a valid FFDC Failure Identifier, but the file referenced by the FFDC Failure Identifier does not exist.

Examples

To obtain a brief report of the information stored in the FFDC Error Stack file **/var/adm/ffdc/stacks/myprog.562.19981001143052**:

```
$ fcstkprpt -r -s myprog.562.19981001143052
```

To obtain a detailed report of the information contained in the FFDC Error Stack where the FFDC Failure Identifier **.3Iv04ZVVfvp.wtY0xRXQ7.....** was recorded, and present this information in parent-child ordering:

```
$ fcstkprpt -p -f .3Iv04ZVVfvp.wtY0xRXQ7.....
```

Implementation Specifics

This command is provided as part of the First Failure Data Capture option of IBM RS Cluster Technology (RSCT).

Related Information

“fcclear Command” on page 105, “fcdecode Command” on page 97, “fcdispfid Command” on page 95, “fcfilter Command” on page 99, “fcpushstk Command” on page 80, “fcreport Command” on page 103, “fc_push_stack Subroutine” on page 59

fcreport Command

Purpose

Locates and displays the report of a failure and any failures associated with the failure.

Syntax

```
/usr/sbin/rsct/bin/fcreport { [-a] FFDC_Failure_ID } | -h
```

Flags

-a

Displays all information contained in a report for a failure. The default is to display the network address of the node where the failure report was generated, the time stamp on the failure report, and the description of the incident recorded in the failure report.

-h

Displays a help message to standard output and exits. No other processing is performed, regardless of the options specified.

Parameters

FFDC_Failure_ID

Specifies the FFDC Failure Identifier of the failure to begin the report. **fcreport** will attempt to obtain the failure information for this failure, as well as any failures that this report lists as an associated failure. Only one FFDC Failure Identifier may be provided to this command.

Description

fcreport decodes an FFDC Failure Identifier, and obtains reports on the failure identified by it. The command also detects if any failure was associated with the FFDC Failure Identifier, and if so, obtains the report on that failure. The command continues to examine the report of each failure it locates for associated failures and to obtain reports on the associated failures until one of the following conditions is met:

- No further associated failures are detected.
- The report for an associated failure cannot be found. This may occur when the associated failure report resides on a remote node that cannot be reached at the moment, or the record of the failure has been removed from the node where it resided.

Using this command, the user can obtain a report for the entire list of failures that caused a specific failure. **fcreport** is not capable of locating reports for any failures that may have been caused by the initial failure provided to the command; it can only obtain reports of failures that caused this failure.

Return Values

fcreport generates one of the following exit status codes upon completion:

0

Failure report located and displayed for the FFDC Failure Identifier provided. Zero or more related failure reports may have been located and displayed as well.

2

Help information displayed and processing ended.

10

Required options or arguments are not provided.

11

The FFDC Failure Identifier provided to this command was generated by a later release of the FFDC software. The command is not capable of correctly interpreting this identifier.

12

Unknown option specified to this command.

20

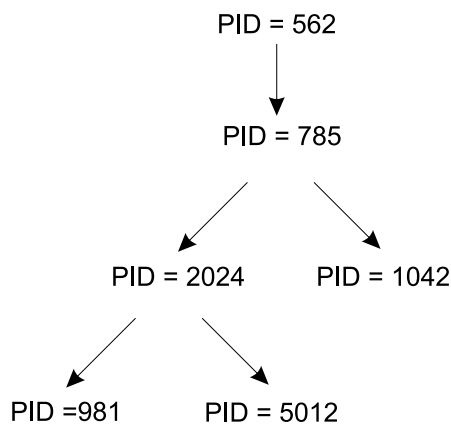
The FFDC Failure Identifier refers to an entry made in an FFDC Error Stack on this system, but the FFDC Error Stack file cannot be accessed. The file may have been removed, or permissions may have been altered on the file to prevent access to it.

27

The FFDC Failure Identifier provided to this command is not a valid identifier.

Examples

Consider the case where the several processes were created in the following parent-child order:



In this example, process 785 generated the FFDC Failure Identifier `.3Iv04ZVVfvp.wtY0xRXQ7.....` and passed it back to Process 562. To obtain a detailed report for FFDC Failure Identifier `.3Iv04ZVVfvp.wtY0xRXQ7.....` and any previous failures that led to this specific failure:

```
$ fcreport -a .3Iv04ZVVfvp.wtY0xRXQ7.....
```

This report will contain the details of the specified FFDC Failure Identifier, as well as any failures in processes 2024, 1042, 981, and 5012 that may have caused it. The report will not contain any failures in process 562 that may have been caused as a result of process 785's failure.

Security

fcreport uses **rsh** to obtain failure reports that may reside on remote nodes. The user must have sufficient privilege to execute **rsh** commands to these remote nodes. If the user does not have this permission, **fcreport** can only trace the list of related failures so long as they exist on the local node.

Implementation Specifics

This command is provided as part of the First Failure Data Capture option of IBM RS Cluster Technology (RSCT).

Related Information

"**fcclear** Command" on page 105, "**fcdecode** Command" on page 97, "**fcdispid** Command" on page 95, "**fcfilter** Command" on page 99, "**fclogerr** Command" on page 87, "**fcpushstk** Command" on page 80, "**fcstkprt** Command" on page 101, "**fc_push_stack** Subroutine" on page 59, "**fc_log_error** Subroutine" on page 66

fcclear Command

Purpose

Removes FFDC Error Stacks and detail data files from the local node.

Syntax

```
/usr/sbin/rsct/bin/fcclear -h |
{
  [-d filename[,filename,...]]
  [-D filename[,filename,...]]
  [-f FFDC_Failure_ID[,FFDC_Failure_ID,...]]
  [-F FFDC_Failure_ID[,FFDC_Failure_ID,...]]
  [-s filename[,filename,...]]
  [-S filename[,filename,...]]
  [-t days]
}
```

Flags

-d

Removes detail data files by specifying a list of one or more detail data file names. These file names may be absolute path names, or relative to the **/var/adm/ffdc/dumps** directory. These files are removed if they exist on the local node. Files on remote nodes cannot be removed through this command. If more than one file name is provided, they must be separated by a comma (,) without any intervening white space.

-D

Preserves detail data files by specifying a list of one or more detail data file names. These file names may be absolute path names, or relative to the **/var/adm/ffdc/dumps** directory. These files are retained if they exist on the local node. Files on remote nodes cannot be retained through this command. If more than one file name is provided, they must be separated by a comma (,) without any intervening white space.

-f

Removes FFDC Error Stack files by specifying a list of one or more FFDC Failure Identifiers. The FFDC Error Stacks associated with these FFDC Error Identifiers are located and removed if they are present on the local node. FFDC Error Stacks on remote nodes will not be removed. If more than one FFDC Failure Identifier is supplied, they must be separated by a comma (,) with no intervening white space.

-F

Preserves FFDC Error Stack files by specifying a list of one or more FFDC Failure Identifiers. The FFDC Error Stacks associated with these FFDC Error Identifiers are located and retained if they are present on the local node. FFDC Error Stacks on remote nodes will not be retained. If more than one FFDC Failure Identifier is supplied, they must be separated by a comma (,) with no intervening white space.

-h

Displays help and usage information to the standard output device. No other processing is performed.

-s

Removes FFDC Error Stack files by specifying a list of one or more FFDC Error Stack file names. These file names can be absolute path names or file names relative to the **/var/adm/ffdc/stacks** directory. These files are removed if they exist on the local node. FFDC Error Stacks on remote nodes cannot be removed through this command. If more than one file name is provided, each must be separated by a comma (,) without any intervening white space.

-s

Removes FFDC Error Stack files by specifying a list of one or more FFDC Error Stack file names. These file names can be absolute path names or file names relative to the **/var/adm/ffdc/stacks** directory. These files are removed if they exist on the local node. FFDC Error Stacks on remote nodes cannot be removed through this command. If more than one file name is provided, each must be separated by a comma (,) without any intervening white space.

-t

Indicates that FFDC Error Stacks and detail data files that are older than a specific number of days should be removed from the local node. This selection criteria is independent of the other selection criteria.

Description

fcclear is used to remove FFDC Error Stack files that are no longer needed for problem determination efforts from the local node. Specific FFDC Error Stack files can be removed, as well as FFDC Error Stack files containing the records of specific FFDC Failure Identifiers. Individual entries within an FFDC Error Stack cannot be removed.

Using the **-t** option, **fcclear** can be used to remove FFDC Error Stack files older than a specific number of days. To use **fcclear** in an automatic fashion to clean out unneeded FFDC Error Stacks, see the **cron** command for automating the execution of commands.

To remove all FFDC Error Stacks from the local node, specify a value of zero (0) for the number of days option argument.

Return Values

fcclear generates the following exit status values upon completion:

0

Successful completion of the command. The command may complete successfully if no FFDC Error Stack files or detail data files match the selection criteria.

2

Help information successfully displayed. No further processing is performed.

10

No files are removed from the local system. A required option was not specified to this command.

11

No files are removed from the local system. The argument of the **-t** option is not numeric.

12

No files are removed from the local system. Unknown option specified by the caller.

19

The directory **/var/adm/ffdc/stacks** does not exist or is not mounted.

26

No files are removed from the local system. The same option was specified more than once.

28

No files were removed from the system. The caller provided options that instruct the command to both remove and retain the same file. This condition can occur when the command user specified an FFDC Failure Identifier that is recorded in an FFDC Error Stack file specified by name to this command.

Examples

To remove any FFDC Error Stack and detail data files older than seven days from the local node:

```
fcclear -t 7
```

To remove all FFDC Error Stack and detail data files older than seven days, but retain the FFDC Error Stack that contains information for the FFDC Failure Identifier **/3Iv04ZVVfvp.wtY0xRXQ7.....**, issue the command:

```
fcclear -t 7 -F /3Iv04ZVVfvp.wtY0xRXQ7.....
```

To remove the FFDC Error Stack file that contains the record for the FFDC Failure Identifier **/3Iv04ZVVfvp.wtY0xRXQ7.....**, issue the command:

```
fcclear -f /3Iv04ZVVfvp.wtY0xRXQ7.....
```

To remove the FFDC Error Stack files **myprog.14528.19990204134809** and **a.out.5134.19990130093256** from the system, plus the detail data file **myprog.14528.19990204135227**:

```
fcclear -s myprog.14528.19990204134809,a.out.5134.19990130093256  
-d myprog.14528.19990204135227
```

To extend the previous command to remove the named files plus any FFDC Error Stack and detail data files older than 14 days:

```
fcclear -s myprog.14528.19990204134809,a.out.5134.19990130093256  
-d myprog.14528.19990204135227 -t 14
```

Implementation Specifics

This command is provided as part of the First Failure Data Capture option of IBM RS Cluster Technology (RSCT).

Related Information

“fccheck Command” on page 108, “fcreport Command” on page 103, “fcstkrpt Command” on page 101

fccheck Command

Purpose

Performs basic problem determination on the First Failure Data Capture utilities.

Syntax

```
/usr/sbin/rsct/bin/fccheck [-q] | [-h]
```

Flags

-h

Displays help and usage information to standard output. No other processing is performed.

-q

Specified "quiet" mode. The command does not display the results of each test to standard output. The exit status of the command must be used to determine the results of the tests. If more than one condition was detected, the exit status will reflect the most severe condition detected by **fccheck**.

Description

fccheck perform basic problem determination for the First Failure Data Capture utilities. The command checks for the following conditions and information on the local node:

- Checks if FFDC Error Stack usage has been disabled in the current process environment.
- Obtains the IP address that would be currently used by FFDC to identify the local node.
- Checks if **/var/adm/ffdc/stacks** is available, and if so, how much space is available in the file system where the directory resides. Checks to see if there is insufficient space to create FFDC Error Stacks.
- Checks if **/var/adm/ffdc/dumps** is available, and if so, how much space is available in the file system where the directory resides.

Results of these tests are displayed to standard output unless the "quiet" option has been specified.

fccheck sets an exist status value to indicate the most severe condition it detected during the execution of its tests.

Return Values

The following integer exit status codes can be generated by this command:

0

All conditions tested by **fccheck** were found to be in normal operational parameters.

2

Help information successfully displayed. No further processing is performed.

12

No checking performed. Invalid option specified to this command.

19

The directory **/var/adm/ffdc/stacks** is not mounted or does not exist.

20

Cannot access or examine one or more directories in the path **/var/adm/ffdc/stacks**. Permissions may have been changed on one or more of the directories in this path to prevent access.

24

Cannot access or examine one or more directories in the path **/var/adm/ffdc/dumps**. Permissions may have been changed on one or more of the directories in this path to prevent access.

32

The directory **/var/adm/ffdc/dumps** is not mounted or does not exist.

40

Insufficient space is available in the **/var/adm/ffdc/stacks** directory to create FFDC Error Stacks on the local node.

41

Unable to obtain file system information from the operating system. This indicates a potential problem with the operating system itself.

42

FFDC Error Stack creation and usage has been disabled in this process environment.

Examples

To check for possible problems with the FFDC utilities on the local node:

```
fccheck
fccheck Status: All tests completed
```

If the local node had disabled the creation of FFDC Error Stacks, **fccheck** would indicate this as a problem:

```
fccheck

fccheck Status: Creation and use of FFDC Error Stacks has been expressly disabled in the current
execution environment. Any processes created in the current execution environment cannot create their
own FFDC Error Stacks or inherit use of existing FFDC Error Stacks.

fccheck Status: All checks completed. Examine the previous status output for possible FFDC problem
conditions and take the recommended actions listed in these messages.
```

Implementation Specifics

This command is provided as part of the First Failure Data Capture option of IBM RS Cluster Technology (RSCT).

Related Information

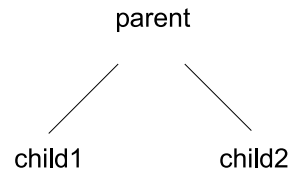
“fcclear Command” on page 105, “fcinit Command” on page 74

Appendix A. Examples

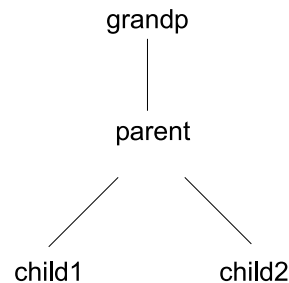
This appendix contains two extended listings that demonstrate the First Failure Data Capture (FFDC) application: the parent/child source-code examples and the daemon source-code example.

Parent/Child Source-Code Examples

The source code modules in this appendix make use of the First Failure Data Capture (FFDC) application programming interfaces and command line interfaces to record information about failure conditions for use in later problem analysis. Originally, these modules were designed to be executed as follows:



At some point after these modules were written, a new module was added to the example. This update caused the programs to be executed as follows:



The example demonstrates these concepts:

- Using an FFDC Error Stack to capture failure information for a related set of processes that are executed in parent-child relationships
- Creating and inheriting an FFDC Environment, and when a process would choose to do either
- How creation requests are handled after an FFDC Environment has been established by an ancestor process

The parent.sh Script

```
#!/bin/ksh
# IBM_PROLOG_BEGIN_TAG
# This is an automatically generated prolog.
#
#
# Licensed Materials - Property of IBM
#
# Restricted Materials of IBM
#
# (C) COPYRIGHT International Business Machines Corp. 1999
# All Rights Reserved
#
# US Government Users Restricted Rights - Use, duplication or
# disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
#
# IBM_PROLOG_END_TAG
# "@(#)12 1.1 src/rsct/ffdc/policycode/parent.sh, ffdc, rsct_rmoh 4/19/99 16:37:36"
#-----
# Module Name:      parent
```

```

#
# Purpose:   Used to demonstrate the use of First Failure Data Capture
#           (FFDC) command line interfaces from scripts. The key concepts
#           demonstrated by this script are:
#           - The establishment of an FFDC Environment that makes
#             use of an FFDC Error Stack.
#           - Recording information to the FFDC Error Stack when
#             a failure is detected.
#           - Returning information to the script's client to
#             indicate that a failure occurred, and where the
#             failure information is recorded.
#
# Design:   "parent" expects to be the top-most process in a
#           parent/child/descendent process relationship. It creates two
#           child processes, "child1" and "child2". The success or failure
#           of "parent" is dependent upon the success or failure of these
#           child processes. To make it easier for a problem investigator
#           to determine why "parent" failed, it will create an FFDC
#           Environment that makes use of an FFDC Error Stack. This will
#           permit "child1" and "child2" to record failure information to
#           the FFDC Error Stack if they so choose, and "parent" can record
#           its own related failure condition as well. By recording the
#           failure information to the FFDC Error Stack, problem investi-
#           gators can determine the initial failure condition, and also
#           understand the related failures that occurred because of it.
#
# Usage:   parent -n <name>
#
# Exit Status:   0   "parent" completed successfully
#               1   "parent" not invoked correctly
#               2   "parent" failed - an FFDC Failure Identifier is
#                   displayed to inform the client where the failure
#                   information resides, if the FFDC Environment could be
#                   established
#
# Notes:   For the sake of brevity in this example, "parent" does not
#           conform to some specific RS/6000 SP lab software policies,
#           such as the NLS policy for output messages. It is believed
#           that clarity and brevity of the example is preferred over
#           forcing the code example to conform to every lab policy.
#-----

```

```

#-----
# Function:   fcinit_rc_check
# Description: Check exit status of the fcinit command, display any
#             appropriate messages, and take appropriate action. Terminate
#             the script in cases where a coding error is evidenced.
# Parameters: $1   Exit status from fcinit
#-----
function fcinit_rc_check
{
    case $1 in
        42) print "$0 Warning: FFDC Error Stack use has been"
            print "\t disabled by the system administrator."
            ;;
        19) print "$0 Warning: The FFDC Error Stack directory"
            print "\t does not exist - contact the system"
            print "\t administrator and report the problem."
            ;;
        13) print "$0 Warning: Cannot reserve an FFDC Error"
            print "\t Stack for use by this process, because"
            print "\t the file that would be used to store"
            print "\t the stack already exists."
            ;;
        17) print "$0 Warning: Could not establish an FFDC"
            print "\t Environment because the process environment"
            print "\t appears to be corrupted."
    esac
}

```

```

;;
18) print "$0 Warning: Could not establish an FFDC"
    print "\t Environment because memory could not be"
    print "\t allocated. Check the system for processes"
    print "\t that are hoarding memory."
;;
16) print "$0 Warning: Could not establish an FFDC"
    print "\t Environment because fcinit could not modify"
    print "\t the process's environment."
;;
99) print "$0 Warning: Unexpected failure in the fcinit"
    print "\t command. Contact the system administrator"
    print "\t to have problem determination performed."
;;
*) print "$0 Warning: fcinit failed with exit status of"
    print "\t $1. The command has been used incorrectly."
    print "\t Check the fcinit man page for an explanation"
    print "\t of this failure and modify this script to"
    print "\t use fcinit correctly."
    exit 2
;;
    esac
}

#-----
# Function:  fcpushstk_rc_check
# Description:  Check exit status of the fcpushstk command, display any
#               appropriate messages
# Parameters:  $1  Exit status from fcpushstk
# Return Code:  0   "Acceptable" failure - the return code was a warning
#               that incomplete information was provided to fcpushstk
#               1   Failure - could not record to the FFDC Error Stack
#-----
function fcpushstk_rc_check
{
    integer FRC
    case $1 in
        #
        # Failures that did not prevent recording to the FFDC Error
        # Stack are tested in this section.
        #
        28) print "$0 Warning: Resource name is missing. Fix the"
            print "\t script to pass the resource name with -r."
            FRC=0
            ;;
        30) print "$0 Warning: Default description message is"
            print "\t missing. Fix the script to pass the default"
            print "\t message to the fcpushstk command."
            FRC=0
            ;;
        29) print "$0 Warning: File name or location information is"
            print "\t missing. Fix the script to pass the location"
            print "\t information to the fcpushstk command."
            FRC=0
            ;;
        35) print "$0 Warning: Failure detail information was not"
            print "\t provided, or was NULL."
            FRC=0
            ;;
        37) print "$0 Warning: fcpushstk could not create an FFDC"
            print "\t Failure Identifier for the failure report."
            FRC=0
            ;;
        #
        # Failures that prevented a recording to the FFDC Error Stack
        # are tested in this section.
        #
    esac
}

```

```

17) print "$0 Warning: Cannot record information to the"
    print "\t FFDC Error Stack because the process's"
    print "\t environment appears to be corrupted."
    FRC=1
    ;;
19) print "$0 Warning: Cannot record information to the"
    print "\t FFDC Error Stack because the FFDC Error"
    print "\t Stack directory does not exist. Contact"
    print "\t the system administrator and report the"
    print "\t problem."
    FRC=1
    ;;
20) print "$0 Warning: Cannot record information to the"
    print "\t FFDC Error Stack because access to the FFDC"
    print "\t Error Stack directory is denied to this"
    print "\t script. Contact the system administrator"
    print "\t and report the problem."
    FRC=1
    ;;
24) print "$0 Warning: Cannot record information to the"
    print "\t FFDC Error Stack because the FFDC Error Stack"
    print "\t contents appear to be corrupted. This may"
    print "\t happen if another process accidentally writes"
    print "\t data to the file storing the FFDC Error"
    print "\t Stack."
    FRC=1
    ;;
25) print "$0 Warning: Cannot record information to the"
    print "\t FFDC Error Stack because a file has been"
    print "\t created using the directory name. Contact"
    print "\t the system administrator and report the"
    print "\t problem."
    FRC=1
    ;;
40) print "$0 Warning: Cannot record information to the"
    print "\t FFDC Error Stack because there is not enough"
    print "\t space in the FFDC Error Stack directory's"
    print "\t file system. Contact the system"
    print "\t administrator and report the problem."
    FRC=1
    ;;
41) print "$0 Warning: Cannot record information to the"
    print "\t FFDC Error Stack because the FFDC Error Stack"
    print "\t contents appear to be corrupted. This may"
    print "\t happen if another process accidentally writes"
    print "\t data to the file storing the FFDC Error"
    print "\t Stack."
    FRC=1
    ;;
99) print "$0 Warning: Unexpected failure in the fcpushstk"
    print "\t command. Contact the system administrator"
    print "\t to have problem determination performed."
    FRC=1
    ;;
*) print "$0 Warning: fcpushstk failed with exit status of"
    print "\t $1. The command has been used incorrectly."
    print "\t Check the fcpushstk man page for an"
    print "\t explanation of this failure and modify this"
    print "\t script to use fcinit correctly."
    return 1
    ;;
esac
return $FRC
}

#-----
# -- MAINLINE CODE -- MAINLINE CODE -- MAINLINE CODE -- MAINLINE CODE --

```

```

#-----
typeset PARAMS
typeset OPTION
typeset NAME
typeset INFO
typeset RESULTS
typeset FID
typeset FID_MSG
typeset MSG_NUM
integer RC
integer NRC
integer FID_FAILED
integer I
integer NUM_PARAMS
#
# Save input parameters to special variables. This is done because the script
# will later invoke fcinit.sh in the current shell, which may overwrite the
# input parameter variables.
#
PARAMS=$*
NUM_PARAMS=$#
OPTION=$1
NAME=$2
#
# Time to establish the FFDC Environment to capture failures detected by this
# script and any descendent process it creates. Because "parent" expects to be
# the top-most process in this process hierarchy (because it probably expects
# to be used from the command line itself), "parent" will CREATE the FFDC
# Environment, not INHERIT it. It does so because it does not expect that its
# parent process would have established an FFDC Environment previously.
#
# "parent" may sometimes be invoked by a parent process that has previously
# established an FFDC Environment. For example, the user sets up a script
# to execute "parent" and a number of other processes, and wants to capture
# failure information for these processes. Will the CREATE request made by
# "parent" cause problems in this case? No. The FFDC commands will recognize
# that an FFDC Environment was established by the "parent"'s parent process.
# In that case, the FFDC commands will convert "parent"'s CREATE request into
# an INHERIT request automatically, and let "parent" know by setting a special
# exit status code in this case (1).
#
# Note that the fcinit.sh command is "sourced" from this script, meaning that it
# is executed within the script's own environment.
#
# ----
# NOTE
# ----
# The reader of this example might question why the FFDC Environment was not
# established after the testing the input parameters. This is a valid question.
# If "parent" had expected to be invoked by a parent process that previously
# established an FFDC Environment, then "parent" could decide to establish the
# FFDC Environment for itself FIRST and tested the input SECOND. By testing
# the input FIRST, "parent" would betrays its expectation that it will always
# be the top-most process. However, this would cause problems when "parent" is
# invoked incorrectly from another process, since the explanation for why
# "parent" failed would not be within the FFDC Error Stack. Problem investi-
# gators would not find any reason to explain "parent"'s failure, removing the
# benefits provided by the FFDC Error Stack.
#
# To ensure that "parent" will record information about incorrect usage to the
# FFDC Error Stack (to provide support for invoking "parent" from a script),
# "parent" will establish the FFDC Environment first.
#
. fcinit.sh -sc
RC=?
if ((RC != 0 && RC != 1))
then

```

```

    fcinit_rc_check $RC
fi
#
# Verify that the command was invoked correctly.  There are two failures that
# can occur in this case:
# - Incorrect number of parameters    Msg Set #1, Message #1
# - Unknown option specified          Msg Set #1, Message #2
# The error messages will be written to the FFDC Error Stack if the FFDC
# Environment was successfully established above.  The messages written to
# the FFDC Error Stack are contained in the mythical message catalog file
# "ffdexpl.cat".
#
if ((NUM_PARAMS != 2))
then
    #
    # This section tests to make sure that the FFDC Environment was
    # successfully established before trying to record to the FFDC Error
    # Stack.
    #
    fcteststk > /dev/null
    if (($? == 0))
    then
        FID_FAILED=0
        I=0
        while ((I < 4))
        do
            FID_MSG=$(fcpushstk -c ffdexpl.cat -m1 -n1 -l PSSP \
                -p$LINENO -r ffdexpl -s $0 -v "1.1" \
                -d "Command line parameters: \"$PARAMS\"" \
                "Script not invoked correctly")

            NRC=$?
            if ((NRC != 0))
            then
                if ((NRC == 22))
                then
                    #
                    # This can happen if multiple processes
                    # are trying to write into the same
                    # FFDC Error Stack.  If this happens
                    # four times, consider it unrecoverable.
                    # Four was just a number picked out of
                    # the air for this example.
                    #
                    ((I = I + 1))
                else
                    fcpushstk_rc_check $NRC
                    FID_FAILED=$?
                    I=4
                fi
            else
                I=4
            fi
        done
    else
        FID_FAILED=1
    fi
    #
    # If the FFDC Environment was not established, or a recording could
    # not be made to the FFDC Error Stack, display the error information.
    # We do this so that some record is made of the failure, even if it
    # is to the terminal.
    #
    if ((FID_FAILED == 1))
    then
        print "$0 Error: Script not invoked correctly."
        print "\t Usage:"
        print "\t\t parent -n <name>"
    fi

```

```

else
    print $FID_MSG | read MSGNUM FID
    fcdispfid $FID
fi
exit 1
fi
if [[ "$OPTION" != "-n" ]]
then
    fcteststk > /dev/null
    if (($? == 0))
    then
        FID_FAILED=0
        I=0
        while ((I < 4))
        do
            FID_MSG=$(fcpushstk -c ffdcxpl.cat -m1 -n2 -l PSSP \
                -p$LINENO -r ffdcxpl -s $0 -v "1.1" \
                -d "Invalid option detected: \"$OPTION\"" \
                "Invalid option specified")
            NRC=$?
            if ((NRC != 0))
            then
                if ((NRC == 22))
                then
                    ((I = I + 1))
                else
                    fcpushstk_rc_check $NRC
                    FID_FAILED=$?
                    I=4
                fi
            else
                I=4
            fi
        done
    else
        FID_FAILED=1
    fi
    if ((FID_FAILED == 1))
    then
        print "$0 Error: Invalid option specified: $OPTION"
        print "\t Usage:"
        print "\t\t parent -n <name>"
    else
        print $FID_MSG | read MSGNUM FID
        fcdispfid $FID
    fi
    exit 1
fi
#
# "parent" has been invoked correctly. Now create "child1" and wait for it
# to complete. If "child1" fails, this script cannot complete its function,
# and also fails. "parent" expects "child1" to provide an exit status of zero
# (0) in successful completions, and a non-zero exit status when failures
# occur.
#
# The developer of "parent" knows that "child1" is designed to provide its
# normal results to standard output. "child1" is also designed to report
# failure information to standard error if it detects a failure. The FFDC
# Failure Identifier is one of the data items that would be written to standard
# error. "parent" must therefore capture both standard output and standard
# error from "child1". When "parent" invokes "child1" below:
# - standard output is capture in the variable INFO
# - standard error is redirected to the file /tmp/parent.$$
#
rm -f /tmp/parent.$$
INFO=$(child1 -n $NAME 2> /tmp/parent.$$)
RC=$?

```

```

if ((RC != 0))
then
  fcteststk > /dev/null
  if (($? == 0))
  then
    #
    # Obtain the FFDC Failure Identifier from "child1"'s standard
    # error. This script assumes that the only information in
    # "child1"'s standard error is the FFDC Failure Identifier
    # message.
    #
    cat /tmp/parent.$$ | read MSGNUM FID
    #
    # Record "parent"'s failure to the FFDC Error Stack. Note the
    # use of the "-a" option to fcpushstk. This makes a link
    # between "child1"'s failure and the failure in "parent" which
    # it caused. "parent" records its failure using message number
    # 3 from message set 1 in the "ffdcexpl.cat" message catalog.
    #
    FID_FAILED=0
    I=0
    while ((I < 4))
    do
      FID_MSG=$(fcpushstk -c ffdcexpl.cat -m1 -n3 -l PSSP \
        -d "Failed cmd: child1, Exit Status: $RC" \
        -p$LINENO -r ffdcexpl -s $0 -v "1.1" \
        -a "$FID" "Child process failed - exiting")

      NRC=$?
      if ((NRC != 0))
      then
        if ((NRC == 22))
        then
          ((I = I + 1))
        else
          fcpushstk_rc_check $NRC
          FID_FAILED=$?
          I=4
        fi
      else
        I=4
      fi
    done
  else
    FID_FAILED=1
  fi
  if ((FID_FAILED == 1))
  then
    print "$0 Error: Child process failed - exiting"
    print "\t Failed command: child1, Exit Status: $RC"
  else
    print $FID_MSG | read MSGNUM FID
    fcdispfid $FID
  fi
  rm -f /tmp/parent.$$
  exit 2

fi
#
# "child1" has succeeded. Now create "child1" and wait for it
# to complete. If "child2" fails, this script cannot complete its function,
# and also fails. "parent" expects "child2" to provide an exit status of zero
# (0) in successful completions, and a non-zero exit status when failures
# occur.
#
# "child2" is designed to behave in the same manner as "child1": returning
# its normal results to standard output and failure information to standard
# error. "parent" captures this information as follows:

```

```

# - standard output is captured in the variable RESULTS
# - standard error is redirected to the file /tmp/parent.$$
#
rm -f /tmp/parent.$$
RESULTS=$(child2 -i $INFO 2> /tmp/parent.$$)
RC=$?
if ((RC != 0))
then
    fcteststk > /dev/null
    if (($? == 0))
    then
        cat /tmp/parent.$$ | read MSGNUM FID
        FID_FAILED=0
        I=0
        while ((I < 4))
        do
            FID_MSG=$(fcpushstk -c ffdcxpl.cat -m1 -n3 -l PSSP \
                -d "Failed cmd: child2, Exit Status: $RC" \
                -p$LINENO -r ffdcxpl -s $0 -v "1.1" \
                -a$FID "Child process failed - exiting")
            NRC=$?
            if ((NRC != 0))
            then
                if ((NRC == 22))
                then
                    ((I = I + 1))
                else
                    fcpushstk_rc_check $NRC
                    FID_FAILED=$?
                    I=4
                fi
            else
                I=4
            fi
        done
    else
        FID_FAILED=1
    fi
    if ((FID_FAILED == 1))
    then
        print "$0 Error: Child process failed - exiting"
        print "\t Failed command: child1, Exit Status: $RC"
    else
        print $FID_MSG | read MSGNUM FID
        fcdispfid $FID
    fi
    rm -f /tmp/parent.$$
    exit 2
fi

#
# All child processes have successfully completed. Generate the expected
# results and terminate the script.
#
print "$0: $RESULTS"
exit 0

```

The child1.c Program

```

/* IBM_PROLOG_BEGIN_TAG                                     */
/* This is an automatically generated prolog.              */
/*                                                         */
/*                                                         */
/*                                                         */
/* Licensed Materials - Property of IBM                     */
/*                                                         */

```

```

/* Restricted Materials of IBM */
/* */
/* (C) COPYRIGHT International Business Machines Corp. 1999 */
/* All Rights Reserved */
/* */
/* US Government Users Restricted Rights - Use, duplication or */
/* disclosure restricted by GSA ADP Schedule Contract with IBM Corp. */
/* */
/* IBM_PROLOG_END_TAG */
/* */
* Module Name: child1
*
* Purpose: Used to demonstrate the use of First Failure Data Capture
* (FFDC) application programming interface from a C language
* program. The key concepts demonstrated by this program are:
* - The establishment of an FFDC Environment that makes
* use of an FFDC Error Stack.
* - Recording information to the FFDC Error Stack or
* the AIX Error Log when a failure is detected
* - Returning information to the program's client to
* indicate that a failure occurred, and where the
* failure information is recorded.
*
* Compilation: This module must be linked to the following RCST libraries:
* libct_ffdc.a For FFDC function
* libct_cu.a For Cluster Utility Error Handling
* This module must also be linked to the "utils.o" object module.
*
* Design: "child1" expects to be called as one or possibly several
* child processes in a parent/child process relationship. To
* make it easier for a problem investigator to determine why
* "child1" failed, "child1" will establish an FFDC Environment
* for itself. It will anticipate that an FFDC Environment was
* established by its parent process, and therefore "child1" will
* INHERIT the existing FFDC Environment instead of CREATING its
* own. By recording failure information to the FFDC Error Stack,
* "child1" leaves persistent records of its failure conditions,
* and also makes it possible for its parent process to associate
* any related failure it might have with the failure in this
* process.
*
* "child1" will record any failure condition it detects to the
* FFDC Error Stack if an FFDC Environment is successfully
* inherited. If it cannot inherit the FFDC Environment, failure
* information is displayed to the standard output device. This
* is to ensure that at least some record of the failure exists,
* even if it is the terminal.
*
* "child1" also records one specific failure to the AIX Error
* Log: a failure in a resource that it makes available. Only this
* failure is written to the AIX Error Log because only this
* failure condition is one that needs to be brought to the
* attention of a system administrator. Other failure conditions
* do not critical enough to need the system administrator's
* immediate attention (though some may require the administrator's
* eventual attention), so they are recorded to the FFDC Error
* Stack instead.
*
* If "child1" experiences a failure and records a failure record,
* it will display the FFDC Failure Identifier to standard error
* using the fc_display_fid() API. The only other mechanism
* available to "child1" to inform its client of its outcome is the
* exit status code, and this is not sufficient for storing the
* FFDC Failure Identifier. Therefore, standard error will be used
* to convey the FFDC Failure Identifier to its client. The client
* must expect failure information to be returned in this fashion,
* and capture both the standard output and the standard error of

```

```

*      this command.
*
* Usage:      child1 -n <name>
*
* Exit Status:  0      Successful completion
*              1      Command not invoked correctly
*              2      Command failed - an FFDC Failure Identifier will be
*                    displayed to inform "parent" where the failure infor-
*                    mation resides, if the FFDC Environment was successfully
*                    established.
*
* Notes:      For the sake of brevity in this example, "child1" does not
*              conform to some specific RS/6000 SP lab software policies,
*              such as the NLS policy for output messages. It is believed
*              that clarity and brevity of the example is preferred over
*              forcing the code example to conform to every lab policy.
*/

#include <stdio.h>      /* Basic I/O capability      */
#include <errno.h>     /* Error code definitions  */
#include <memory.h>    /* Memory manipulation routines */
#include <unistd.h>    /* Command line option parsing */
#include <sys/vmount.h> /* File system mounting capa- */
                    /* bility (for sake of example) */
#include <rsct/ct_ffdc.h> /* FFDC definitions      */
#include <rsct/ct_cu.h> /* Cluster Error Handling  */
#include <ffdcexpl.err.S.h> /* AIX Error Log template header*/
                    /* generated by the build */
                    /* process */
#include "ffdcexpl.child1.h" /* Message symbol definitions - */
                    /* constructed from message */
                    /* catalog during build process */
#include "ffdcexpl.child1.c" /* Message set definitions and */
                    /* generated routine to obtain */
                    /* default messages - construc- */
                    /* ted by the build process */

#define RESOURCE_NAME      "ffdcexpl"
#define LPP_NAME           "PSSP"
#define VERSION_ID         "1.1"
#define DEVICE_NAME        "/dev/ffdcexpl"

#define DETAIL_DATA_STR(loc, val, maxlen) \
    strlen(val) < (maxlen) ? memcpy((loc), (val), (maxlen)) : \
    memcpy((loc), (val), strlen(val))

/* CATALOG_NAME is defined by the file "ffdcexpl.child1.c", which is */
/* generated by the build process from the contents of the message */
/* catalog source file. */

/*****
* -- MAINLINE CODE -- MAINLINE CODE -- MAINLINE CODE -- MAINLINE CODE --
*****/

main(int argc, char **argv)
{
    int      rc;      /* Function call return codes */
    int      i;      /* Loop counter, error flag */
    int      write_errout; /* Flag - if set to non-zero, */
                    /* write information on failure */
                    /* to stdout */
    char      inv_option; /* Invalid option specified on */
                    /* command line */
    char      <name; /* Input parameter to "child1" */
    char      detail_buf[128]; /* Details on detected failures */
    extern void fc_init_rc_check();

```

```

extern void    record_to_error_stack();

/*
 * Establishing an FFDC Environment at this point, to capture any
 * failure information detected by this program and any child or
 * descendants of this process. Because "child1" expects to be called
 * as a child process in a parent/child relationship, "child1" will
 * not CREATE an FFDC Environment. Instead, it will INHERIT an FFDC
 * Environment if one was previously established by one of its
 * ancestors. If none of its ancestors established an FFDC Environment,
 * then the INHERIT attempt will fail, and no FFDC Environment will be
 * established - failures will have to be noted in some other fashion,
 * such as writing failure messages to the terminal.
 *
 * ----
 * NOTE
 * ----
 * The reader of this example might question why the FFDC Environment
 * was not established after the testing the input parameters. This is
 * a valid question. The code was designed this way because "child1"
 * expects to be a child process whenever it is started. By estab-
 * lishing the FFDC Environment before testing the input parameters,
 * "child1" can now record information to the FFDC Error Stack if it
 * is invoked incorrectly. A problem investigator can then determine
 * that "child1" failed because its parent started it incorrectly. If
 * the FFDC Environment had been established after the input parameters
 * were verified, "child1" would not be able to record a failure report
 * to the FFDC Error Stack if it was invoked incorrectly, and problem
 * investigators would not know from the FFDC Error Stack contents why
 * "child1" failed.
 */
rc = fc_init(FC_STACK_INHERIT);
if (FC_INHERIT_SUCCESS != rc) {
    fc_init_rc_check(rc);
}
/*
 * Now that we've gone through the effort of trying to establish the
 * FFDC Environment, test the usage of this program. There are two
 * failures that can occur in this case:
 * - Incorrect number of parameters      Msg Set "child1", Msg EMSG101
 * - Unknown option specified            Msg Set "child1", Msg EMSG102
 * The error messages will be written to the FFDC Error Stack if the
 * FFDC Environment was successfully established above. The messages
 * written to the FFDC Error Stack are contained in the mythical
 * message catalog file "ffdexpl.cat".
 */
if (3 != argc) {
    /*
     * Record information on incorrect usage to the FFDC
     * Error Stack. In this section, we'll record the
     * input parameters we received to help later problem
     * determination afford figure out the incorrect
     * usage.
     */
    memset((void *) detail_buf, 0, (sizeof(char) * 128));
    sprintf(detail_buf, "Arg Count = %d\n", argc);
    for (i = 1 ; i < argc ; i++) {
        rc = strlen(detail_buf) + strlen(argv[i]) + 1;
        if (128 <= rc) {
            break;
        }
        strcat(detail_buf, argv[i]);
        strcat(detail_buf, "\n");
    }
    record_to_error_stack(__FILE__, VERSION_ID, __LINE__,
        CATALOG_NAME, MSGSET, EMSG101,
        "child1 not invoked correctly", NULL,

```

```

        detail_buf);
    return(1);
}
/*
 * Obtain command line options
 */
name = NULL;
inv_option = (char) NULL;
i = 0;
while ((rc = getopt(argc, argv, "n:")) != EOF) {
    switch (rc) {
        case 'n': name = optarg;
                break;
        default: inv_option = (char) rc;
                break;
    }
    if ((char) NULL != inv_option) {
        break;
    }
}
/*
 * If an invalid option was specified, leave a record of this failure
 * before terminating.
 */
if ((char) NULL != inv_option) {
    /*
     * Record information on incorrect usage to the FFDC
     * Error Stack. In this section, we'll record the
     * invalid option we received to help later problem
     * determination afforst figure out the incorrect
     * usage.
     */
    memset((void *) detail_buf, 0, (sizeof(char) * 128));
    sprintf(detail_buf, "Invalid option = -%c\n", inv_option);
    record_to_error_stack(__FILE__, VERSION_ID, __LINE__,
        CATALOG_NAME, MSGSET, EMSG102,
        "Invalid Option specified", NULL,
        detail_buf);

    return(1);
}
/*
 * The program has verified that it was invoked correctly. Now the
 * program will attempt to activate a resource that it controls. The
 * program expects the resource to be accessable when it is executed.
 * If the resource is not accessable to this command, it is a critical
 * failure in this command, one that must be brought to the system
 * administrator's attention immediately. The "resource" used in this
 * example is a file system, which will be "activated" by mounting it.
 */
rc = activate_resource(name);
if (0 != rc) {
    return(2);
}
/*
 * Program has completed.
 */
return(0);
}

/*****
 * Function Name: activate_resource
 * Description:   Activates the resource controlled by this progam - a
 *               local file system (activated by "mounting" it).
 * Usage:        int = activate_resource(char *name)
 * Parameters:   name "Stub" directory used in the mount request
 * Return Codes: 0   Resource successfully activated
 *               1   Resource could not be activated - fid will
 *****/

```

```

*          contain an FFDC Failure Identifier for the
*          report it made
* Notes:    The example admits that use of mount() is discouraged,
*          and vmount() is preferred. mount() is used by this
*          example to make the example more brief, and not to
*          confuse the reader on the nuances of vmount().
*****/

#define  DETAIL_BUF_FORMAT \
"mount() error: %d\n\
Stub Dir: %s\n\
Device Used: %s\n\
r/w mode attempted"

int
activate_resource(char *name)
{
    int rc;          /* Function call return code */
    int local_errno; /* Local copy of errno */
    int linepos;    /* Location where mount() was */
                    /* attempted */
    size_t detail_len; /* Size of AIX Error Log detail */
                    /* data information */
    char device_name[16]; /* Device to be mounted */
    char detail_buf[100]; /* Detail data written to the */
                    /* AIX Error Log - must follow */
                    /* the format defined for detail*/
                    /* data fields #4 onward for */
                    /* FFDC_EXPL_TPL_ER */
    void *p;        /* Used to construct detail */
                    /* data buffer contents */
    fc_eid_t fid;   /* FFDC Failure Identifier for */
                    /* AIX Error Log entry */

    memset((void *) fid, 0, sizeof(fc_eid_t));
    memset((void *) device_name, 0, (sizeof(char) * 16));
    strcpy(device_name, DEVICE_NAME);
    linepos = __LINE__;
    rc = mount(device_name, name, 0);
    local_errno = errno;
    if (0 != rc) {
        /*
         * Check for failures that can be ignored, such as the file
         * system already being mounted.
         */
        if (EBUSY == local_errno) {
            return(0);
        }
        /*
         * Check for error codes that indicate that the device is not
         * known to the operating system. This is a critical failure
         * that requires the system administrator's attention to
         * resolve. If this failure is detected, log it to the AIX
         * Error Log.
         */
        if (ENOTBLK == local_errno) {
            /*
             * ----
             * NOTE
             * ----
             * Notice that fc_test_stack is not used before
             * recording to the AIX Error Log. This is because
             * fc_test_stack only determines if an FFDC Environment
             * using an FFDC Error Stack exists, An FFDC Error
             * Stack is not needed to record to the AIX Error Log.
             *
             * Also notice that the FFDC log event type is selected

```

```

        * to match the Err_Type in the AIX Error Log template.
        */
memset((void *) detail_buf, 0, (sizeof(char) * 100));
p = (void *) detail_buf;
DETAIL_DATA_STR(p, (void *) device_name, 16);
p += sizeof(char) * 16;
detail_len = 16;
DETAIL_DATA_STR(p, (void *) name, 14);
p += sizeof(char) * 14;
detail_len += 14;
memcpy(p, (void *) &local_errno, sizeof(int));
detail_len += sizeof(int);
fc_log_error(&fid, NULL, RESOURCE_NAME, LPP_NAME,
            __FILE__, VERSION_ID, linepos,
            (void *) NULL, FFDC_ERROR,
            ERRID_FFDC_EXPL_TPL_ER, detail_buf,
            detail_len, NULL,
            "File System Device Mount Failed");
fc_display_fid(fid);
return(1);
}
/*
 * Other failures are failures caused by incorrect usage, or by
 * the user having insufficient privileges. These failures do
 * not require the system administrator's immediate attention,
 * but they are needed to determine why "child1" and its parent
 * process failed. For this reason, the failure data is
 * recorded to the FFDC Error Stack instead of the AIX Error Log
 */
memset((void *) detail_buf, 0, (sizeof(char) * 100));
snprintf(detail_buf, 100, DETAIL_BUF_FORMAT, local_errno, name,
        device_name);
record_to_error_stack(__FILE__, VERSION_ID, linepos,
                    CATALOG_NAME, MSGSET, EMSG103,
                    "child1_Error: mount of file system failed",
                    NULL, detail_buf);

return(1);
}
/*
 * File system successfully mounted.
 */
else {
    return(0);
}
/* NOTREACHED */
}

```

The child2.sh Script

```

#!/bin/ksh
# IBM_PROLOG_BEGIN_TAG
# This is an automatically generated prolog.
#
#
# Licensed Materials - Property of IBM
#
# Restricted Materials of IBM
#
# (C) COPYRIGHT International Business Machines Corp. 1999
# All Rights Reserved
#
# US Government Users Restricted Rights - Use, duplication or
# disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
#
# IBM_PROLOG_END_TAG

```

```

#@(#)11 1.1 src/rsct/ffdc/policycode/child2.sh, ffdc, rsct_rmoh 4/19/99 16:37:33"
#-----
# Module Name:      child2
#
# Purpose:   Used to demonstrate the use of First Failure Data Capture
#            (FFDC) command line interfaces from scripts. The key concepts
#            demonstrated by this script are:
#            - Inheriting an FFDC Environment.
#            - Recording information to the FFDC Error Stack when
#              a failure is detected.
#            - Differentiating between critical failures that need
#              to be recorded and returned to the client, and non-
#              critical failures that are only recorded.
#            - Returning information to the script's client to
#              indicate that a failure occurred, and where the
#              failure information is recorded.
#
# Design:     "child2" is a process that can be called in several situations.
#            This script expects to be invoked directly from the command
#            line in some occasions, but also expects to be called by the
#            "parent" script in other occasions. "child2" does not create
#            and subprocesses of its own. "child2" performs a very specific
#            function on behalf of its user - it is not a convoluted process.
#
#            Since "child2" does not create any subprocesses to accomplish its
#            task, "child2" would have no reason to make use of an FFDC Error
#            Stack when used from the command line - there is only one level
#            of failure to investigate. However, "child2" would want to make
#            use of an FFDC Error Stack if it was called by "parent", since
#            any failures it encounters would most likely cause "parent" to
#            fail, and "parent"'s failure should be able to be connected to
#            the failure in "child2". For this reason, "child2" attempts to
#            INHERIT an FFDC Environment, not CREATE one, that makes use of
#            an FFDC Error Stack. This permits "child2" to make use of an
#            FFDC Error Stack if it was called by "parent" (which will create
#            an FFDC Environment using an FFDC Error Stack). When "child2"
#            is invoked from the command line, it won't make use of an FFDC
#            Error Stack, since it does not need one in that case and the
#            command line environment most likely has not established an
#            FFDC Environment for itself.
#
#            PLEASE NOTE THAT "child2" INHERITS THE FFDC ENVIRONMENT FOR A
#            DIFFERENT REASON THAN "child1". In "child1"'s case, the
#            environment was inherited because it knew it would only be
#            invoked by "parent", so to use the environment, "child1" only
#            needed to inherit it. In "child2"'s case, inheritance is used
#            so that FFDC Error Stacks will not be created when the script
#            is executed from the command line.
#
# Usage: child2 [-n <name>]
#
# Exit Status:  0  "child2" completed successfully
#              1  "child2" not invoked correctly
#              2  "child2" failed - an FFDC Failure Identifier is
#                displayed to inform the client where the failure
#                information resides, if the FFDC Environment could be
#                established
#
# Notes: For the sake of brevity in this example, "child2" does not
#        conform to some specific RS/6000 SP lab software policies,
#        such as the NLS policy for output messages. It is believed
#        that clarity and brevity of the example is preferred over
#        forcing the code example to conform to every lab policy.
#-----
#-----
# Function:  fcinit_rc_check

```

```

# Description:    Check exit status of the fcinit command, display any
#               appropriate messages, and take appropriate action. Terminate
#               the script in cases where a coding error is evidenced.
# Parameters: $1  Exit status from fcinit
#-----
function fcinit_rc_check
{
    case $1 in
        42) print "$0 Warning: FFDC Error Stack use has been"
            print "\t disabled by the system administrator."
            ;;
        19) print "$0 Warning: The FFDC Error Stack directory"
            print "\t does not exist - contact the system"
            print "\t administrator and report the problem."
            ;;
        13) print "$0 Warning: Cannot reserve an FFDC Error"
            print "\t Stack for use by this process, because"
            print "\t the file that would be used to store"
            print "\t the stack already exists."
            ;;
        17) print "$0 Warning: Could not establish an FFDC"
            print "\t Environment because the process environment"
            print "\t appears to be corrupted."
            ;;
        18) print "$0 Warning: Could not establish an FFDC"
            print "\t Environment because memory could not be"
            print "\t allocated. Check the system for processes"
            print "\t that are hoarding memory."
            ;;
        16) print "$0 Warning: Could not establish an FFDC"
            print "\t Environment because fcinit could not modify"
            print "\t the process's environment."
            ;;
        99) print "$0 Warning: Unexpected failure in the fcinit"
            print "\t command. Contact the system administrator"
            print "\t to have problem determination performed."
            ;;
        *) print "$0 Warning: fcinit failed with exit status of"
            print "\t $1. The command has been used incorrectly."
            print "\t Check the fcinit man page for an explanation"
            print "\t of this failure and modify this script to"
            print "\t use fcinit correctly."
            exit 2
            ;;
    esac
}

#-----
# Function:    fcpushstk_rc_check
# Description: Check exit status of the fcpushstk command, display any
#               appropriate messages
# Parameters: $1  Exit status from fcpushstk
# Return Code:  0  "Acceptable" failure - the return code was a warning
#               that incomplete information was provided to fcpushstk
#               1  Failure - could not record to the FFDC Error Stack
#-----
function fcpushstk_rc_check
{
    integer FRC
    case $1 in
        #
        # Failures that did not prevent recording to the FFDC Error
        # Stack are tested in this section.
        #
        28) print "$0 Warning: Resource name is missing. Fix the"
            print "\t script to pass the resource name with -r."
            FRC=0
    esac
}

```

```

;;
30) print "$0 Warning: Default description message is"
    print "\t missing. Fix the script to pass the default"
    print "\t message to the fcpushstk command."
    FRC=0
;;
29) print "$0 Warning: File name or location information is"
    print "\t missing. Fix the script to pass the location"
    print "\t information to the fcpushstk command."
    FRC=0
;;
35) print "$0 Warning: Failure detail information was not"
    print "\t provided, or was NULL."
    FRC=0
;;
37) print "$0 Warning: fcpushstk could not create an FFDC"
    print "\t Failure Identifier for the failure report."
    FRC=0
;;
#
# Failures that prevented a recording to the FFDC Error Stack
# are tested in this section.
#
17) print "$0 Warning: Cannot record information to the"
    print "\t FFDC Error Stack because the process's"
    print "\t environment appears to be corrupted."
    FRC=1
;;
19) print "$0 Warning: Cannot record information to the"
    print "\t FFDC Error Stack because the FFDC Error"
    print "\t Stack directory does not exist. Contact"
    print "\t the system administrator and report the"
    print "\t problem."
    FRC=1
;;
20) print "$0 Warning: Cannot record information to the"
    print "\t FFDC Error Stack because access to the FFDC"
    print "\t Error Stack directory is denied to this"
    print "\t script. Contact the system administrator"
    print "\t and report the problem."
    FRC=1
;;
24) print "$0 Warning: Cannot record information to the"
    print "\t FFDC Error Stack because the FFDC Error Stack"
    print "\t contents appear to be corrupted. This may"
    print "\t happen if another process accidentally writes"
    print "\t data to the file storing the FFDC Error"
    print "\t Stack."
    FRC=1
;;
25) print "$0 Warning: Cannot record information to the"
    print "\t FFDC Error Stack because a file has been"
    print "\t created using the directory name. Contact"
    print "\t the system administrator and report the"
    print "\t problem."
    FRC=1
;;
40) print "$0 Warning: Cannot record information to the"
    print "\t FFDC Error Stack because there is not enough"
    print "\t space in the FFDC Error Stack directory's"
    print "\t file system. Contact the system"
    print "\t administrator and report the problem."
    FRC=1
;;
41) print "$0 Warning: Cannot record information to the"
    print "\t FFDC Error Stack because the FFDC Error Stack"
    print "\t contents appear to be corrupted. This may"

```

```

        print "\t happen if another process accidentally writes"
        print "\t data to the file storing the FFDC Error"
        print "\t Stack."
        FRC=1
        ;;
99) print "$0 Warning: Unexpected failure in the fcpushstk"
    print "\t command. Contact the system administrator"
    print "\t to have problem determination performed."
    FRC=1
    ;;
*) print "$0 Warning: fcpushstk failed with exit status of"
   print "\t $1. The command has been used incorrectly."
   print "\t Check the fcpushstk man page for an"
   print "\t explanation of this failure and modify this"
   print "\t script to use fcinit correctly."
   return 1
   ;;
    esac
    return $FRC
}

#-----
# Function: do_child2_stuff
# Description: A stub routine, which would be replaced by the code that
# actually does what "child2" is supposed to do. It's provided
# here simply to complete the example, and does nothing
# incredibly special
# Parameters: $1 File that is to be created by this routine
# Return Code: None
#-----
function do_child2_stuff
{
    /usr/bin/date > $1
}

#-----
# -- MAINLINE CODE -- MAINLINE CODE -- MAINLINE CODE -- MAINLINE CODE --
#-----
typeset PARAMS
typeset OPTION
typeset NAME
typeset FILE
typeset INFO
typeset RESULTS
typeset FID
typeset FID_MSG
typeset MSG_NUM
integer RC
integer NRC
integer FID_FAILED
integer I
integer NUM_PARAMS
integer STATUS
#
# Save input parameters to special variables. This is done because the script
# will later invoke fcinit.sh in the current shell, which may overwrite the
# input parameter variables.
#
PARAMS=$*
NUM_PARAMS=$#
OPTION=$1
NAME=$2
#
# Time to establish the FFDC Environment to capture failures detected by this
# script and any descendent process it creates. Because "child2" expects to be
# used from the command line and as a child process of "parent", "child2" will
# INHERIT an FFDC Environment if it already exists, instead of CREATING an FFDC

```

```

# Environment that makes use of an FFDC Error Stack. This will prevent "child2"
# from creating an FFDC Error Stack when it is used directly from the command
# line.
#
# Note that the fcinit.sh command is "sourced" from this script, meaning that it
# is executed within the script's own environment.
#
# ----
# NOTE
# ----
# "child2" calls fcinit prior to testing its input. This is done for the same
# reason that "parent" made the same check. See the comments in "parent"'s
# source code for the explanation.
#
. fcinit.sh -si
RC=$?
if ((RC != 1))
then
    fcinit_rc_check $RC
fi
#
# Verify that the command was invoked correctly. There are two failures that
# can occur in this case:
# - Incorrect number of parameters    Msg Set #3, Message #1
# - Unknown option specified          Msg Set #3, Message #2
# The error messages will be written to the FFDC Error Stack if the FFDC
# Environment was successfully established above. The messages written to
# the FFDC Error Stack are contained in the message catalog file
# "ffdcexpl.cat".
#
if ((NUM_PARAMS == 1 || NUM_PARAMS > 2))
then
    #
    # This section tests to make sure that the FFDC Environment was
    # successfully established before trying to record to the FFDC Error
    # Stack.
    #
    fcteststk > /dev/null
    if (($? == 0))
    then
        FID_FAILED=0
        I=0
        while ((I < 4))
        do
            FID_MSG=$(fcpushstk -c ffdcxpl.cat -m3 -n1 -l PSSP \
                -p$LINENO -r ffdcxpl -s $0 -v "1.1" \
                -d "Command line parameters: \"$PARAMS\"" \
                "Script not invoked correctly")
            NRC=$?
            if ((NRC != 0))
            then
                if ((NRC == 22))
                then
                    #
                    # This can happen if multiple processes
                    # are trying to write into the same
                    # FFDC Error Stack. If this happens
                    # four times, consider it unrecoverable.
                    # Four was just a number picked out of
                    # the air for this example.
                    #
                    ((I = I + 1))
                else
                    fcpushstk_rc_check $NRC
                    FID_FAILED=$?
                    I=4
                fi
            fi
        fi
    fi

```

```

        else
            I=4
        fi
    done
else
    FID_FAILED=1
fi
#
# If the FFDC Environment was not established, or a recording could
# not be made to the FFDC Error Stack, display the error information.
# We do this so that some record is made of the failure, even if it
# is to the terminal.
#
if ((FID_FAILED == 1))
then
    print "$0 Error: Script not invoked correctly."
    print "\t Usage:"
    print "\t\t child2 [-n <name>]"
else
    print $FID_MSG | read MSGNUM FID
    fcdispfid $FID
fi
exit 1
fi
if ((NUM_PARAMS == 2))
then
    if [[ "$OPTION" != "-n" ]]
    then
        fcteststk > /dev/null
        if (($? == 0))
        then
            FID_FAILED=0
            I=0
            while ((I < 4))
            do
                FID_MSG=$(fcpushstk -c ffdcxpl.cat -m3 -n2 \
                    -l PSSP -p$LINENO -r ffdcxpl -s $0 \
                    -v "1.1" \
                    -d "Invalid option: \"$OPTION\"" \
                    "Invalid option specified")
                NRC=$?
                if ((NRC != 0))
                then
                    if ((NRC == 22))
                    then
                        ((I = I + 1))
                    else
                        fcpushstk_rc_check $NRC
                        FID_FAILED=$?
                        I=4
                    fi
                else
                    I=4
                fi
            done
        else
            FID_FAILED=1
        fi
        if ((FID_FAILED == 1))
        then
            print "$0 Error: Invalid option specified: $OPTION"
            print "\t Usage:"
            print "\t\t child2 [-n <name>]"
        else
            print $FID_MSG | read MSGNUM FID
            fcdispfid $FID
        fi
    fi

```

```

        exit 1
    fi
else
    NAME=/tmp
fi
FILE=$NAME/child2.tokgen
#
# "child2" invoked correctly, and a file name has been constructed from the
# optional user input. "child2" expects that this file does not exist at the
# time it starts, since this would indicate that another instance of "child2"
# is running (or terminated in an error state where it could not clean up the
# file). If the file exists, this may pose a problem, but maybe not a critical
# one, so "child2" will continue if it file is present. In this case, "child2"
# would want to leave a record of the incident to assist later problem
# investigation (if this incident does indeed lead to a failure), but the
# incident is not enough to cause complete failure in "child2". "child2" will
# note that the file exists to the FFDC Error Stack (if it is use), and simply
# overwrite the contents of the file.
#
if [[ -f $NAME ]]
then
    fcteststk > /dev/null
    if (($? == 0))
    then
        FID_FAILED=0
        I=0
        while ((I < 4))
        do
            FID_MSG=$(fcpushstk -c ffdcexpl.cat -m3 -n3 \
                -l PSSP -p$LINENO -r ffdcexpl -s $0 \
                -v "1.1" -d "$NAME" \
                "Warning: $NAME file exists - overwriting")
            NRC=$?
            if ((NRC != 0))
            then
                if ((NRC == 22))
                then
                    ((I = I + 1))
                else
                    fcpushstk_rc_check $NRC
                    FID_FAILED=$?
                    I=4
                fi
            else
                I=4
            fi
        done
    fi
    #
    # Notice that the script continues, and does not "return" the FFDC
    # Failure Identifier of the record made to the caller. This is because
    # "child2" is deciding not to treat this as a critical failure and
    # continuing. Since "child2" does not consider this a critical failure,
    # it also chooses not to return error information to its client. Had
    # this been a critical failure, or a failure important enough to bring
    # to the client's attention, "child2" could be using "fcdispfid" to
    # send this information back to the client.
    #
    rm -f $FILE
fi
#
# Perform this script's function and verify the results. If the results do not
# appear valid, "child2" will consider this to be a critical failure and
# terminate with an error notification to its caller. The result test is
# rather simplistic in this example: were any results generated?
#
do_child2_stuff $FILE

```

```

if [[ ! -s $FILE ]]
then
    STATUS=2
    fcteststk > /dev/null
    if (($? == 0))
    then
        FID_FAILED=0
        I=0
        while ((I < 4))
        do
            FID_MSG=$(fcpushstk -c ffdcxpl.cat -m3 -n4 -l PSSP \
                -p$LINENO -r ffdcxpl -s $0 -v "1.1" \
                -d "date command failed - null output" \
                "Failure in do_child2_stuff - exiting")

            NRC=$?
            if ((NRC != 0))
            then
                if ((NRC == 22))
                then
                    ((I = I + 1))
                else
                    fcpushstk_rc_check $NRC
                    FID_FAILED=$?
                    I=4
                fi
            else
                I=4
            fi
        done
    else
        FID_FAILED=1
    fi
    if ((FID_FAILED == 1))
    then
        print "$0 Error: Failure in procedure do_child2_stuff - exiting"
    else
        print $FID_MSG | read MSGNUM FID
        fcdispfid $FID
    fi
else
    print "I Came, I Saw, I Conquered"
    STATUS=0
fi
#
# Clean up and return the appropriate completion status to the client. Note that
# earlier code would have displayed an FFDC Failure Identifier if a critical
# failure was encountered, so this section of code doesn't handle it.
#
exit $STATUS

```

The grandp.c Program

```

/* IBM_PROLOG_BEGIN_TAG                               */
/* This is an automatically generated prolog.         */
/*                                                    */
/*                                                    */
/* Licensed Materials - Property of IBM                */
/*                                                    */
/* Restricted Materials of IBM                          */
/*                                                    */
/* (C) COPYRIGHT International Business Machines Corp. 1999 */
/* All Rights Reserved                                 */
/*                                                    */
/* US Government Users Restricted Rights - Use, duplication or
/* disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
/*

```

```

/* IBM_PROLOG_END_TAG                                     */
/*
 * Module Name:    child1
 *
 * Purpose:    Used to demonstrate the use of First Failure Data Capture
 *             (FFDC) application programming interface from a C language
 *             program. The key concepts demonstrated by this program are:
 *             - The establishment of an FFDC Environment that makes
 *               use of an FFDC Error Stack, and how this affects
 *               previously existing applications that also made use
 *               of an FFDC Error Stack environment.
 *
 * Compilation:    This module must be linked to the following RCST libraries:
 *             libct_ffdc.a    For FFDC function
 *             libct_cu.a     For Cluster Utility Error Handling
 *             This module must also be linked with the "utils.o" object module
 *
 * Design:    "grandp" is written as a new application that invokes the
 *            previously existing application "parent" to accomplish its task.
 *            "grandp" expects to be invoked as the top-most process in the
 *            parent/child/descendent process relationship. This will force
 *            the "parent" process, formerly accustomed to being the top-most
 *            process, into a subordinate process role. Because the outcome
 *            of "grandp" will depend upon the success or failure of the
 *            subprocesses it creates, "grandp" will CREATE an FFDC Environ-
 *            ment that makes use of an FFDC Error Stack. The FFDC Error
 *            Stack will permit "grandp" to capture the failure information
 *            generated by itself and any process it creates that also chooses
 *            to use the FFDC Error Stack, and permit investigators into
 *            "grandp" failures to associate any failures in the subprocesses
 *            to "grandp"'s outcome.
 *
 *            The purpose of writing "grandp" is to demonstrate how this
 *            application impacts "parent". "parent" had been written to
 *            CREATE and FFDC Environment using an FFDC Error Stack, not to
 *            INHERIT it. How will this application affect "parent"'s use
 *            of the FFDC Error Stack? When "parent" attempts to create the
 *            environment, the FFDC utilities will detect that its parent
 *            process "grandp" has already created an environment making use
 *            of an FFDC Error Stack. FFDC will conclude that "grandp" wants
 *            to capture all failure information for any process descendent
 *            from it in "grandp"'s FFDC Error Stack, and the utilities will
 *            convert "parent"'s request to CREATE an FFDC Environment into a
 *            request to INHERIT the existing FFDC Environment. If the FFDC
 *            utilities did not choose to do this, "parent" would wind up
 *            creating its own FFDC Error Stack, and "grandp" would have no
 *            way of capturing any failure information from "parent" or any
 *            of "parent"'s child processes. This would defeat the purpose of
 *            the FFDC Error Stack.
 *
 *            Now that "grandp" has created an FFDC Error Stack environment,
 *            "parent" will instead INHERIT the environment. Note that
 *            "parent" was coded for this eventuality. "parent" will still
 *            operate as normal, since the FFDC Environment will still exist,
 *            and "parent" and any of its children will record failure infor-
 *            mation to "grandp"'s FFDC Error Stack.
 *
 * Usage:    grandp
 *
 *            If any command line options are provided, they are ignored.
 *            This decision is made to simplify the example code.
 *
 * Exit Status:    0    Successful completion
 *                2    "grandp" command failed - an FFDC Failure Identifier
 *                    will be displayed to inform the client of the failure,
 *                    if the FFDC Environment was successfully established.
 *

```

```

* Notes:      For the sake of brevity in this example, "grandp" does not
*             conform to some specific RS/6000 SP lab software policies,
*             such as the NLS policy for output messages. It is believed
*             that clarity and brevity of the example is preferred over
*             forcing the code example to conform to every lab policy.
*/

#include <stdio.h>      /* Basic I/O capability      */
#include <errno.h>     /* Error code definitions */
#include <string.h>    /* Character string manipulation*/
#include <memory.h>   /* Memory manipulation    */
#include <stdlib.h>   /* system() interface    */
#include <sys/wait.h> /* Exit status macros    */
#include <rsct/ct_ffdc.h> /* FFDC definitions    */
#include <rsct/ct_cu.h> /* Cluster error handling */
#include "ffdcexpl.grandp.h" /* Message symbol definitions - */
                          /* generated from the message */
                          /* catalog by the build process */
#include "ffdcexpl.grandp.c" /* Message set definition and */
                          /* routine to obtain default */
                          /* messages - generated from the*/
                          /* message catalog by the build */
                          /* process */

#define INPUT_FILE      "/tmp/ffdcexpl.dat"
#define VERSION_ID     "1.1"

#define GRANDP_DETAIL_FMT0 \
"Subroutine: %s\n\
Null character pointer provided as input parameter"

#define GRANDP_DETAIL_FMT1 \
"Subroutine: %s\n\
fopen Mode: R/O\n\
Input File: %s\n\
Errno Value: %d\n"

/*****
* -- MAINLINE CODE -- MAINLINE CODE -- MAINLINE CODE -- MAINLINE CODE --
*****/

main(int argc, char **argv)
{
    int    rc;      /* Function call return code */
    int    i;      /* Loop counter */
    int    linepos; /* Approximation of failing */
                /* line of code position */
    int    estatus; /* Exit status from "parent" */
    char   name[80]; /* Parameter to "parent" */
    char   tfile[L_tmpnam]; /* For stderr from "parent" */
    char   cmd_buf[80]; /* Used in system() call */
    char   detail_buf[100]; /* Details on failure */
    extern void fc_init_rc_check();
    extern void record_to_error_stack();
    extern void interpret_parent_failure();

    /*
     * Establishing an FFDC Environment at this point, to capture any
     * failure information detected by this program and any child or
     * descendants of this process. Because "grandp" expects to be the
     * top-most process in the parent/child process relationship, it will
     * CREATE the environment. Any descendants of this process that
     * attempt to CREATE an FFDC Environment using an FFDC Error Stack
     * will discover that such an environment was already established by
     * "grandp", and the FFDC utilities will instead INHERIT the existing
     * environment.
     */
}

```

```

* ----
* NOTE
* ----
* Notice that "grandp" requests only to create an FFDC Error Stack,
* not also to make use of the AIX Error Log. This will NOT prevent
* any of "grandp"'s descendants from making use of the AIX Error
* Log, since use of the AIX Error Log does not depend on the FFDC Error
* Stack environment.
*
* Also note that "grandp" accounts for the fact that it may also wind
* up inheriting the FFDC Environment instead of creating it.
*/
rc = fc_init(FC_STACK_CREAT);
if (FC_SUCCESS != rc && FC_INHERIT_SUCCESS != rc) {
    fc_init_rc_check(rc);
}
/*
* Do processing specific to "grandp".
*/
memset((void *) name, 0, (sizeof(char) * 80));
for (i = 0 ; i < 4 ; i++) {
    rc = do_grandp_stuff(name);
    switch (rc) {
        case 0: break;
        case 1: return(2);
        case 2: continue;
        case 3: return(2);
    }
}
if (2 == rc) {
    /*
    * Unable to open the input file because the oprn attempt
    * keeps getting interrupted. After four failed attempts,
    * "grandp" considers this a critical failure. Information
    * for the failure condition is recorded to the FFDC Error
    * Stack.
    */
    memset((void *) detail_buf, 0, (sizeof(char) * 100));
    snprintf(detail_buf, 100, GRANDP_DETAIL_FMT1, "do_grandp_stuff",
             INPUT_FILE, EINTR);
    record_to_error_stack(__FILE__, VERSION_ID, __LINE__,
                        CATALOG_NAME, MSGSET, EMSG307,
                        getmsg_ffdcexpl_grandp(EMSG307), NULL,
                        detail_buf);

    return(2);
}
/*
* Invoke process "parent", and check the results. If "parent" fails,
* then this program also considers itself to have failed. In that
* case, "grandp" will record its own failure to the FFDC Error Stack,
* citing the failure in "parent". Notice that "parent"'s standard
* error is redirected to a file so that "grandp" can obtain an FFDC
* Failure Identifier from it if it needs to.
*/
memset((void *) tfile, 0, (sizeof(char) * L_tmpnam));
tmpnam(tfile);
memset((void *) cmd_buf, 0, (sizeof(char) * 80));
sprintf(cmd_buf, 80, "./parent -n %s 2> %s", name, tfile);
linepos = __LINE__;
rc = system(cmd_buf);
estatus = WEXITSTATUS(rc);
if (0 != estatus) {
    interpret_parent_failure(estatus, linepos, tfile);
    unlink(tfile);
    return(2);
}
/*

```

```

    * Execution of "grandP" was successful - exit gracefully.
    */
    unlink(tfile);
    return(0);
}

/*****
* Function Name: do_grandp_stuff
* Description:   Obtains the name to use as an argument to the "parent"
*               script. If this routine fails, it signifies a critical
*               failure in the program.
* Usage:        int = do_grandp_stuff(char *name)
* Parameters:   name Where to store input retrieved by this routine -
*               must have been previously allocated by caller.
* Return Codes: 0 Successful completion
*               1 Failure detected - information is reported to
*               the FFDC Error Stack file if an FFDC Environment
*               was successfully established.
*               2 Routine was interrupted by a signal - retry
*               3 Incorrect parameters provided.
* Notes:        The name is obtained from a file that is expected to
*               exist. While this is a critical failure, the designer
*               of "grandp" does not consider this a failure critical
*               enough to warrant the system administrator's attention
*               (the program was written as a convenience to a specific
*               user, not for all users or the system administrator).
*               If the failure were critical enough to require the
*               system administrator's attention (if the tool were meant
*               for any user at large, and the failure prohibited any
*               user from making use of the program), then the failure
*               should be recorded to the AIX Error Log.
* Additional Notes: getmsg_ffdcexpl_grandp() is defined in the file
*                  "ffdcexpl.grandp.c", which is generated during the build
*                  process, and is created from the contents of the message
*                  catalog file. CATALOG_NAME is also defined in this
*                  file.
*****/
int
do_grandp_stuff(char *name)
{
    int rc; /* Function call return code */
    int linepos; /* Approximation of the line of */
                /* code that failed */
    int local_errno; /* Copy of "errno" value */
    FILE *infile; /* Where to get "name" from */
    char *p; /* Input string scanning pointer*/
    char inline[80]; /* Input from input file */
    char detail_buf[100]; /* Details on failure */
    extern void record_to_error_stack();

    /*
    * Validate parameters, as best as possible. If the parameters are
    * invalid, record an appropriate indication to the FFDC Error Stack.
    * ----
    * NOTE
    * ----
    * Recording information to the FFDC Error Stack in this case may be
    * considered by some readers to be a bit of "overkill". After all,
    * the return code setting should be enough to inform the caller of
    * the error. Providers of library modules may correctly consider an
    * appropriate error code setting to be sufficient for reporting the
    * failure. However, should this failure occur between internal
    * subroutines of a program or a daemon in product level code, not only
    * should a persistent record of it be made, but more than likely it
    * should be recorded to the AIX Error Log (since such a failure implies
    * a coding error in the product level code itself, and needs to be
    * brought to the attention of Service).
    */

```

```

*/
if (NULL == name) {
    memset((void *) detail_buf, 0, (sizeof(char) * 100));
    snprintf(detail_buf, 100, GRANDP_DETAIL_FMT0, "do_grandp_stuff");
    record_to_error_stack(__FILE__, VERSION_ID, __LINE__,
                        CATALOG_NAME, MSGSET, EMSG305,
                        getmsg_ffdcexpl_grandp(EMSG305), NULL,
                        detail_buf);

    return(3);
}
/*
 * Attempt to open an input file. If the open request fails, determine
 * the reason and record an appropriate failure indication to the FFDC
 * Error Stack.
 */
linepos = __LINE__;
infile = fopen(INPUT_FILE, "r");
local_errno = errno;
if ((FILE *) NULL == infile) {
    memset((void *) detail_buf, 0, (sizeof(char) * 100));
    snprintf(detail_buf, 100, GRANDP_DETAIL_FMT1, "do_grandp_stuff",
            INPUT_FILE, local_errno);
    switch (local_errno) {
        case EINTR: /* Interrupted. Not a failure, */
                    /* so no record will be recorded*/
                    /* to the FFDC Error Stack */
        return(2);
        case ENOENT: /* File does not exist */
        record_to_error_stack(__FILE__, VERSION_ID,
                            linepos, CATALOG_NAME,
                            MSGSET, EMSG301,
                            getmsg_ffdcexpl_grandp(EMSG301),
                            NULL, detail_buf);

        return(1);
        case ENFILE: /* Too many files opened */
        record_to_error_stack(__FILE__, VERSION_ID,
                            linepos, CATALOG_NAME,
                            MSGSET, EMSG302,
                            getmsg_ffdcexpl_grandp(EMSG302),
                            NULL, detail_buf);

        return(1);
        case ENOTDIR: /* Directory containing the file*/
                    /* is not available */
        record_to_error_stack(__FILE__, VERSION_ID,
                            linepos, CATALOG_NAME,
                            MSGSET, EMSG303,
                            getmsg_ffdcexpl_grandp(EMSG303),
                            NULL, detail_buf);

        return(1);
        default: /* Some error that should not */
                /* happen. */
                /* ---- */
                /* NOTE */
                /* ---- */
                /* This indicates a coding bug. */
                /* If this were product level */
                /* code, this failure should be */
                /* recorded to the AIX Error Log*/
        record_to_error_stack(__FILE__, VERSION_ID,
                            linepos, CATALOG_NAME,
                            MSGSET, EMSG304,
                            getmsg_ffdcexpl_grandp(EMSG304),
                            NULL, detail_buf);

        return(1);
    }
}
}

```

```

/*
 * File has been accessed. Attempt to obtain the input from the file.
 * Once again, if the request fails, record an appropriate indication
 * to the FFDC Error Stack.
 */
memset((void *) inline, 0, (sizeof(char) * 80));
linepos = __LINE__;
if (NULL == fgets(inline, 80, infile)) {
    local_errno = errno;
    memset((void *) detail_buf, 0, (sizeof(char) * 100));
    snprintf(detail_buf, 100, GRANDP_DETAIL_FMT1, "do_grandp_stuff",
             INPUT_FILE, local_errno);
    record_to_error_stack(__FILE__, VERSION_ID, linepos,
                        CATALOG_NAME, MSGSET, EMSG306,
                        getmsg_ffdcexpl_grandp(EMSG306),
                        NULL, detail_buf);

    fclose(infile);
    return(1);
}
/*
 * Input retrieved from file - return information to caller
 */
p = strchr(inline, '\n');
(*p) = '\0';
strcpy(name, inline);
return(0);
}

/*****
 * Function Name: interpret_parent_failure
 * Description:   Determines the manner in which the "parent" process
 *               failed, and records an entry to the FFDC Error Stack
 *               to indicate "grandp"'s related failure.
 * Usage:        (void) interpret_parent_failure(int estatus,
 *               int linepos, char *tfile)
 * Parameters:   estatus      Exit status from "parent"
 *               linepos     Location of the failing code call
 *               tfile       The name of a file containing standard
 *               error output from "parent"
 * Return Codes: None
 *****/
void
interpret_parent_failure(int estatus, int linepos, char *tfile)
{
    FILE      *einfo;      /* Access to "parent"'s stderr */
    char      *p;          /* Substring pointer */
    char      inline[80];  /* Input from "einfo" */
    char      detail_buf[100]; /* Details on the failure */
    fc_eid_t  assoc_fid;   /* FFDC Failure Identifier from
                          /* "parent" */

    /*
     * Attempt to open the standard error file to obtain the FFDC Failure
     * Identifier from "parent" (if one was created). If the file cannot
     * be opened, indicate that no details on "grandp"'s failure exist
     * and record a failure notice to the FFDC Error Stack.
     */
    einfo = fopen(tfile, "r");
    if (NULL == einfo) {
        record_to_error_stack(__FILE__, VERSION_ID, linepos,
                            CATALOG_NAME, MSGSET, EMSG308,
                            getmsg_ffdcexpl_grandp(EMSG308),
                            NULL, "No detail information available");

        return;
    }
    /*
     * Attempt to obtain the FFDC Failure Identifier from "parent"'s

```

```

* standard error. Again, if this information cannot be read,
* indicate that no details on "grandp"'s failure exist and record a
* failure notice to the FFDC Error Stack.
*/
memset((void *) inline, 0, (sizeof(char) * 80));
if (NULL == fgets(inline, 80, einfo) {
    fclose(einfo);
    record_to_error_stack(__FILE__, VERSION_ID, linepos,
                        CATALOG_NAME, MSGSET, EMSG308,
                        getmsg_ffdcexpl_grandp(EMSG308),
                        NULL, "No detail information available");

    return;
}
fclose(einfo);
/*
* Since the FFDC Failure Identifier is reported by "parent" using the
* "fcdispfid" command, "grandp" can expect the message to be in the
* form:
* 2615-000<single_blank_space><FFDC Failure Identifier>
* "grandp" needs to confirm that the output is in this format, and
* strip the FFDC Failure Identifier from the message. The stripped
* FFDC Failure Identifier will be used as the "associated" failure in
* grandp"'s failure report to the FFDC Error Stack.
*/
if (0 != strncmp(inline, "2615-000 ", 9)) {
    record_to_error_stack(__FILE__, VERSION_ID, linepos,
                        CATALOG_NAME, MSGSET, EMSG308,
                        getmsg_ffdcexpl_grandp(EMSG308),
                        NULL, "No detail information available");

    return;
}
memset((void *) assoc_fid, 0, sizeof(fc_eid_t));
p = &(inline[9]);
strncpy(assoc_fid, p, FC_EID_LEN);
/*
* Now that "parent"'s FFDC Failure Identifier has been obtained,
* record the associated failure of "grandp" in the FFDC Error Stack.
*/
memset((void *) detail_buf, 0, (sizeof(char) * 100));
sprintf(detail_buf, 100, "Failed process: %s\nExit Status: %d",
        "parent", estatus);
record_to_error_stack(__FILE__, VERSION_ID, linepos, CATALOG_NAME,
                    MSGSET, EMSG308, getmsg_ffdcexpl_grandp(EMSG308),
                    assoc_fid, detail_buf);

return;
}

```

The utils.c Utility Code Module

```

/* IBM_PROLOG_BEGIN_TAG                                     */
/* This is an automatically generated prolog.              */
/*                                                         */
/*                                                         */
/*                                                         */
/* Licensed Materials - Property of IBM                    */
/*                                                         */
/* Restricted Materials of IBM                             */
/*                                                         */
/* (C) COPYRIGHT International Business Machines Corp. 1999 */
/* All Rights Reserved                                     */
/*                                                         */
/* US Government Users Restricted Rights - Use, duplication or */
/* disclosure restricted by GSA ADP Schedule Contract with IBM Corp. */
/*                                                         */
/* IBM_PROLOG_END_TAG                                     */
/*

```

```

* Module Name:    utils.c
*
* Purpose:    Used to demonstrate the use of First Failure Data Capture
*             (FFDC) application programming interface from a C language
*             program. The key concepts demonstrated by this program are:
*             - Recording information to the FFDC Error Stack
*               when a failure is detected.
*             - Checking return codes from the FFDC API utilities.
*             - Returning information to the program's client to
*               indicate that a failure occurred, and where the
*               failure information is recorded.
*
* Compilation:  This module must be linked to the following RCST libraries:
*             libct_ffdc.a  For FFDC function
*             libct_cu.a   For Cluster Utility Error Handling
*             The object produced from this module can be linked to object
*             modules from other C source code files.
*
* Design:      This module contains routines to test return codes from the
*             fc_init and fc_push_stack API routines. It also contains a
*             routine to record information to an FFDC Error Stack via the
*             fc_push_stack API. A separate file has been created for these
*             functions to allow multiple source code files to make use of
*             the same functions. The object module produced from this
*             source code module can be linked to other object code modules.
*
* Notes:      For the sake of brevity in this example, "child1" does not
*             conform to some specific RS/6000 SP lab software policies,
*             such as the NLS policy for output messages. It is believed
*             that clarity and brevity of the example is preferred over
*             forcing the code example to conform to every lab policy.
*/

```

```

#include <stdio.h>      /* Basic I/O capability */
#include <errno.h>     /* Error code definitions */
#include <memory.h>    /* Memory manipulation routines */
#include <rsct/ct_ffdc.h> /* FFDC definitions */
#include <rsct/ct_cu.h> /* Cluster Error Handling */
#include <ffdcexpl.err.S.h> /* AIX Error Log template header*/
/* generated by the build */
/* process */
#include "ffdcexpl.child1.h" /* Message symbol definitions - */
/* constructed from message */
/* catalog during build process */
/* #include "ffdcexpl.child1.c" * Message set definitions and */
/* generated routine to obtain */
/* default messages - construc- */
/* ted by the build process */

```

```

#define RESOURCE_NAME "ffdcexpl"
#define LPP_NAME "PSSP"
#define DEVICE_NAME "/dev/ffdcexpl"

```

```

/*****
* Function Name: fc_init_rc_check
* Description:   Check return code from fc_init, and display any
*               appropriate error messages.
* Usage:        (void) fc_init_rc_check(int code)
* Parameters:   code Return code from the fc_init API call
* Return Codes: None
* Notes:       Note the use of cu_get_error and cu_get_errmsg to
*               retrieve details on the failure experienced by fc_init.
*               This retrieves the failure information from thread
*               specific data. The cu_get_error and cu_get_errmsg calls
*               allocate memory for their callers, so the caller must
*               use cu_rel_error and cu_rel_errmsg after it is finished
*               using the error information.
*/

```

```

*****/
void
fc_init_rc_check(int rc)
{
    char      *errmsg; /* Error message from fc_init */
    cu_error_t  *errblock; /* Error info from fc_init */

    errmsg = (char *) NULL;
    errblock = (cu_error_t *) NULL;
    switch (rc) {
        /*
         * The following cases are conditions beyond the application
         * writer's control...
         */
        case FC_STACK_DISABLED:
        case FC_STACK_DIRECTORY:
        case FC_ENV_CORRUPT:
        case FC_NO_MEMORY:
        case FC_ENV_FAILED:
        case FC_INTERNAL:
            cu_get_error(&errblock);
            cu_get_errmsg(errblock, &errmsg);
            fprintf(stdout, "%s", errmsg);
            cu_rel_errmsg(errmsg);
            cu_rel_error(errblock);
            return;

        /*
         * Anything else is an application programming error and
         * should be corrected.
         */
        default:
            printf("child1 Warning: fc_init failed, return code ");
            printf("%d\n", rc);
            printf("\t This indicates that the function call\n");
            printf("\t was not issued correctly. Check the\n");
            printf("\t following message for details on the\n");
            printf("\t failure, and repair this program to\n");
            printf("\t use fc_init correctly:\n");
            cu_get_error(&errblock);
            cu_get_errmsg(errblock, &errmsg);
            fprintf(stdout, "%s", errmsg);
            cu_rel_errmsg(errmsg);
            cu_rel_error(errblock);
            return;
            return;
    }
    /* NOTREACHED */
}

/*****
 * Function Name: record_to_error_stack
 * Description:   Place a record of a failure into the FFDC Error Stack.
 *               make repeated attempts if fc_push_stack indicates that
 *               the FFDC Error Stack is locked by another process, but
 *               give up after four consecutive lock failures.
 * Usage:        (void) record_to_error_stack(char *file, char *version,
 *               int linepos, char *catalog,
 *               int msg_set, int msg_num,
 *               char *default_msg,
 *               fc_eid_t assoc_failure_id,
 *               char *details)
 * Parameters:   file      Name of file detecting the failure
 *               condition
 *               version   SCCS ID of "file"
 *               linepos   Location that detected the failure, or
 *               a reasonable approximation of it
 *               catalog   Name of the message catalog containing

```

```

*           the message describing the failure
*           condition
*   msg_set      Message set in the message catalog
*               containing a description of the failure
*               condition
*   msg_num      Message within the message set that
*               describes the failure condition
*   default_msg  The message to be used when the message
*               catalog cannot be located or opened
*   assoc_failure_id Optional parameter. If provided, it
*               specifies the FFDC Failure Identifier of
*               a previously recorded failure condition
*               that is related to this failure. If
*               no associated failure ID is to be
*               provided, the caller should supply
*               NULL for this value.
*   details      Detailed information on the failure
*               condition - meant for use by problem
*               investigators only
* Return Codes:  None.
* Notes:        This function is used to demonstrate that failure
*               information can be recorded by a subroutine. If a
*               program chooses to use a subroutine to report a failure
*               instead of reporting the failure in-line, the subroutine
*               must be provided with information that indicates where
*               the failure was detected, and pass this information
*               along to fc_push_stack. If this subroutine uses its
*               own location information instead of the detecting code's
*               location information, all failures will look like they
*               originate from THIS routine, confusing problem
*               investigators.
*****/
void
record_to_error_stack(char *file, char *version, int linepos, char *catalog,
                    int msg_set, int msg_num, char *default_msg,
                    fc_eid_t assoc_failure_id, char *details)
{
    int    rc;        /* Function call return code */
    int    i;        /* Loop counter */
    int    rec_failed; /* Flag - set to non-zero if */
                    /* unable to record to FFDC */
                    /* Error Stack */
    fc_eid_t fid;    /* FFDC Failure Identifier */

    /*
     * Only write to the FFDC Error Stack if an FFDC Environment using an
     * FFDC Error Stack was successfully established. In other cases,
     * display the error message to standard output (so at least some
     * record of the failure might exist).
     */
    rc = fc_test_stack();
    if (FC_SUCCESS != rc) {
        rec_failed = 1;
    }
    else {
        memset((void *) fid, 0, sizeof(fc_eid_t));
        rec_failed = 0;
        for (i = 0 ; i < 4 ; i++) {
            rc = fc_push_stack(&fid, assoc_failure_id,
                              RESOURCE_NAME, LPP_NAME, file,
                              version, linepos, (void *) NULL,
                              details, (char *) NULL,
                              catalog, msg_set, msg_num,
                              default_msg);
            if (FC_STACK_LOCK == rc) {
                continue;
            }
        }
    }
}

```

```

        if (FC_SUCCESS == rc) {
            break;
        }
        else {
            rec_failed = fc_push_stack_rc_check(rc);
            break;
        }
    }
}
/*
 * If an FFDC Error Stack is not in use, or a recording could not be
 * made to the FFDC Error Stack, display the error description to
 * standard output.
 */
if (0 != rec_failed) {
    fprintf(stdout, "%s\n", default_msg);
}
/*
 * If an entry was made to the FFDC Error Stack, "return" the FFDC
 * Failure Identifier to "child1"'s client by displaying the FFDC
 * Failure Identifier to standard error.
 */
else {
    rc = fc_display_fid(fid);
}
return;
}

/*****
 * Function Name: fc_push_stack_rc_check
 * Description:   Check return code from fc_push_stack, and display any
 *               appropriate error messages.
 * Usage:        int = fc_push_stack_rc_check(int code)
 * Parameters:   code Return code from the fc_init API call
 * Return Codes: 0 "Acceptable" return code
 *               1 Caller should consider the attempt to write to
 *               FFDC Error Stack a failure
 *               2 Copy of detail data file failed - do not remove
 *               the original version!
 * Notes:        Note the use of cu_get_error and cu_get_errmsg to
 *               retrieve details on the failure experienced by FFDC.
 *               This retrieves the failure information from thread
 *               specific data. The cu_get_error and cu_get_errmsg calls
 *               allocate memory for their callers, so the caller must
 *               use cu_rel_error and cu_rel_errmsg after it is finished
 *               using the error information.
 *****/
int
fc_push_stack_rc_check(int rc)
{
    char    *errmsg; /* Error message from fc_init */
    cu_error_t  *errblock; /* Error info from fc_init */

    errmsg = (char *) NULL;
    errblock = (cu_error_t *) NULL;
    switch (rc) {
        /*
         * The following are "acceptable" failures... they are really
         * warnings to the application writer that incomplete data
         * was provided to the fc_push_stack routine.
         */
        case FC_RESOURCE_INV:
        case FC_DEFAULTDESC_INV:
        case FC_DESC_INV:
        case FC_DETECTFILE_INV:
        case FC_DETAILDATA_INV:
            cu_get_error(&errblock);

```

```

        cu_get_errmsg(errblock, &errmsg);
        fprintf(stdout, "%s", errmsg);
        cu_rel_errmsg(errmsg);
        cu_rel_error(errblock);
        return(0);
/*
 * In this case, a recording was made, but the detail data
 * provided by the program could not be copied to FFDC's
 * dump directory. The program's user has to be informed
 * not to remove the original copy, since the FFDC Error
 * Stack record will refer to it.
 */
case FC_COPY_FAILED:
    cu_get_error(&errblock);
    cu_get_errmsg(errblock, &errmsg);
    fprintf(stdout, "%s", errmsg);
    cu_rel_errmsg(errmsg);
    cu_rel_error(errblock);
    return(2);
/*
 * In the following case, even though a record was made for
 * the failure to the FFDC Error Stack, fc_push_stack could
 * not create an FFDC Failure Identifier to locate the report
 * with. The program will have nothing to return to its client
 * to use to identify the failure. In this case, we'll treat
 * it as if the record hadn't been made, and another record
 * of the failure must be made (we didn't have to make this
 * decision since the record is in the FFDC Error Stack
 * anyway, but this program chooses to do this so that the
 * user will see some indication of a failure).
 */
case FC_EID_FAILED:
    cu_get_error(&errblock);
    cu_get_errmsg(errblock, &errmsg);
    fprintf(stdout, "%s", errmsg);
    cu_rel_errmsg(errmsg);
    cu_rel_error(errblock);
    return(1);
/*
 * The rest are actual failures that prevented a recording to
 * the FFDC Error Stack.
 */
default:
    cu_get_error(&errblock);
    cu_get_errmsg(errblock, &errmsg);
    fprintf(stdout, "%s", errmsg);
    cu_rel_errmsg(errmsg);
    cu_rel_error(errblock);
    return(1);
}
/* NOTREACHED */
}

```

The ffdcxpl.msg Message Catalog File

```

$ IBM_PROLOG_BEGIN_TAG
$ This is an automatically generated prolog.
$
$
$ Licensed Materials - Property of IBM
$
$ Restricted Materials of IBM
$
$ (C) COPYRIGHT International Business Machines Corp. 1999
$ All Rights Reserved

```

```

$
$ US Government Users Restricted Rights - Use, duplication or
$ disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
$
$ IBM_PROLOG_END_TAG
$ -----
$ Module Name:      ffdcxpl.msg
$
$ Purpose:      Used to demonstrate the use of First Failure Data Capture
$               (FFDC) command line and application programming interfaces.
$               The messages within this message catalog file are used to
$               report failure conditions to the FFDC Error Stack, and to
$               construct AIX Error Logging templates.
$ -----
$ @(#)35  1.1  src/rsct/ffdc/policycode/ffdcexpl.msg, ffdc, rsct_rmo2, rmo2t1fg 4/19/99 16:44:57
$ -----
$ double-quotes (") will be used to delimit messages in the catalog
$ quote "
$ -----
$ Messages used by the "parent" script
$set parent
MSG001 "parent Error: Script not invoked correctly - exiting"
MSG002 "parent Error: Invalid option provided - exiting"
MSG003 "parent Error: Failure in child process, cannot continue - exiting"
$ -----
$ Messages used by the "child1" program
$set child1
MSG101 "child1 Error: Script not invoked correctly - exiting"
MSG102 "child1 Error: Invalid option provided - exiting"
MSG103 "child1 Error: mount of file system failed - see detail information"
$ -----
$ Messages used by the "child2" script
$set child2
MSG201 "child2 Error: Script not invoked correctly - exiting"
MSG202 "child2 Error: Invalid option provided - exiting"
MSG203 "child2 Warning: File %1$s exists, another instance of child2 may be \n\
running or may have terminated prematurely. Overwriting the file"
MSG204 "child2 Error: Failure in procedure %1$s - exiting"
$ -----
$ Messages used by the "grandp" program
$set grandp
MSG301 "grandp Error: Input file does not exist - exiting"
MSG302 "grandp Error: Cannot open file - too many files opened - exiting"
MSG303 "grandp Error: Directory missing - exiting"
MSG304 "grandp Error: Unexpected file access failure - exiting"
MSG305 "grandp Error: Invalid parameters to do_grandp_stuff - exiting"
MSG306 "grandp Error: Cannot read input from file - exiting"
MSG307 "grandp Error: Cannot open file - open interrupted four times - exiting"
MSG308 "grandp Error: Failure in subprocess - exiting"
$ -----
$ Messages used to construct an AIX Error Log templates
$set errtempl
MSG901 "File System Mount Failure"
MSG902 "File system device not defined to the operating system"
MSG903 "Configuration process failed to define the file system device"
MSG904 "Run configuration process again"
MSG905 "Note any errors generated by configuration process"
MSG906 "Take any recommended actions suggested by the configuration process"
MSG907 "Consult the ffdcxpl problem determination documentation for\
further assistance"
MSG908 "File system device deleted by an authorized system user"
MSG909 "Run configuration process to redefine the file system device"
MSG910 "If attempting to disable the software, perform uninstall procedures"
MSG911 "Check if other file system devices have been deleted from the system"
MSG912 "Disk device containing file system device may have experienced failure"
MSG913 "Obtain disk drive information for file system device from SMIT"
MSG914 "Examine Error Log for failures on this device"

```

```

EMSG915 "Replace any defective disk drive hardware"
EMSG916 "File System Device Name"
EMSG917 "Stub Directory Name"
EMSG918 "Error code received from mount, listed in <sys/errno.h>"

```

Makefile for the Example Code

Note: This file includes the file, **make_template_rules**, for instructions on how to generate a *.S file from an *.E file and the message catalog. It also overrides the default setting of **GENMSGCODE** to obtain a version that can generate default messages for Error Log Template source files.

```

# "@(#)10 1.1 src/rsct/ffdc/policycode/Makefile, ffdc, rsct_rmo2, rmo2t1fg 4/19/99 16:37:31";
# IBM_PROLOG_BEGIN_TAG
# This is an automatically generated prolog.
#
#
# Licensed Materials - Property of IBM
#
# Restricted Materials of IBM
#
# (C) COPYRIGHT International Business Machines Corp. 1999
# All Rights Reserved
#
# US Government Users Restricted Rights - Use, duplication or
# disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
#
# IBM_PROLOG_END_TAG

MSG_HDRS = ffdcxpl.parent.h ffdcxpl.child1.h ffdcxpl.child2.h \
          ffdcxpl.grandp.h ffdcxpl.errtempl.h ffdcxpl.errtempl.eth
MSG_CODE = ${MSG_HDRS} ffdcxpl.child1.c ffdcxpl.grandp.c

PROGRAMS = child1 grandp
SCRIPTS = parent child2
HDRFILES = ffdcxpl.err.S.h
INCLUDES = ${MSG_CODE} ${HDRFILES}
CATFILES = ffdcxpl.cat

EXPDIR = /usr/include/
LIBS = -lct_ffdc -lct_cu
INCFLAGS = -I.
OPT_LEVEL = -g

child1_OFILES = child1.o utils.o
grandp_OFILES = grandp.o utils.o

ffdcxpl.err.S.h: ${ffdcxpl.err.S:P}
    ${ERRUPDATE} -n -h ${ffdcxpl.err.S:P}

.include <${RULES_MK}>

# The following rules file gives build instructions to convert a *.E file to
# a *.S file. The *.E file uses the labels for messages in the catalog
# (ex: EMSG001) instead of the numeric constant assigned to a message (ex: 1),
# so that the template does not have to be modified if messages change order
# in the catalog. The build instructions in the following file tell the
# build process how to convert the *.E to a *.S, where the numeric constants
# are substituted automatically for the labels.

.include <make_template_rules>

# Redefine GENMSGCODE to pick up the revised version that permits AIX Error
# Logs to be built from message labels

GENMSGCODE = /project/ode/bin/genmsgcode.perl.new

```

Daemon Source-Code Example

The source code module in this appendix makes use of the First Failure Data Capture (FFDC) application programming interfaces to record information about failure conditions for use in later problem analysis. The example used in this appendix is of a daemon process named **ffdcclnt**, started by the **inetd** process.

ffdcclnt is designed to make a request of a network service named **ffdcserv**, obtain the results of this request, and place the results of the request in a file where clients of the **ffdcclnt** daemon can retrieve the results.

The example demonstrates these concepts:

- Using the AIX Error Log through the FFDC interfaces to capture failure information.
- Associating a failure condition experienced in one application (**ffdcclnt**) directly with the condition encountered in another application (**ffdcserv**) which caused the failure condition.
- Providing an identifier for a failure experienced in an application (**ffdcclnt**) to the client of the application, so the client may associate any failure it experiences as a direct result of this condition to the application's failure.

The client.c Program

```
/*
 * Module Name:    client.c
 *
 * Purpose:    Used to demonstrate the use of First Failure Data Capture
 *             (FFDC) application programming interfaces from a C language
 *             program. The concepts demonstrated by this program are:
 *             - Use of the AIX Error Log for recording failure
 *               information from a process not invoked as part of
 *               a parent/child relationship (which makes use of the
 *               FFDC Error Stack impractical)
 *             - Differentiation of various error conditions:
 *               + permanent errors that will persist until the
 *                 condition is investigated and resolved
 *               + errors caused by lack of system resources,
 *                 which may not exist in the next execution of
 *                 this application but will cause this execution
 *                 of the application to fail
 *               + errors for which the application compensated
 *                 by using an alternate resource
 *               + failures directly caused by the failure of
 *                 another application
 *             - Recording of the various levels of failure and status
 *               experienced by the application to the AIX Error Log
 *               through the FFDC interfaces
 *
 * Compilation:    This module must be linked to the following RCST libraries:
 *                 libct_ffdc.a  For FFDC function
 *                 This module must also be linked with the Reliable Daemons
 *                 libraries:
 *                 libdae.a  libdae_bsd.a
 *
 * Design:    "client.c" is written to be a daemon process that will be known
 *            by the name of "ffdcclnt". "ffdcclnt" is a client of an
 *            application called "ffdcserv", which is another daemon process
 *            monitoring a known port somewhere on the network. The basic
 *            flow of control for the application is:
 *            - Initialize itself as a daemon, using the Reliable
 *              Daemons library routines
 *            - Change the working directory to its own directory,
 *              so any "core" files are not placed in the process
 *              owner's $HOME (this is needed because this process
 *              can be owned by "root", which owns many processes)
 *            - Obtain the name of an "ffdcserv" server system from
 *              a configuration file
 */
```

- * - Establish a TCP/IP socket connection to the "ffdcserv" server system
- * - Transmit a data packet to the "ffdcserv" application to request a specific service from that application
- * - Monitor the socket connection to "ffdcserv" for any response from the service - the response is expected within a finite period of time
- * - Receive the response from the "ffdcserv" application
- * - Write its results in binary format to a file, where its own client would expect to find the information.
- * - End execution

* At each stage of this process's execution, it may experience a failure condition. Because this process is executing as a daemon, it has no control terminal where failure information can be displayed. The AIX Error Log is used to record information about these failure conditions so that the information can be available for detection and later use in problem determination procedures.

* Each failure condition is reported using an AIX Error Log template tailored to the failure condition. These templates are built so that the Err_Desc field indicates the type of failure encountered in a succinct manner, so that the system administrator can determine the type of failure from a brief AIX Error Log report (generated by "errpt" without any options). System administrators should not be forced to obtain a detailed AIX Error Log report (generated by "errpt -a") to understand a specific failure report; doing so on more than a handful of nodes can result in output too large for use by mere humans.

* AIX Error Log templates are built to be specific to a failure condition. They are built to convey all necessary information in the body of the template, reserving the Detail Data fields for diagnostic information of use to the software manufacturer only. Only in rare cases should the Detail Data be used to convey information to the system administrator, and in those cases, the information in the template must contain data only; Detail Data should not contain "messages" or descriptive text, because this information is not translatable once it is recorded.

* "client.c" differentiates between several different failure types:

- * - External failures, those caused by incorrect use of the application. These are recorded using the FFDC Event Type of FFDC_DEBUG and AIX Error Log Type UNKN. Information is recorded more as information to explain why the application decided to terminate, instead of a failure that requires immediate attention. These are not labelled as "permanent" failures, since the failure will no longer occur when the program is used in the correct environment.
- * - Resource failures, those caused by an essential being denied, being unavailable, or failing in an unforeseen manner. These are recorded using two kinds of reports:
 - * + FFDC event type FFDC_RECOV and AIX Error Log template type TEMP if a backup resource was located. This record is made so that the administrator knows that there is a condition that should be checked out, but it is not imperative that it be checked out immediately.
 - * + FFDC event type FFDC_ERROR and AIX Error Log type PERM if no backup resource is available, or if the code cannot make an attempt to find a suitable alternate resource (such as in the case of no dynamic memory). This record is made to indicate that the failure will

```

*           continue to exist until the failing resource
*           is repaired or made available.
*   - Internal failures, a.k.a. "code bugs, cause by
*     incorrect coding within the module (for instance, a
*     C library routine responding with an EINVAL error
*     code, which should not happen in product level code).
*     FFDC event type FFDC_ERROR and AIX Error Log type PERM
*     is used to record this failure, to indicate that the
*     condition will persist until the code is repaired.
*     This record type is used even in code paths that are
*     not commly executed, because this code does not use
*     a mechanism to disable the code path at run time to
*     prevent it from occurring again.
*/

```

```

#include <dae.h>           /* Reliable daemons routines */
#include <errno.h>        /* Error codes from C library */
#include <fcntl.h>        /* File control primitives */
#include <libgen.h>       /* Directory name processing */
#include <memory.h>       /* Memory manipulation */
#include <netdb.h>        /* Host & service information */
#include <unistd.h>       /* Process owner and pipes */
#include <sys/socket.h>   /* Socket definitions */
#include <sys/socketvar.h> /* More socket definitions */
#include <sys/select.h>   /* Notification of socket input */
#include <sys/time.h>     /* Pausing the client */
#include <sys/types.h>    /* System data types */
#include <netinet/in.h>   /* Internet protocols */
#include <rsct/ct_ffdc.h> /* FFDC interfaces and types */
#include <ffdcclnt.err.S.h> /* Error template definitions */

```

```

#define RESOURCE_NAME      "ffdcexpl"
#define LPP_NAME           "PSSP"
#define VERSION_ID         "%I%"
#define APPLICATION_NAME   "ffdcclnt"
#define SERVICE_NAME       "ffdcserv"
#define SERVICE_PROTOCOL   "tcp"
#define CONFIG_FILE        "/tmp/ffdc.conf"
#define OUTPUT_FILE        "/tmp/ffdc.out"
#define TIMEOUT_SECONDS    15

```

```

/*
 * Data types used for socket communications
 */

```

```

typedef enum versn_code    version_t;
enum versn_code {
    VERSION_1 = 1,        /* Supported client/server */
    VERSION_2,            /* versions... */
    VERSION_3
};

```

```

typedef enum comnd_code    command_t;
enum comnd_code {
    COMMAND_1 = 1,        /* Supported server instructns */
    COMMAND_2,
    COMMAND_3
};

```

```

typedef enum reply_code    reply_t;
enum reply_code{
    FCA_SUCCESS = 1,      /* Success... */
    FCA_DENY,            /* Request denied... */
    FCA_FAILED           /* Cannot fulfill request... */
};

```

```

typedef struct client_request    client_req_t;
struct client_request {
    /* Makes requests of server... */
};

```

```

    version_t version; /* Version code of requestor */
    command_t command; /* Specific request being made */
};

typedef struct server_response    server_resp_t;
struct server_response {          /* Server's answer to request */
    version_t version; /* Version code of server */
    reply_t    reply;   /* Indicates success or failure */
    fc_eid_t   ffdcid;  /* Identifier for failure rec */
    int        len;     /* Length of data (success) */
                    /* (data follows packet on sock)*/
};

typedef struct client_response    client_resp_t;
struct client_response {          /* Client's response to its own */
    /* client: */
    version_t version; /* Version code of client appl */
    reply_t    reply;   /* Indicates success or failure */
    fc_eid_t   ffdcid;  /* Identifier for failure recd */
    int        len;     /* Length of data in success */
                    /* cases */
                    /* (data follows this structure */
                    /* in output file) */
};

/*
 * Macros used by the source code
 */
#define CLEAR_DETAIL_BUF (memset((void *) details, 0, (sizeof(char) * 100)))

#define WRITE_CLIENT_RESULTS(fildes, response, failure_id, data_len, data_buf) \
{
    client_resp_t response_buf;
    memset(&response_buf, 0, sizeof(response_buf));
    response_buf.version = VERSION_1;
    response_buf.reply = response;
    if (fc_eid_is_set(failure_id)) {
        strcpy(response_buf.ffdcid, failure_id);
    }
    if (data_len > 0) {
        response_buf.len = data_len;
    }
    if (fildes != -1) {
        write(fildes, &response_buf, sizeof(response_buf));
        if (data_len > 0) {
            write(fildes, (data_buf), data_len);
        }
        close(fildes);
    }
}

/*
 * Global data area definitions. Some are needed to correctly handle the
 * reception of termination signals.
 */
int sckt;

/*****
 * Function Name: term_handler
 * Description:   SIGTERM handler for this application. The handler
 *               checks if a socket connection exists, and if so,
 *               closes the socket connection. The handler also
 *               makes a recording to the AIX Error Log to indicate
 *               that the application is terminating.
 * Usage:        (void) term_handler(int signal)
 * Parameters:   signal      Value of the signal that terminated
*****/

```

```

*           the application - provided by AIX
* Return Codes:  None - routine does not return to caller
* Exit Status:   This routine forces an exit of the application, and
*               provides the value of the terminating signal as the
*               exit status for the application.
*****/
void
term_handler(int sigval)
{
    extern int    sckt;        /* Socket file descriptor    */
    int          rc;         /* Function call return code */
    fc_eid_t     fid;        /* FFDC Failure Identifier   */

    /*
     * Release socket if a connection looks like it's been established
     */
    if (-1 != sckt) {
        close(sckt);
    }
    /*
     * Generate termination record in persistent storage. Note that
     * this record is "informational" and records a status change,
     * not an "error" reporting a "permanent" or "emergency" condition.
     */
    fc_eid_init(fid);
    rc = fc_log_error(&fid, NULL, RESOURCE_NAME, LPP_NAME, __FILE__,
                     VERSION_ID, __LINE__, (void *) NULL, FFDC_STATE,
                     ERRID_FCLNT_TERMSIG_ST, (void *) &sigval,
                     sizeof(int), NULL,
                     "ffdcclnt exiting - termination signal received");

    /*
     * Exit with the signal value as the exit status
     */
    exit(sigval);
}

/*****
* Function Name:  init_client
* Description:    Initializes the daemon process and sets up a termination
*               signal handler to drop any socket cponnection when an
*               SIGTERM signal is received. Uses the Reliable Daemons
*               library to set up this function.
* Usage:         int = init_client(fc_eid_t *returned_fid)
* Parameters:    returned_fid  Location where this routine would
*               return an FFDC Failure Identifier, if
*               an unrecoverable failure occurred.
* Return Codes:  0  Daemon initialization successful
*               1  Daemon was started by the SRC, which is not
*               supported by this daemon
*               2  Daemon could not initialize because dae_init
*               experienced a failure
*               99 Internal failure, indicating possible code bug
*****/
int
init_client(fc_eid_t *returned_fid)
{
    int          rc;         /* Function call return code */
    int          linepos;    /* Approx. spot of failure    */
    dae_parent_t parent;     /* How daemon is started     */
    dae_parent_t *pptr;      /* Data pointer               */
    dae_error_detail_t einfo; /* dae_init failure information */
    dae_error_detail_t *eptr; /* Data pointer               */
    void         *p;         /* Temporary pointer         */
    char         details[100]; /* Details for AIX Error Log */
    extern int    sckt;      /* Socket file descriptor     */
    extern void   term_handler(); /* SIGTERM signal handler */

```

```

/*
 * Initialize the socket file descriptor and set up termination
 * signal handler.
 */
sckt = -1;
dae_init_term_sig(term_handler, 0);
/*
 * Initialize the daemon. Two types of errors can be logged by this
 * routine:
 * - Errors indicating a misuse of the dae_init() interface,
 *   which is considered an "internal" failure, or
 * - Errors in dae_init() itself, which is considered a "resource"
 *   failure
 */
pptr = &parent
eptr = &einfo
parent = DAE_P_INETD | DAE_P_OTHER;
memset(eptr, 0, sizeof(einfo));
linepos = __LINE__;
rc = dae_init(pptr, eptr);
switch(rc) {
    /* Check for success first */
    case DAE_E_OK:
        /*
         * Change directory of the running process to a specific
         * directory. This is done so that a "core" file is
         * written to some other directory beside the process
         * owner's $HOME directory. In this example, the
         * directory is changed to /tmp, which is not very
         * practical, but it serves the purpose in the example.
         * Actual product level code would change the directory
         * to a directory reserved for use by the code itself.
         * The application also has to document where the "core"
         * file would be dumped in its user documentation, so
         * the user is not forced to search everywhere for it.
         */
        chdir(dirname(CONFIG_FILE));
        return(0);
        /* Was daemon started in an unsupported manner? */
    case DAE_E_PWRONG:
        return(1);
        /* Check for "resource" failures in dae_init() */
    case DAE_E_PERROR:
    case DAE_E_CHILD:
    case DAE_E_SIGNAL:
    case DAE_E_CLOSE:
    case DAE_E_DEVNULL:
    case DAE_E_CHDIR:
    case DAE_E_SETPSALLOC:
        CLEAR_DETAIL_BUF;
        p = (void *) details;
        memcpy(p, &rc, sizeof(int));
        p += sizeof(int);
        strncpy(p, einfo.dae_routine, 20);
        p += sizeof(char) * 20;
        strncpy(p, einfo.dae_error_string, 76);
        p += sizeof(char) * 76;
        rc = fc_log_error(returned_fid, NULL, RESOURCE_NAME,
                        LPP_NAME, __FILE__, VERSION_ID,
                        linepos, (void *) NULL, FFDC_ERROR,
                        ERRID_FCLNT_DAEINIT_ER, details,
                        (p - (void *) details), NULL,
                        "ffdcclnt failed, initialization error");
        return(2);
    /*
     * All other cases indicate a misuse of the dae_init routine
     * by this program. Log error information to persistent

```

```

    * storage, and include the parameters used in the call to
    * dae_init
    */
default:
    CLEAR_DETAIL_BUF;
    p = (void *) details;
    memcpy(p, &rc, sizeof(int));
    p += sizeof(int);
    memcpy(p, &parent, sizeof(int));
    p += sizeof(int);
    memcpy(p, &pptr, sizeof(int));
    p += sizeof(int);
    memcpy(p, &einfo, sizeof(int));
    p += sizeof(int);
    rc = fc_log_error(returned_fid, NULL, RESOURCE_NAME,
                     LPP_NAME, __FILE__, VERSION_ID,
                     linepos, (void *) NULL, FFDC_ERROR,
                     ERRID_FCLNT_INTERN7_ER, details,
                     (p - (void *) details), NULL,
                     "ffdcclnt failed, initialization error");

    return(99);
}
/* NOTREACHED */
}

/*****
 * Function Name: open_output_file
 * Description:   Creates an output file where the "ffdcclnt" application
 *               will provide its output to its own client. This file
 *               contains data in binary format. The client is expected
 *               to know the format of the data within the file.
 * Usage:        int = open_output_file(int *fildes,
 *                                     fc_eid_t *returned_fid)
 * Parameters:   fildes      Location where the output file's
 *                       descriptor will be placed.
 *               returned_fid In case of internal failure, contains
 *                       the FFDC Failure Identifier of the
 *                       record made for the failure
 * Return Codes: 0    Successful completion
 *               1    Permissions have been altered on the /tmp
 *                       directory
 *               2    System file table is full, no further files
 *                       can be opened
 *               3    No space remaining on the /tmp file system
 *               4    User's disk space usage quota is exceeded
 *               5    Unable to open file, attempt continuously
 *                       interrupted
 *               99   Internal failure, indicating possible code bug
 *****/
int
open_output_file(int *fildes, fc_eid_t *returned_fid)
{
    int    rc;      /* Function call return code */
    int    i;      /* Loop counter */
    int    outmode; /* Mode used on open() */
    int    linepos; /* Approx. location of failure */
    int    local_errno; /* So errno doesn't get lost... */
    void   *p;     /* Temporary pointer */
    char   details[100]; /* Details for AIX Error Log */

    /*
     * The location of the output is hardcoded and published in the
     * "ffdcclnt" user documentation. Make 5 attempts maximum to open
     * the file in cases where the open() is interrupted.
     */
    outmode = O_WRONLY | O_CREAT | O_TRUNC;
    for (i = 0 ; i < 5 ; i++) {

```

```

linepos = __LINE__;
rc = open(OUTPUT_FILE, outmode, 0660);
local_errno = errno;
if (0 == rc) {
    (*fildes) = rc;
    return(0);
}
switch (local_errno) {
    case EACCES: return(1);
    case EINTR: continue;
    case ENFILE: return(2);
    case ENOSPC: return(3);
    case EDQUOT: return(4);
    default: /*
        * Indicates a misuse of the open()
        * routine, or problems with the
        * hardcoded path name. Record info
        * on this failure to persistent
        * storage. NOTE: Although this failure
        * cannot be seen by the application's
        * client (since the output file cannot
        * be created), failure information is
        * recorded anyway.
        */
        CLEAR_DETAIL_BUF;
        p = (void *) details;
        strcpy((char *) p, "open");
        p += sizeof(char) * 10;
        memcpy(p, &outmode, sizeof(int));
        p += sizeof(int);
        memcpy(p, &local_errno, sizeof(int));
        p += sizeof(int);
        rc = fc_log_error(returned_fid, NULL,
            RESOURCE_NAME, LPP_NAME,
            __FILE__, VERSION_ID, linepos,
            (void *) NULL, FFDC_ERROR,
            ERRID_FCLNT_INTERNAL_ER, details,
            (p - (void *) details), NULL,
            "ffdcclnt failed, internal file creation error");
        return(99);
    }
}
return(5);
}

/*****
* Function Name: pause_client
* Description: Pauses the execution of the client for a sepcific
* mount of time. Used by the client during retry logic.
* Usage: (void) pause_client(void)
* Parameters: None
* Return Codes: None
*****/
void
pause_client()
{
    struct timestruc_t wait_time;

    /*
     * Pause the program for 17 microseconds, or until interrupted
     */
    wait_time.tv_sec = 0;
    wait_time.tv_nsec = 17 * NS_PER_MSEC;
    nsleep(&wait_time, 0);
    return;
}

```

```

}

/*****
* Function Name: socket_read
* Description:   Reads information from a network socket connection.
*               Since all data may not be immediately available to
*               read from the socket, this routine loops until all
*               data expected from the socket is read, or until an
*               error is detected. The caller of this routine should
*               verify that there is data available on the socket to be
*               read before calling this routine, or this routine will
*               cause the caller to "hang" in this routine.
* Usage:        int = socket_read(int sckt, void *buffer,
*                               size_t expect, fc_eid_t *returned_fid)
* Parameters:   sckt      Indicates a previously established
*               socket connection to a service
*               buffer    Location where data is to be placed
*               once read from the socket
*               expect    Number of bytes to be read from the
*               socket
*               returned_fid Location where this routine would
*               return an FFDC Failure Identifier, if
*               an unrecoverable failure occurred.
* Return Codes: 0      Data successfully read from the socket
*               99     Internal error, indicates possible code bug
*****/
int
socket_read(int sckt, void *buffer, size_t expected, fc_eid_t *returned_fid)
{
    int      rc;      /* Function call return code */
    int      i;      /* Loop counter */
    int      linepos; /* Approx. location of failure */
    int      local_errno; /* So errno won't get lost... */
    size_t   bytes;   /* # bytes read this iteration */
    size_t   remain;  /* # bytes left to be read */
    void     *p, *q;  /* Temporary pointers */
    char     details[100]; /* Details to AIX Error Log */

    /*
     * Make repeated attempts to get data from the socket. If an
     * error occurs that indicates a possible failure in the coding of
     * this routine, record information about the failure to persistent
     * storage. Note that this routine needs to know exactly how the
     * Detail Data fields in the AIX Error Log template are laid out,
     * and that the code ensures that the data falls on the proper
     * data boundaries (even if that means fields are padded to the
     * proper length).
     */
    q = buffer;
    for (i = 0 ; ; i++) {
        linepos = __LINE__;
        bytes = read(sckt, q, remain);
        local_errno = errno;
        if (-1 == bytes) {
            if (EAGAIN == local_errno) {
                pause_client();
                continue;
            }
        }
        CLEAR_DETAIL_BUF;
        p = (char *) details;
        strcpy((char *) p, "socket_read");
        p += sizeof(char) * 20;
        strcpy((char *) p, "read");
        p += sizeof(char) * 8;
        memcpy(p, &local_errno, sizeof(int));
        p += sizeof(int);
        memcpy(p, &sckt, sizeof(int));
    }
}

```

```

    p += sizeof(int);
    memcpy(p, &i, sizeof(int));
    p += sizeof(int);
    memcpy(p, &expected, sizeof(int));
    p += sizeof(int);
    memcpy(p, &remain, sizeof(int));
    p += sizeof(int);
    memcpy(p, &q, sizeof(int));
    p += sizeof(int);
    rc = fc_log_error(returned_fid, NULL, RESOURCE_NAME,
                     LPP_NAME, __FILE__, VERSION_ID,
                     linepos, (void *) NULL, FFDC_ERROR,
                     ERRID_FCLNT_INTERN6_ER, details,
                     (p - (void *) details), NULL,
                     "ffdcclnt failed, read error");

    return(99);
}
/* Continue reading if more data is expected from the socket */
if (bytes != remain) {
    q += bytes;
    remain -= bytes;
    continue;
}
}
return(0);
}

/*****
* Function Name: connect_to_service
* Description:   Connect to a service on a specific node. The service
*               name is hardcoded, while the caller has the freedom to
*               specify the host to which to connect.
* Usage:        int = connect_to_service(char *service_host,
*                                       int *sckt,
*                                       fc_eid_t *returned_fid)
* Parameters:   service_host  The host where the service is to be
*                       used
*               sckt         Location where this routine will place
*                       the socket description on successful
*                       completion
*               returned_fid  Location where this routine would
*                       return an FFDC Failure Identifier, if
*                       an unrecoverable failure occurred.
* Return Codes: 0   Successfully connected to service on the host
*               1   Incorrect usage - service_host is a NULL string
*               2   service_host cannot be found
*               3   service_host is a valid host name, but its
*                   address cannot be found
*               4   Unable to obtain host information for
*                   service_host
*               5   Cannot obtain information for the ffdcserv
*                   service from /etc/services
*               6   Cannot establish a socket because this process
*                   has too many files opened
*               7   Cannot establish a socket because of system
*                   resource problems
*               8   Cannot connect to remote system - address of
*                   remote system is not available, or no path to
*                   the host exists
*               9   Cannot connect to remote system - connection
*                   timed out
*               10  Cannot connect to remote system - remote system
*                   refused the connection attempt
*               99  Internal failure, indicating likely code bug
* Notes:        In cases of internal errors only (code 99), this routine
*               records information on the failure to persistent
*               storage via FFDC.
*****/

```

```

*****/
int
connect_to_service(char *service_host, int *sckt, fc_eid_t *returned_fid)
{
    int          rc; /* Function call return code */
    int          i; /* Loop counter */
    int          sock; /* Socket file descriptor */
    int          lpos; /* Line of code position */
    int          local_errno; /* Retain errno setting */
    char         details[100]; /* Detailed failure info */
    void         *p; /* Temporary pointer */
    struct hostent hbuf; /* Host providing the service */
    struct servent sbuf; /* Info on the service itself */
    struct sockaddr_in serv; /* Info for socket connection */
    union {
        struct hostent_data host;
        struct servent_data serv;
    }           dbuf; /* Needed for re-entrant calls */

    /*
     * Check input for obvious problems - if this fails, the caller is
     * responsible for figuring out why and making any permanent record
     * of the failure.
     */
    if ((char *) NULL == service_host || NULL == (*service_host)) {
        return(1);
    }
    /*
     * Get information for the service used - note that this assumes
     * that the service information on this node will match the information
     * on a remote node for the service.  If the service cannot be located,
     * this routine will assume that there is a configuration error with
     * the ffdcserv service and record an error log entry to report this.
     */
    memset((void *) &sbuf, 0, sizeof(sbuf));
    memset((void *) &dbuf, 0, sizeof(dbuf));
    lpos = __LINE__;
    rc = getservbyname_r(SERVICE_NAME, SERVICE_PROTOCOL, &sbuf,
                        &(dbuf.serv));

    if (-1 == rc) {
        /*
         * Note that when this section of code creates a buffer to
         * write to the AIX Error Log, it needs to know how the detail
         * data fields (after the first three required by fc_log_error)
         * are defined and laid out.
         */
        CLEAR_DETAIL_BUF;
        snprintf(details, 100, "%.16s%.16s%.4s", APPLICATION_NAME,
                 SERVICE_NAME, SERVICE_PROTOCOL);
        fc_log_error(returned_fid, NULL, RESOURCE_NAME, LPP_NAME,
                     __FILE__, VERSION_ID, lpos, (void *) NULL, FFDC_ERROR,
                     ERRID_FCLNT_SVCNOTF_ER, details, strlen(details), NULL,
                     "ffdcserv Service not configured properly");
        return(5);
    }
    /*
     * Attempt to get information for the specified host - give up after
     * 5 retry attempts.  No information is recorded if this fails.  This
     * is to give the caller the chance to decide whether or not this is
     * a critical failure (maybe the caller can use a backup).
     */
    for (i = 0 ; i < 5 ; i++) {
        memset((void *) &hbuf, 0, sizeof(hbuf));
        memset((void *) &dbuf, 0, sizeof(dbuf));
        rc = gethostbyname_r(service_host, &hbuf, &(dbuf.host));
        if (-1 == rc) {
            switch (h_errno) {

```

```

        case HOST_NOT_FOUND:      return(2);
        case NO_ADDRESS:         return(3);
        case SERVICE_UNAVAILABLE: return(4);
        case NO_RECOVERY:        return(4);
        case TRY_AGAIN:          pause_client();
                                continue;
    }
}
}
if (5 <= i) {
    return(4);
}
/*
 * Set up a socket to use the service on the host requested by the
 * caller. If a failure occurs that indicates a possible coding
 * error, record data about the failure to persistent storage so that
 * the failure can be identified and diagnosed later (items such as
 * the parameters used in the call). Note that this code needs to
 * understand how the detail data fields in the AIX Error Log
 * template are laid out.
 */
lpos = __LINE__;
sock = socket(AF_INET, SOCK_STREAM, 0);
local_errno = errno;
if (-1 == sock) {
    switch(local_errno) {
        case EMFILE:  return(6);
        case ENOBUFS: return(7);
        default:     CLEAR_DETAIL_BUF;
                    p = (void *) details;
                    strcpy((char *) p,
                        "connect_to_service");
                    p += sizeof(char) * 20;
                    strcpy((char *) p,
                        "socket(AF_INET, SOCK_STREAM, 0)");
                    p += sizeof(char) * 40;
                    memcpy(p, (void *) &local_errno,
                        sizeof(int));
                    p += sizeof(int);
                    rc = fc_log_error(returned_fid, NULL,
                        RESOURCE_NAME, LPP_NAME,
                        __FILE__, VERSION_ID, lpos,
                        (void *) NULL, FFDC_ERROR,
                        ERRID_FCLNT_INTERN1_ER, details,
                        (p - (void *) details), NULL,
                        "ffdcclnt failed, socket error");
                    return(99);
    }
}
/*
 * Connect the socket to the service on the requested system. If a
 * failure occurs that indicates a possible coding error, record data
 * about the failure to persistent storage so that the failure can
 * be identified and diagnosed later (items such as the parameters
 * used in the call).
 */
memset((void *) &serv, 0, sizeof(serv));
memcpy(&(serv.sin_addr), &(hbuf.h_addr), hbuf.h_length);
serv.sin_family = AF_INET;
serv.sin_port = htons(sbuf.s_port);
lpos = __LINE__;
rc = connect(sock, (struct sockaddr *) &serv,
    sizeof(struct sockaddr_in));
local_errno = errno;
if (-1 == rc) {
    switch(local_errno) {
        case EADDRNOTAVAIL: close(sock);

```

```

        return(8);
    case ETIMEDOUT:    close(sock);
        return(9);
    case ECONNREFUSED: close(sock);
        return(10);
    case ENETUNREACH: close(sock);
        return(8);
    default: close(sock);
        CLEAR_DETAIL_BUF;
        p = (void *) details;
        strcpy((char *) p,
            "connect_to_service");
        p += sizeof(char) * 20;
        strcpy((char *) p, "connect");
        p += sizeof(char) * 8;
        memcpy(p, &local_errno, sizeof(int));
        p += sizeof(int);
        memcpy(p, &sock, sizeof(int));
        p += sizeof(int);
        memcpy(p, &serv, sizeof(int));
        p += sizeof(int);
        memcpy(p, &serv,
            sizeof(struct sockaddr_in));
        p += sizeof(int);
        memcpy(p, &(serv.sin_addr),
            sizeof(int));
        p += sizeof(int);
        memcpy(p, &(serv.sin_port),
            sizeof(int));
        p += sizeof(int);
        memcpy(p, &(serv.sin_family),
            sizeof(int));
        p += sizeof(int);
        rc = fc_log_error(returned_fid, NULL,
            RESOURCE_NAME, LPP_NAME,
            _FILE_, VERSION_ID, lpos,
            (void *) NULL, FFDC_ERROR,
            ERRID_FCLNT_INTERN2_ER, details,
            (p - (void *) details), NULL,
            "ffdcclnt failed, connect error");
        return(99);
    }
}
/*
 * Socket connected to service on remote node. Provide appropriate
 * indication to the caller.
 */
(*sckt) = sock;
return(0);
}

/*****
 * Function Name:  get_server_names
 * Description:    Obtains host names for a service from a configuration
 *                file. One or more names are expected to be present in
 *                the configuration file. An empty file or a missing
 *                file is considered to be a configuration error. To
 *                simplify this example, a maximum of five server names
 *                will be read from the configuration file. To further
 *                simplify, the configuration file is set to be a specific
 *                location which can be hardcoded.
 * Usage:         rc = get_server_names(char *servers[],
 *                fc_eid_t *returned_fid)
 * Parameters:    servers      References an array of character
 *                strings
 *                returned_fid Location where this routine would
 *                return an FFDC Failure Identifier, if

```

```

*          an unrecoverable failure occurred.
* Return Codes: 0  Server systems successfully read from configu-
*                ration file
*                1  Configuration file, or the directory containing
*                the configuration file, does not exist
*                2  Cannot access the configuration file in R/O
*                mode
*                3  Caller has too many files opened, cannot open
*                the configuration file.
*                4  Cannot open file, each attempt was interrupted
*                5  Configuration file is empty.
*                99 Internal failure, indicating likely code bug
* Notes:       In cases of internal errors only (code 99), this routine
*                records information on the failure to persistent
*                storage via FFDC.
*****/
int
get_server_names(char servers[][80], fc_eid_t *returned_fid)
{
    int    rc;        /* Function call return code */
    int    i;        /* Loop counter */
    int    linepos;  /* Used in failure report */
    int    local_errno; /* Prevent losing "errno" value */
    void   *p;       /* Temporary pointer */
    char   inline[80]; /* Input from config file */
    char   details[100]; /* AIX Error Log details */
    FILE   *config_file; /* Access to configuration file */

    /*
     * Access the configuration file. If an "unexpected" failure in the
     * access attempt occurs, record information about the failure to
     * persistent storage, since such failures indicate a possible coding
     * problem. The code "expects" that the file may not exist, or that
     * permissions may have changed, and other "usual" failures in the
     * resource. The code does not "expect" to get EINVAL failures.
     */
    config_file = (FILE *) NULL;
    for (i = 0 ; i < 5 ; i++) {
        linepos = __LINE__;
        config_file = fopen(CONFIG_FILE, "r");
        local_errno = errno;
        if ((FILE *) NULL == config_file) {
            switch(local_errno) {
                case EINTR: /* Retry the fopen() */
                    continue;
                case ENOENT: /* File missing */
                    /* Intentional fall-through */
                case ENOTDIR: /* Directory not available */
                    return(1);
                case EACCES: /* Cannot read file */
                    return(2);
                case ENFILE: /* Too many opened files */
                    return(3);
                default: /* Unexpected - indicated a */
                    /* possible code bug */
                    CLEAR_DETAIL_BUF;
                    p = (void *) details;
                    strcpy((char *) p, "get_server_name");
                    p += sizeof(char) * 20;
                    strcpy((char *) p, "fopen");
                    p + sizeof(char) * 8;
                    memcpy(p, &local_errno, sizeof(int));
                    p += sizeof(int);
                    strcpy((char *) p, CONFIG_FILE);
                    p += sizeof(char) * 60;
                    strcpy((char *) p, "r");
                    p += sizeof(char) * 4;
            }
        }
    }
}

```

```

                rc = fc_log_error(returned_fid, NULL,
                                RESOURCE_NAME, LPP_NAME,
                                FILE_, VERSION_ID, linepos,
                                (void *) NULL, FFDC_ERROR,
                                ERRID_FCLNT_INTERN3_ER, details,
                                (p - (void *) details), NULL,
                                "ffdcclnt failed, fopen error");
                return(99);
            }
        }
    }
}
if ((FILE *) NULL == config_file && 5 <= i) {
    return(4);
}
/*
 * Obtain list of servers (5 maximum) from the configuration file.
 */
memset((void *) inline, 0, (sizeof(char) * 80));
for (i = 0 ; i < 5 ; i++) {
    linepos = __LINE__;
    if (NULL == fgets(inline, 80, config_file)) {
        local_errno = errno;
        if (0 == i) {
            /* No servers listed - configuration error */
            fclose(config_file);
            return(5);
        }
        break;
    }
    strcpy(servers[i], inline);
}
/*
 * At least one server was read. Return success indicator to caller.
 */
fclose(config_file);
return(0);
}

/*****
 * Function Name: send_service_request
 * Description:   Sends a data packet to the server to request a service
 *               from it.
 * Usage:        int = send_service_request(int sckt,
 *                                       fc_eid_t *returned_fid)
 * Parameters:   sckt      Previously established connection to
 *                   the service
 *               returned_fid Location where this routine would
 *                   return an FFDC Failure Identifier, if
 *                   an unrecoverable failure occurred.
 * Return Codes: 0      Request successfully sent
 *               1      Socket connection dropped prematurely by the
 *                   server
 *               99     Internal failure, indicating likely code bug
 * Notes:        For the sake of brevity, this routine does not attempt
 *               to verify the identity of the service on the other end
 *               of the socket before making the service request. The
 *               routine also assumes that the service is also not
 *               verifying identity as well. If this code were true
 *               product level code and part of the trusted computing
 *               base, the code should use DCE or Kerberos to ensure
 *               that the partner can be trusted.
 *****/

int
send_service_request(int sckt, fc_eid_t *returned_fid)
{
    int    i;        /* Loop ctr - for error only */

```

```

int      rc;          /* Function call return code */
int      linepos;    /* Approx. location of failure */
int      local_errno; /* Make sure "errno" isn't lost */
size_t   bytes;      /* Bytes transferred via socket */
size_t   remain;     /* Bytes remaining to be sent */
size_t   orig_len;   /* Orig. # bytes to be sent */
void     *p, *q;     /* Temporary pointers */
char     details[100]; /* AIX Error Log details */
client_req_t cbuf;   /* Data sent to service */

/*
 * Set up request data to be sent to service
 */
orig_len = remain = sizeof(cbuf);
memset((void *) &cbuf, 0, remain);
cbuf.version = VERSION_1;
cbuf.command = COMMAND_2;
/*
 * Transmit request to service over previously established socket.
 */
q = (void *) &cbuf
for (i = 1 ;; i++) {
    linepos = __LINE__;
    bytes = write(sckt, q, remain);
    local_errno = errno;
    if (-1 == bytes) {
        /*
         * Check for failures in the write attempt. Anything
         * but an EPIPE (which indicates a broken socket link)
         * is considered a potential coding error. To help
         * understand why this failure occurred, a failure
         * report is logged to indicate how much data was
         * being transmitted, the contents of the data, and
         * the loop iteration (to try and detect an infinite
         * loop).
         */
        switch(local_errno) {
            case EPIPE: /* Socket dropped by server */
                return(1);
            default: /* Possible coding bug */
                CLEAR_DETAIL_BUF;
                p = (void *) details;
                strcpy((char *) p,
                    "send_service_request");
                p += sizeof(char) * 20;
                strcpy((char *) p, "write");
                p += sizeof(char) * 8;
                memcpy(p, &local_errno, sizeof(int));
                p += sizeof(int);
                memcpy(p, &i, sizeof(int));
                p += sizeof(int);
                memcpy(p, &orig_len, sizeof(int));
                p += sizeof(int);
                memcpy(p, &remain, sizeof(int));
                p += sizeof(int);
                if (56 < remain) {
                    memcpy(p, q, remain);
                }
                else {
                    memcpy(p, q, 56);
                }
                p += sizeof(char) * 56;
                rc = fc_log_error(returned_fid, NULL,
                    RESOURCE_NAME, LPP_NAME,
                    __FILE__, VERSION_ID, linepos,
                    (void *) NULL, FFDC_ERROR,
                    ERRID_FCLNT_INTERN4_ER, details,

```

```

                (p - (void *) details), NULL,
                "ffdcclnt failed, write error");
        return(99);
    }
}
if (remain > bytes) {
    p += bytes;
    remain -= bytes;
    continue;
}
break;
}
/*
 * Request successfully transmitted.
 */
return(0);
}

/*****
 * Function Name: detect_server_reply
 * Description:   Examines a socket connection to a service, looking for
 *               input being sent from that service
 * Usage:        int = detect_server_reply(int sckt,
 *               fc_eid_t *returned_fid)
 * Parameters:   sckt      Socket previously connected to a service
 *               returned_fid Location where this routine would
 *                       return an FFDC Failure Identifier, if
 *                       an unrecoverable failure occurred.
 * Return Codes: 0      Reply successfully received
 *               1      Service did not respond in the time permitted
 *               2      Unable to determine if server has responded
 *               99     Internal failure, indicating likely code bug
 *****/
int
detect_server_reply(int sckt, fc_eid_t *returned_fid)
{
    int    rc;          /* Function call return code */
    int    i;          /* Loop counter */
    int    linepos;    /* Line that may have failed */
    int    local_errno; /* So errno won't be lost */
    void   *p;         /* Temporary pointer */
    fd_set read_set;   /* Used by select routine */
    fd_set *read_set_addr; /* Address of structure */
    struct timeval timeout; /* Time to wait for response */
    char   details[100]; /* Details to AIX Error Log */

    /*
     * Set up timeout period to wait for service response.
     */
    memset(&timeout, 0, sizeof(timeout));
    timeout.tv_sec = TIMEOUT_SECONDS;
    /*
     * Set up a select to wait for input from the socket connection to the
     * service.
     */
    read_set_addr = &read_set
    FD_ZERO(&read_set);
    FD_SET(sckt, &read_set);
    /*
     * Check/wait for input from socket. This code is only checking for
     * input, not output or exceptions. Make a maximum of 5 retries.
     */
    for (i = 0 ; i < 5 ; ) {
        linepos = __LINE__;
        rc = select((sckt + 1), read_set_addr, (fd_set *) NULL,
                   (fd_set *) NULL, &timeout);
        local_errno = errno;

```

```

/* Check for timeouts */
if (0 == rc) {
    return(1);
}
/*
 * Check for failures in the select call. Anything but an
 * EAGAIN implies an error in the coding logic of this section,
 * and is treated as an internal failure which needs to be
 * logged. To help understand what might have gone wrong, the
 * log report contains the parameters to the select routine
 * and the error code returned from the call.
 */
if (-1 == rc) {
    if (EAGAIN == local_errno) {
        i++;
        pause_client();
        continue;
    }
    CLEAR_DETAIL_BUF;
    p = (void *) details;
    strcpy((char *) p, "detect_server_reply");
    p += sizeof(char) * 20;
    strcpy((char *) p, "select");
    p += sizeof(char) * 8;
    memcpy(p, &local_errno, sizeof(int));
    p += sizeof(int);
    memcpy(p, &sckt, sizeof(int));
    p += sizeof(int);
    memcpy(p, &read_set_addr, sizeof(int));
    p += sizeof(int);
    memcpy(p, &read_set, sizeof(int));
    p += sizeof(int);
    rc = fc_log_error(returned_fid, NULL, RESOURCE_NAME,
                     LPP_NAME, __FILE__, VERSION_ID,
                     linepos, (void *) NULL, FFDC_ERROR,
                     ERRID_FCLNT_INTERN5_ER, details,
                     (p - (void *) details), NULL,
                     "ffdcclnt failed, select error");

    return(99);
}
/* Check for anything to read on our socket */
if (FD_ISSET(sckt, &read_set)) {
    break;
}
}
if (5 <= i) {
    /* Unable to determine if service replied */
    return(2);
}
/*
 * Data is available for reading on the socket
 */
return(0);
}

/*****
 * Function Name:  recv_server_reply
 * Description:    Reads the response structure from the ffdcserver
 *                application from the socket connection, plus any other
 *                data from the socket connection. If the ffdcserver
 *                application responded successfully, additional response
 *                data will be in the socket, preceded by a length field.
 *                If not, only the response structure will be present,
 *                which may contain an FFDC Failure Identifier if ffdcserver
 *                encountered a failure of its own.
 * Usage:         int = recv_server_reply(int sckt, server_resp_t *resp,
 *                char **sdata, int *sdata_size,
 *****/

```

```

*                                     fc_eid_t *returned_fid)
* Parameters:      sckt      Previously established socket
*                 connection to the ffdcserv application
*                 resp      Pointer to a previously allocated
*                 structure to contain ffdcserv's response
*                 sdata     Address of a data location where any
*                 additional ffdcserv response data should
*                 be stored
*                 sdata_size If non-zero, indicates the length of the
*                 "sdata" information
*                 returned_fid Location where this routine would
*                 return an FFDC Failure Identifier, if
*                 an unrecoverable failure occurred. Note
*                 that this is *not* the FFDC Failure
*                 Identifier from ffdcserv.
* Return Codes:   0      Response successfully received
*                 1      Unable to receive response - memory allocation
*                 was not successful
*                 2      Server response appears invalid - unsupported
*                 version code detected in response
*                 3      Server response appears invalid - unsupported
*                 reply code detected in response
*                 5      Server denied service, or was unable to provide
*                 the service.
*                 99     Internal failure, indicating likely code bug
* Notes:          detect_server_reply should be used prior to this routine
*                 to determine if anything is available on the socket to
*                 be read. Caller is responsible for freeing the memory
*                 allocated to "sdata" upon successful completion of this
*                 routine.
*****/
int
recv_server_reply(int sckt, server_resp_t *resp, char **sdata,
                 int *sdata_size, fc_eid_t *returned_fid)
{
    int      rc;          /* Function call return code */
    int      i;          /* Loop ctr, for reporting only */
    int      linepos;    /* Approx. position of failure */
    int      local_errno; /* So errno won't get lost... */
    size_t   expect;     /* Input expected from service */
    size_t   remain;     /* Input left to read */
    size_t   bytes;      /* Input read this iteration */
    server_resp_t response; /* Reply from service program */
    char     *inbuf;     /* Where to read input to */
    void     *p, *q;     /* Temporary pointers */
    char     details[100]; /* Details on failure to log */

    /*
     * Set up memory areas for our use.
     */
    expect = remain = sizeof(server_resp_t);
    inbuf = malloc(expect);
    if ((char *) NULL == inbuf) {
        return(1);
    }
    q = (void *) inbuf;
    memset(resp, 0, expect);
    /*
     * Read information off the socket.
     */
    rc = socket_read(sckt, inbuf, expect, returned_fid);
    if (0 != rc) {
        free(inbuf);
        return(rc);
    }
    memcpy(resp, (void *) inbuf, sizeof(server_resp_t));
    free(inbuf);

```

```

/*
 * Verify that the response that was received was in the format that
 * this client can expect.
 */
if (VERSION_1 != resp->version && VERSION_2 != resp->version &&
    VERSION_3 != resp->version)
{
    return(2);
}
if (FCA_SUCCESS != resp->reply && FCA_DENY != resp->reply &&
    FCA_FAILED != resp->reply)
{
    return(3);
}
/*
 * Check if the server denied service, or if the server failed in
 * providing the service. In the latter case, an FFDC Failure ID
 * should have been included in the response. In both cases, there
 * is nothing more to read off the socket.
 */
if (FCA_SUCCESS != resp->reply) {
    return(5);
}
/*
 * Read remaining reply off of the socket
 */
rc = socket_read(sckt, &expect, sizeof(size_t), returned_fid);
if (0 != rc) {
    return(rc);
}
(*sdata_size) = expect;
if (0 == expect) {
    return(0);
}
inbuf = malloc(expect);
if ((char *) NULL == inbuf) {
    return(1);
}
memset(inbuf, 0, expect);
rc = socket_read(sckt, inbuf, expect, returned_fid);
if (0 != rc) {
    free(inbuf);
    return(rc);
}
/*
 * Provide data back to the caller
 */
(*sdata) = inbuf;
return(0);
}

/*****
 * Function Name: connect_trace
 * Description:    Keeps track of the attempts to contact an ffdcserver
 *                application provider. If a node that is supposed to
 *                provide the ffdcserver service cannot be contacted or
 *                refuses the contact request, information on these
 *                failures is written to a file for diagnostic
 *                purposes. This file can later be referenced by an
 *                AIX Error Log entry.
 * Usage:         int = connect_trace(char **trace_file, char *server,
 *                int server_rc)
 * Parameters:    trace_file    If this references a NULL string, the
 *                routine creates a new trace file and
 *                places the name in this parameter. If
 *                a file name is referenced, trace data
 *                is recorded to this file.
 *****/

```

```

*          server      Network name of the server with which
*                      ffdclnt experienced the problem
*          server_rc Error code from connect_to_service()
* Return Codes:  0    Information successfully recorded
*               1    Information not recorded - memory allocation
*                   failure
*               2    Information not recorded - could not create
*                   trace file name
*               3    Information not recorded - could not open the
*                   trace file in "append" mode
* Notes:         Caller is responsible for freeing the memory used to
*               store the "trace_file" parameter. Also note that this
*               routine does not record information about failures
*               caused by this application to the trace file - those
*               failures are separately logged.
*****/
int
connect_trace(char **trace_file, char *server, int server_rc)
{
    int      rc;          /* Function call return code */
    char     *file_name; /* Used to store file name */
    FILE     *fptr;      /* Trace file descriptor */

    /*
     * Allocate file name and create trace file if necessary. If the
     * name cannot be allocated or the file cannot be created, the
     * routine will return an error but *not* log anything to persistent
     * storage. This is because the daemon doesn't "need" the trace file
     * to provide its function, so the code decides that this is a "failure"
     * it can tolerate.
     */
    if ((char *) NULL == (*trace_file)) {
        file_name = NULL;
        file_name = malloc(sizeof(char) * L_tmpnam);
        if (NULL == file_name) {
            return(1);
        }
        memset(file_name, 0, sizeof(char) * L_tmpnam);
        tmpnam(file_name);
        if (NULL == (*file_name)) {
            return(2);
        }
        (*trace_file) = file_name;
    }
    fptr = (FILE *) NULL;
    fptr = fopen((*trace_file), "a");
    if ((FILE *) NULL == fptr) {
        return(3);
    }
    /*
     * Trace file has been allocated and/or opened. Record information
     * about the server that could not or would not be contacted, and
     * also provide the failure information.
     */
    fprintf(fptr, "Failure to contact ffdcserv server %s: ", server);
    switch (server_rc) {
        case 1:
            fprintf(fptr, "Server name is a NULL value\n");
            break;
        case 2:
            fprintf(fptr, "Cannot find the server\n");
            break;
        case 3:
            fprintf(fptr, "Cannot find server's network address\n");
            break;
        case 4:
            fprintf(fptr, "Cannot obtain host information for server\n");

```

```

        break;
    case 8:
        fprintf(fp, "No network path to server, or cannot obtain server's address\n");
        break;
    case 9:
        fprintf(fp, "Connection timed out\n");
        break;
    case 10:
        fprintf(fp, "Server refused the network connection\n");
        break;
    default:
        break;
}
fclose(fp);
return(0);
}

```

```

/*****
 * -- MAIN ROUTINE -- MAIN ROUTINE -- MAIN ROUTINE -- MAIN ROUTINE --
 * Description:      Establishes a socket connection to an ffdcserver
 *                  application server and makes a request of that service.
 *                  Examines the response from that service.
 * Usage:           int
 * Parameters:
 * Exit Status:     0    application executed successfully
 *                  1    application executed successfully, ffdcserver
 *                  application denied the request made by this
 *                  application
 *                  2    application encountered a failure
 * Notes:           This application makes itself a daemon process. It is
 *                  not designed to execute under System Resource Controller
 *                  control.
 *****/

```

```

int
main(int argc, char **argv)
{
    int      rc;          /* Function call return code */
    int      i;          /* Loop counter */
    int      linepos;    /* Approx. location of failure */
    int      length;     /* Length of ffdcserver response */
    int      outfile;    /* Where ffdcclnt places output */
    size_t   resp_size;  /* Used in error reporting */
    uid_t    proc_owner; /* Owner of the current process */
    void     *p;         /* Temporary pointer */
    fc_eid_t fid;        /* FFDC Failure Identifier */
    fc_eid_t assoc_fid;  /* Another FFDC Failure ID */
    char     *trace_file; /* Traces connect attempts */
    char     *server_used; /* ffdcserver Server in use */
    char     *server_data; /* Response data from ffdcserver */
    char     servers[5][80]; /* Names of ffdcserver providers */
    char     details[100]; /* Details to AIX Error Log */
    server_resp_t resp;  /* ffdcserver's response to requ. */
    extern int sckt;     /* Connection to ffdcserver node */

    /*
     * Initialize the daemon
     */
    proc_owner = getuid();
    outfile = -1;
    fc_eid_init(fid);
    linepos = __LINE__;
    rc = init_client(&fid);
    switch (rc) {
        case 0:
            /* Successful */
            rc = fc_log_error(&fid, NULL, RESOURCE_NAME, LPP_NAME,
                __FILE__, VERSION_ID, linepos, (void *) NULL,

```

```

        FFDC_STATE, ERRID_FCLNT_ACTIVE_ST, &proc_owner,
        sizeof(proc_owner), NULL,
        "ffdcclnt active");
    break;
case 1:
    /*
     * Usage error. This is reported to persistent
     * storage, since there may be no other way to inform
     * anyone of the failure. However, notice that the
     * failure is recorded as a "debug" entry instead of
     * a permanent failure or an emergency. This is not
     * a permanent failure, since the problem will be
     * rectified once the daemon is used correctly.
     */
    rc = fc_log_error(&fid, NULL, RESOURCE_NAME, LPP_NAME,
        __FILE__, VERSION_ID, linepos, (void *) NULL,
        FFDC_DEBUG, ERRID_FCLNT_SRC_DE, NULL, 0, NULL,
        "ffdcclnt exiting - started from SRC");
    WRITE_CLIENT_RESULTS(outfile, FCA_FAILED, fid, 0, NULL);
    return(2);
case 2:
    /* Initialization error - record already logged */
    WRITE_CLIENT_RESULTS(outfile, FCA_FAILED, fid, 0, NULL);
    return(2);
case 99:
    /* Internal error - record already logged */
    WRITE_CLIENT_RESULTS(outfile, FCA_FAILED, fid, 0, NULL);
    return(2);
}
linepos = __LINE__;
rc = open_output_file(&outfile, &fid);
switch (rc) {
    case 0:
        /* Successful */
        break;
    case 99:
        /* Internal error - record already logged */
        WRITE_CLIENT_RESULTS(outfile, FCA_FAILED, fid, 0, NULL);
        return(2);
    default:
        /*
         * Record this to persistent storage as a "permanent"
         * failure, since the failure will persist until the
         * resource is made available.
         */
        CLEAR_DETAIL_BUF;
        p = (void *) details;
        memcpy(p, &proc_owner, sizeof(int));
        p += sizeof(int);
        strcpy((char *) p, "open_output_file");
        p += sizeof(char) * 20;
        memcpy(p, &rc, sizeof(int));
        p += sizeof(int);
        rc = fc_log_error(&fid, NULL, RESOURCE_NAME, LPP_NAME,
            __FILE__, VERSION_ID, linepos, (void *) NULL,
            FFDC_ERROR, ERRID_FCLNT_OUTPUT_ER, details,
            (p - (void *) details), NULL,
            "ffdcclnt failure - cannot open output file");
        WRITE_CLIENT_RESULTS(outfile, FCA_FAILED, fid, 0, NULL);
        return(2);
}
memset(servers, 0, (sizeof(char) * 5 * 80));
memset(&resp, 0, sizeof(resp));
trace_file = NULL;
server_data = NULL;
fc_eid_init(assoc_fid);
/*

```

```

* Get list of servers that provide the ffdcerv service. This list
* is expected to be contained in the ffdcclnt configuration file,
* which is expected to exist by the time ffdcclnt is configured to
* be executable.
*/
linepos = __LINE__;
rc = get_server_names(servers, &fid);
switch (rc) {
    case 0:
        /* Successful */
        break;
    case 99:
        /* Internal failure - record already logged */
        WRITE_CLIENT_RESULTS(outfile, FCA_FAILED, fid, 0, NULL);
        return(2);
    default:
        /*
        * Cannot read configuration information. This failure
        * can have a number of causes, which the AIX Error
        * Log template reflects. The template is structured
        * so that the more likely causes are listed first.
        * The user ID of the process owner is provided to
        * assist the permission checking effort.
        */
        CLEAR_DETAIL_BUF;
        p = (char *) details;
        memcpy(p, &proc_owner, sizeof(uid_t));
        p += sizeof(uid_t);
        strcpy((char *) p, "get_server_names");
        p += sizeof(char) * 20;
        memcpy(p, &rc, sizeof(int));
        p += sizeof(int);
        fc_log_error(&fid, NULL, RESOURCE_NAME, LPP_NAME,
            __FILE__, VERSION_ID, linepos, (void *) NULL,
            FFDC_ERROR, ERRID_FCLNT_CONFIG_ER,
            (void *) details, (p - (void *) details), NULL,
            "ffdcclnt exiting - configuration error");
        WRITE_CLIENT_RESULTS(outfile, FCA_FAILED, fid, 0, NULL);
        return(2);
}
/*
* Attempt to get a connection to one of the known servers.
*/
for (i = 0 ; i < 5 ; i++) {
    if (0 == strcmp(servers[i], "")) {
        break;
    }
    linepos = __LINE__;
    rc = connect_to_service(servers[i], &sckt, &fid);
    switch (rc) {
        case 0: server_used = servers[i];
            break;
        case 1: /* Intentional fall-through */
        case 2: /* Intentional fall-through */
        case 3: /* Intentional fall-through */
        case 4: /* Intentional fall-through */
        case 8: /* Intentional fall-through */
        case 9: /* Intentional fall-through */
        case 10: connect_trace(&trace_file, servers[i],
            rc);
            break;
        case 5: break;
        case 6: rc = fc_log_error(&fid, NULL,
            RESOURCE_NAME, LPP_NAME,
            __FILE__, VERSION_ID, linepos,
            (void *) NULL, FFDC_DEBUG,
            ERRID_FCLNT_OPENF_DE, NULL, 0,

```

```

        NULL,
        "ffdcclnt failed - too many opened files to create socket");
WRITE_CLIENT_RESULTS(outfile,
    FCA_FAILED, fid, 0, NULL);
return(2);
case 7: rc = fc_log_error(&fid, NULL,
    RESOURCE_NAME, LPP_NAME,
    __FILE__, VERSION_ID, linepos,
    (void *) NULL, FFDC_DEBUG,
    ERRID_FCLNT_NOBUFS_DE, NULL, 0,
    NULL,
    "ffdcclnt failed - memory buffer shortage");
case 99: WRITE_CLIENT_RESULTS(outfile,
    FCA_FAILED, fid, 0, NULL);
return(2);
}
}
if (-1 == sckt) {
    /*
     * All known servers tried, no successful connections
     * established.
     */
    rc = fc_log_error(&fid, NULL, RESOURCE_NAME, LPP_NAME, __FILE__,
        VERSION_ID, linepos, (void *) NULL,
        FFDC_ERROR, ERRID_FCLNT_NOSVC_ER, NULL, 0,
        trace_file,
        "ffdcclnt failed - No ffdcserver servers");
    if (FC_COPY_FAILED != rc) {
        unlink(trace_file);
    }
    free(trace_file);
    WRITE_CLIENT_RESULTS(outfile, FCA_FAILED, fid, 0, NULL);
    return(2);
}
if (0 < i) {
    /*
     * At least one retry had to be made to get a connection.
     */
    rc = fc_log_error(&fid, NULL, RESOURCE_NAME, LPP_NAME, __FILE__,
        VERSION_ID, linepos, (void *) NULL,
        FFDC_RECOV, ERRID_FCLNT_SRETRY_RE, NULL, 0,
        trace_file,
        "ffdcclnt failed - No ffdcserver servers");
    if (FC_COPY_FAILED != rc) {
        unlink(trace_file);
    }
    free(trace_file);
}
/*
 * The application decides that once it is connected to a server, it
 * will stick with that server. In other words, if the connection
 * should drop or the server should experience a failure, this
 * application will not attempt to connect to another server and try
 * the request again. It cold, and future versions might consider
 * doing just that, but this version will not. Now send the request
 * to the server.
 */
linepos = __LINE__;
rc = send_service_request(sckt, &fid);
switch (rc) {
    case 0: break;
    case 1: rc = fc_log_error(&fid, NULL, RESOURCE_NAME,
        LPP_NAME, __FILE__, VERSION_ID, linepos,
        (void *) NULL, FFDC_DEBUG,
        ERRID_FCLNT_SDROP_DE, server_used,
        strlen(server_used), NULL,
        "ffdcclnt failed - ffdcserver dropped connection");
}

```

```

        close(sckt);
        WRITE_CLIENT_RESULTS(outfile, FCA_FAILED, fid,
            0, NULL);
        return(2);
    case 99: close(sckt);
        WRITE_CLIENT_RESULTS(outfile, FCA_FAILED, fid,
            0, NULL);
        return(2);
}
/*
 * Wait for response from the server
 */
linepos = __LINE__;
rc = detect_server_reply(sckt, &fid);
switch (rc) {
    case 0: break;
    case 1: rc = fc_log_error(&fid, NULL, RESOURCE_NAME,
        LPP_NAME, __FILE__, VERSION_ID, linepos,
        (void *) NULL, FFDC_DEBUG,
        ERRID_FCLNT_STIMEDO_DE, server_used,
        strlen(server_used), NULL,
        "ffdcclnt failed - ffdcscrv timed out");
        close(sckt);
        WRITE_CLIENT_RESULTS(outfile, FCA_FAILED, fid,
            0, NULL);
        return(2);
    case 2: rc = fc_log_error(&fid, NULL, RESOURCE_NAME,
        LPP_NAME, __FILE__, VERSION_ID, linepos,
        (void *) NULL, FFDC_DEBUG,
        ERRID_FCLNT_SRESP_DE, NULL, 0, NULL,
        "ffdcclnt failed - select failed");
        close(sckt);
        WRITE_CLIENT_RESULTS(outfile, FCA_FAILED, fid,
            0, NULL);
        return(2);
    case 99: close(sckt);
        WRITE_CLIENT_RESULTS(outfile, FCA_FAILED, fid,
            0, NULL);
        return(2);
}
/*
 * Read response from service, and determine if the response is a
 * "good" response.  If the server responded with a failure, record
 * a failure in this application as well that cites the server's
 * failure directly (through the use of the server's FFDC Failure
 * Identifier).
 */
length = 0;
resp_size = sizeof(server_resp_t);
linepos = __LINE__;
rc = rcv_server_reply(sckt, &resp, &server_data, &length, &fid);
switch (rc) {
    case 0: break;
    case 1: if (0 == length) {
        rc = fc_log_error(&fid, NULL,
            RESOURCE_NAME, LPP_NAME,
            __FILE__, VERSION_ID, linepos,
            (void *) NULL, FFDC_DEBUG,
            ERRID_FCLNT_NOMEM_DE,
            &resp_size, sizeof(int), NULL,
            "ffdcclnt failed - memory allocation failure");
    }
    else {
        rc = fc_log_error(&fid, NULL,
            RESOURCE_NAME, LPP_NAME,
            __FILE__, VERSION_ID, linepos,
            (void *) NULL, FFDC_DEBUG,

```

```

        ERRID_FCLNT_NOMEM_DE,
        &length, sizeof(int), NULL,
        "ffdcclnt failed - memory allocation failure");
    }
    close(sckt);
    WRITE_CLIENT_RESULTS(outfile, FCA_FAILED, fid,
        0, NULL);
    return(2);
case 2: /* Intentional fall-through */
case 3: CLEAR_DETAIL_BUF;
    p = (void *) details;
    strcpy((char *) p, server_used);
    p += sizeof(char) * 80;
    memcpy(p, &resp, (sizeof(int) * 2));
    rc = fc_log_error(&fid, NULL, RESOURCE_NAME,
        LPP_NAME, __FILE__, VERSION_ID, linepos,
        (void *) NULL, FFDC_ERROR,
        ERRID_FCLNT_SRESP_DE, details,
        (p - (void *) details), NULL,
        "ffdcclnt failed - sffdcserv response invalid");
    close(sckt);
    WRITE_CLIENT_RESULTS(outfile, FCA_FAILED, fid,
        0, NULL);
    return(2);
case 5: /* Figure out what happened in next code */
    break;
case 99: close(sckt);
    WRITE_CLIENT_RESULTS(outfile, FCA_FAILED, fid,
        0, NULL);
    return(2);
}
close(sckt);
switch (resp.reply) {
    case FCA_SUCCESS: /*
        * Service was provided - continue normal
        * execution.
        */
        break;
    case FCA_DENY: /*
        * Service was denied by ffdcerv. The
        * application programmer would need to know
        * why ffdcerv might deny a request, and
        * decide whether the request should be
        * retried on another server or if the
        * application should proceed without the
        * information. This code assumed that a
        * request is "denied" for reasons other than
        * a failure experienced by ffdcerv or
        * an invalid request made of ffdcerv.
        */
        WRITE_CLIENT_RESULTS(outfile, FCA_DENY, fid, 0,
            NULL);
        return(1);
    case FCA_FAILED: /*
        * ffdcerv application experienced a failure,
        * which in turn causes a failure in this
        * application as well. This is treated as
        * a failure and recorded to persistent
        * storage, and the report cites the FFDC
        * Failure Identifier returned by ffdcerv.
        * This establishes a link between the failures,
        * and makes it clear that ffdcclnt's failure
        * is a direct result of ffdcerv's failure.
        * Once the ffdcerv failure is resolved,
        * ffdcclnt should not experience this failure.
        */
        rc = fc_log_error(&fid, resp.ffdcid,

```

```

        RESOURCE_NAME, LPP_NAME, __FILE__,
        VERSION_ID, linepos, (void *) NULL,
        FFDC_ERROR, ERRID_FCLNT_SVCERR_ER,
        server_used, strlen(server_used), NULL,
        "ffdcclnt failure - ffdcserv error");
WRITE_CLIENT_RESULTS(outfile, FCA_FAILED, fid,
0, NULL);
return(2);
}
/*
 * Server successfully responded to the request. Do whatever needs
 * to be done with the data.
 */
WRITE_CLIENT_RESULTS(outfile, FCA_SUCCESS, fid, length, server_data);
/*
 * Data processing completed - normal termination of ffdccclnt
 */
if (NULL != server_data) {
    free(server_data);
}
rc = fc_log_error(&fid, NULL, RESOURCE_NAME, LPP_NAME, __FILE__,
VERSION_ID, linepos, (void *) NULL, FFDC_STATE,
ERRID_FCLNT_EXIT_ST, &proc_owner, sizeof(proc_owner),
NULL, "ffdcclnt exiting normally");
return(0);
}

```

The ffdccclnt.err.E Error Log Template File

```

#include <ffdcclnt.client.eth>
*!ffdcexpl.cat

+ FCLNT_ACTIVE_ST:
  Comment = "FFDC Client Application - Client Is Active"
  Class = S
  Log = True
  Report = True
  Alert = False
  Err_Type = INFO
  Err_Desc = {client, EMSG502, client_MSG502_DEFAULT}
  Prob_Causes = E840
  Fail_Causes = E902
  Fail_Actions = E810
  Detail_Data = 46, 00A2, ALPHA
  Detail_Data = 42, EB2B, ALPHA
  Detail_Data = 42, 0030, ALPHA
  Detail_Data = 4, {client, EMSG655, client_MSG655_DEFAULT}, DEC

+ FCLNT_EXIT_ST:
  Comment = "FFDC Client Application - Client Is Exiting"
  Class = S
  Log = True
  Report = True
  Alert = False
  Err_Type = INFO
  Err_Desc = {client, EMSG515, client_MSG515_DEFAULT}
  Prob_Causes = E840
  Fail_Causes = E902
  Fail_Actions = E810
  Detail_Data = 46, 00A2, ALPHA
  Detail_Data = 42, EB2B, ALPHA
  Detail_Data = 42, 0030, ALPHA
  Detail_Data = 4, {client, EMSG655, client_MSG655_DEFAULT}, DEC

+ FCLNT_TERMSIG_ST:
  Comment = "FFDC Client Application - Terminating By Signal"
  Class = S

```

```

Log = True
Report = True
Alert = False
Err_Type = INFO
Err_Desc = {client, MSG504, client_MSG504_DEFAULT}
Prob_Causes = {client, MSG531, client_MSG531_DEFAULT}, \
               {client, MSG532, client_MSG532_DEFAULT}
Fail_Causes = E902
Fail_Actions = E810
Detail_Data = 46, 00A2, ALPHA
Detail_Data = 42, EB2B, ALPHA
Detail_Data = 42, 0030, ALPHA
Detail_Data = 4, {client, MSG676, client_MSG676_DEFAULT}, DEC

```

```

+ FCLNT_SRC_DE:
Comment = "FFDC Client Application Not Started - Started Incorrectly"
Class = S
Log = True
Report = True
Alert = False
Err_Type = UNKN
Err_Desc = {client, MSG506, client_MSG506_DEFAULT}
Prob_Causes = {client, MSG534, client_MSG534_DEFAULT}, \
               {client, MSG535, client_MSG535_DEFAULT}
User_Causes = {client, MSG557, client_MSG557_DEFAULT}
User_Actions = {client, MSG609, client_MSG609_DEFAULT}, \
                {client, MSG610, client_MSG610_DEFAULT}, \
                {client, MSG611, client_MSG611_DEFAULT}, \
                {client, MSG612, client_MSG612_DEFAULT}
Detail_Data = 46, 00A2, ALPHA
Detail_Data = 42, EB2B, ALPHA
Detail_Data = 42, 0030, ALPHA

```

```

+ FCLNT_OUTPUT_ER:
Comment = "FFDC Client Application - File Creation Error"
Class = S
Log = True
Report = True
Alert = False
Err_Type = PERM
Err_Desc = {client, MSG506, client_MSG506_DEFAULT}
Prob_Causes = {client, MSG547, client_MSG547_DEFAULT}
Fail_Causes = {client, MSG583, client_MSG583_DEFAULT}, \
               {client, MSG561, client_MSG561_DEFAULT}, \
               {client, MSG582, client_MSG582_DEFAULT}, \
               {client, MSG584, client_MSG584_DEFAULT}
Fail_Actions = {client, MSG636, client_MSG636_DEFAULT}, \
                {client, MSG635, client_MSG635_DEFAULT}, \
                {client, MSG637, client_MSG637_DEFAULT}, \
                {client, MSG632, client_MSG632_DEFAULT}
User_Causes = {client, MSG585, client_MSG585_DEFAULT}, \
               {client, MSG562, client_MSG562_DEFAULT}
User_Actions = {client, MSG638, client_MSG638_DEFAULT}, \
                {client, MSG616, client_MSG616_DEFAULT}
Detail_Data = 46, 00A2, ALPHA
Detail_Data = 42, EB2B, ALPHA
Detail_Data = 42, 0030, ALPHA
Detail_Data = 4, {client, MSG683, client_MSG683_DEFAULT}, DEC
Detail_Data = 20, {client, MSG658, client_MSG658_DEFAULT}, ALPHA
Detail_Data = 4, {client, MSG684, client_MSG684_DEFAULT}, DEC

```

```

+ FCLNT_CONFIG_ER:
Comment = "FFDC Client Application Failure - Configuration File Failure"
Class = S
Log = True
Report = True
Alert = False

```

```

Err_Type = PERM
Err_Desc = {client, EMSG507, client_MSG507_DEFAULT}
Prob_Causes = {client, EMSG536, client_MSG536_DEFAULT}, \
               {client, EMSG537, client_MSG537_DEFAULT}
Inst_Causes = {client, EMSG558, client_MSG558_DEFAULT}
Inst_Actions = {client, EMSG613, client_MSG613_DEFAULT}
Fail_Causes = {client, EMSG559, client_MSG559_DEFAULT}, \
              {client, EMSG560, client_MSG560_DEFAULT}, \
              {client, EMSG561, client_MSG561_DEFAULT}
Fail_Actions = {client, EMSG613, client_MSG613_DEFAULT}, \
               {client, EMSG614, client_MSG614_DEFAULT}, \
               {client, EMSG615, client_MSG615_DEFAULT}
User_Causes = {client, EMSG562, client_MSG562_DEFAULT}
User_Actions = {client, EMSG616, client_MSG616_DEFAULT}
Detail_Data = 46, 00A2, ALPHA
Detail_Data = 42, EB2B, ALPHA
Detail_Data = 42, 0030, ALPHA
Detail_Data = 4, {client, EMSG683, client_MSG683_DEFAULT}, DEC
Detail_Data = 20, {client, EMSG657, client_MSG657_DEFAULT}, ALPHA
Detail_Data = 4, {client, EMSG684, client_MSG684_DEFAULT}, DEC

```

+ FCLNT_DAEINIT_ER:

```

Comment = "FFDC Client Failure - Daemon Initialization Resource Failed"
Class = S
Log = True
Report = True
Alert = False
Err_Type = PERM
Err_Desc = {client, EMSG506, client_MSG506_DEFAULT}
Prob_Causes = {client, EMSG533, client_MSG533_DEFAULT}
Fail_Causes = {client, EMSG554, client_MSG554_DEFAULT}, \
              {client, EMSG555, client_MSG555_DEFAULT}
Fail_Actions = {client, EMSG606, client_MSG606_DEFAULT}, \
               {client, EMSG607, client_MSG607_DEFAULT}, \
               {client, EMSG608, client_MSG608_DEFAULT}
Detail_Data = 46, 00A2, ALPHA
Detail_Data = 42, EB2B, ALPHA
Detail_Data = 42, 0030, ALPHA
Detail_Data = 4, {client, EMSG677, client_MSG677_DEFAULT}, DEC
Detail_Data = 20, {client, EMSG678, client_MSG678_DEFAULT}, ALPHA
Detail_Data = 76, {client, EMSG679, client_MSG679_DEFAULT}, ALPHA

```

+ FCLNT_SVCNOTF_ER:

```

Comment = "FFDC Client Failure - Cannot Locate Service Information"
Class = S
Log = True
Report = True
Alert = False
Err_Type = PERM
Err_Desc = {client, EMSG501, client_MSG501_DEFAULT}
Prob_Causes = {client, EMSG521, client_MSG521_DEFAULT}, \
               {client, EMSG522, client_MSG522_DEFAULT}, \
               {client, EMSG523, client_MSG523_DEFAULT}
Inst_Causes = {client, EMSG551, client_MSG551_DEFAULT}
Inst_Actions = {client, EMSG601, client_MSG601_DEFAULT}, \
               {client, EMSG602, client_MSG602_DEFAULT}, \
               {client, EMSG603, client_MSG603_DEFAULT}
User_Causes = {client, EMSG524, client_MSG524_DEFAULT}
User_Actions = {client, EMSG601, client_MSG601_DEFAULT}, \
               {client, EMSG602, client_MSG602_DEFAULT}, \
               {client, EMSG603, client_MSG603_DEFAULT}
Detail_Data = 46, 00A2, ALPHA
Detail_Data = 42, EB2B, ALPHA
Detail_Data = 42, 0030, ALPHA
Detail_Data = 16, {client, EMSG651, client_MSG651_DEFAULT}, ALPHA
Detail_Data = 16, {client, EMSG652, client_MSG652_DEFAULT}, ALPHA
Detail_Data = 4, {client, EMSG653, client_MSG653_DEFAULT}, ALPHA

```

```

+ FCLNT_OPENF_DE:
  Comment = "FFDC Client Application - Too Many Opened Files"
  Class = S
  Log = True
  Report = True
  Alert = False
  Err_Type = UNKN
  Err_Desc = {client, EMSG506, client_MSG506_DEFAULT}
  Prob_Causes = {client, EMSG525, client_MSG525_DEFAULT}
  User_Causes = {client, EMSG562, client_MSG562_DEFAULT}
  User_Actions = {client, EMSG616, client_MSG616_DEFAULT}
  Detail_Data = 46, 00A2, ALPHA
  Detail_Data = 42, EB2B, ALPHA
  Detail_Data = 42, 0030, ALPHA

+ FCLNT_NOBUFS_DE:
  Comment = "FFDC Client Application - No Buffers Socket Failure"
  Class = S
  Log = True
  Report = True
  Alert = False
  Err_Type = UNKN
  Err_Desc = {client, EMSG508, client_MSG508_DEFAULT}
  Prob_Causes = {client, EMSG538, client_MSG538_DEFAULT}
  Fail_Causes = {client, EMSG563, client_MSG563_DEFAULT}
  Fail_Actions = {client, EMSG617, client_MSG617_DEFAULT}, \
                 {client, EMSG618, client_MSG618_DEFAULT}, \
                 {client, EMSG619, client_MSG619_DEFAULT}
  Detail_Data = 46, 00A2, ALPHA
  Detail_Data = 42, EB2B, ALPHA
  Detail_Data = 42, 0030, ALPHA

+ FCLNT_NOMEM_DE:
  Comment = "FFDC Client Application - Memory Allocation Failure"
  Class = S
  Log = True
  Report = True
  Alert = False
  Err_Type = UNKN
  Err_Desc = {client, EMSG508, client_MSG508_DEFAULT}
  Prob_Causes = {client, EMSG544, client_MSG544_DEFAULT}
  Fail_Causes = {client, EMSG574, client_MSG574_DEFAULT}, \
               {client, EMSG563, client_MSG563_DEFAULT}
  Fail_Actions = {client, EMSG618, client_MSG618_DEFAULT}, \
               {client, EMSG619, client_MSG619_DEFAULT}, \
               {client, EMSG630, client_MSG630_DEFAULT}
  User_Causes = {client, EMSG575, client_MSG575_DEFAULT}
  User_Actions = {client, EMSG628, client_MSG628_DEFAULT}, \
               {client, EMSG629, client_MSG629_DEFAULT}
  Detail_Data = 46, 00A2, ALPHA
  Detail_Data = 42, EB2B, ALPHA
  Detail_Data = 42, 0030, ALPHA
  Detail_Data = 4, {client, EMSG687, client_MSG687_DEFAULT}, DEC

+ FCLNT_NOSVC_ER:
  Comment = "FFDC Client Application - No servers Available"
  Class = S
  Log = True
  Report = True
  Alert = False
  Err_Type = PERM
  Err_Desc = {client, EMSG509, client_MSG509_DEFAULT}
  Prob_Causes = {client, EMSG539, client_MSG539_DEFAULT}
  Fail_Causes = {client, EMSG564, client_MSG564_DEFAULT}, \
               {client, EMSG565, client_MSG565_DEFAULT}, \
               {client, EMSG566, client_MSG566_DEFAULT}, \

```

```

        {client, EMSG567, client_MSG567_DEFAULT}
Fail_Actions = {client, EMSG601, client_MSG601_DEFAULT}, \
        {client, EMSG620, client_MSG620_DEFAULT}, \
        {client, EMSG621, client_MSG621_DEFAULT}, \
        {client, EMSG622, client_MSG622_DEFAULT}
Detail_Data = 46, 00A2, ALPHA
Detail_Data = 42, EB2B, ALPHA
Detail_Data = 42, 0030, ALPHA
Detail_Data = 80, {client, EMSG685, client_MSG685_DEFAULT}, ALPHA

+ FCLNT_SRETRY_RE:
Comment = "FFDC Client Application - Connected After Retries"
Class = S
Log = True
Report = True
Alert = False
Err_Type = TEMP
Err_Desc = {client, EMSG540, client_MSG540_DEFAULT}
Prob_Causes = {client, EMSG510, client_MSG510_DEFAULT}
Fail_Causes = {client, EMSG564, client_MSG564_DEFAULT}, \
        {client, EMSG565, client_MSG565_DEFAULT}, \
        {client, EMSG566, client_MSG566_DEFAULT}, \
        {client, EMSG567, client_MSG567_DEFAULT}
Fail_Actions = {client, EMSG623, client_MSG623_DEFAULT}, \
        {client, EMSG622, client_MSG622_DEFAULT}
Detail_Data = 46, 00A2, ALPHA
Detail_Data = 42, EB2B, ALPHA
Detail_Data = 42, 0030, ALPHA
Detail_Data = 80, {client, EMSG685, client_MSG685_DEFAULT}, ALPHA

+ FCLNT_SDROP DE:
Comment = "FFDC Client Application - Connection Dropped By Server"
Class = S
Log = True
Report = True
Alert = False
Err_Type = UNKN
Err_Desc = {client, EMSG511, client_MSG511_DEFAULT}
Prob_Causes = {client, EMSG541, client_MSG541_DEFAULT}
Fail_Causes = {client, EMSG568, client_MSG568_DEFAULT}, \
        {client, EMSG569, client_MSG569_DEFAULT}
Fail_Actions = {client, EMSG624, client_MSG624_DEFAULT}, \
        {client, EMSG625, client_MSG625_DEFAULT}
Inst_Causes = {client, EMSG570, client_MSG570_DEFAULT}
Inst_Actions = {client, EMSG602, client_MSG602_DEFAULT}, \
        {client, EMSG626, client_MSG626_DEFAULT}
Detail_Data = 46, 00A2, ALPHA
Detail_Data = 42, EB2B, ALPHA
Detail_Data = 42, 0030, ALPHA
Detail_Data = 80, {client, EMSG686, client_MSG686_DEFAULT}, ALPHA

+ FCLNT_STIMEDO DE:
Comment = "FFDC Client Application - ffdcserv Timed Out"
Class = S
Log = True
Report = True
Alert = False
Err_Type = UNKN
Err_Desc = {client, EMSG512, client_MSG512_DEFAULT}
Prob_Causes = {client, EMSG542, client_MSG542_DEFAULT}
Fail_Causes = {client, EMSG571, client_MSG571_DEFAULT}, \
        {client, EMSG572, client_MSG572_DEFAULT}
Fail_Actions = {client, EMSG624, client_MSG624_DEFAULT}, \
        {client, EMSG625, client_MSG625_DEFAULT}, \
        {client, EMSG627, client_MSG627_DEFAULT}
Detail_Data = 46, 00A2, ALPHA
Detail_Data = 42, EB2B, ALPHA

```

```

Detail_Data = 42, 0030, ALPHA
Detail_Data = 80, {client, MSG686, client_MSG686_DEFAULT}, ALPHA

+ FCLNT_SRESP_DE:
Comment = "FFDC Client Application - Cannot Detect Server Response"
Class = S
Log = True
Report = True
Alert = False
Err_Type = UNKN
Err_Desc = {client, MSG513, client_MSG513_DEFAULT}
Prob_Causes = {client, MSG543, client_MSG543_DEFAULT}
Fail_Causes = {client, MSG573, client_MSG573_DEFAULT}, \
               {client, MSG563, client_MSG563_DEFAULT}
Fail_Actions = {client, MSG618, client_MSG618_DEFAULT}, \
               {client, MSG619, client_MSG619_DEFAULT}
Detail_Data = 46, 00A2, ALPHA
Detail_Data = 42, EB2B, ALPHA
Detail_Data = 42, 0030, ALPHA

+ FCLNT_SBADRSP_ER:
Comment = "FFDC Client Application - ffdcserv Response Invalid"
Class = S
Log = True
Report = True
Alert = False
Err_Type = PERM
Err_Desc = {client, MSG514, client_MSG514_DEFAULT}
Prob_Causes = {client, MSG545, client_MSG545_DEFAULT}
Fail_Causes = {client, MSG569, client_MSG569_DEFAULT}
Fail_Actions = {client, MSG624, client_MSG624_DEFAULT}, \
               {client, MSG625, client_MSG625_DEFAULT}
Inst_Causes = {client, MSG576, client_MSG576_DEFAULT}, \
               {client, MSG570, client_MSG570_DEFAULT}
Inst_Actions = {client, MSG631, client_MSG631_DEFAULT}, \
               {client, MSG602, client_MSG602_DEFAULT}, \
               {client, MSG626, client_MSG626_DEFAULT}, \
               {client, MSG632, client_MSG632_DEFAULT}
Detail_Data = 46, 00A2, ALPHA
Detail_Data = 42, EB2B, ALPHA
Detail_Data = 42, 0030, ALPHA
Detail_Data = 80, {client, MSG686, client_MSG686_DEFAULT}, ALPHA
Detail_Data = 8, {client, MSG688, client_MSG688_DEFAULT}, HEX

+ FCLNT_SVCERR_ER:
Comment = "FFDC Client Application - ffdcserv Service Failure"
Class = S
Log = True
Report = True
Alert = False
Err_Type = PERM
Err_Desc = {client, MSG516, client_MSG516_DEFAULT}
Prob_Causes = {client, MSG546, client_MSG546_DEFAULT}
Fail_Causes = {client, MSG577, client_MSG577_DEFAULT}
Fail_Actions = {client, MSG633, client_MSG633_DEFAULT}, \
               {client, MSG634, client_MSG634_DEFAULT}
Detail_Data = 46, 00A2, ALPHA
Detail_Data = 42, EB2B, ALPHA
Detail_Data = 42, 0030, ALPHA
Detail_Data = 80, {client, MSG686, client_MSG686_DEFAULT}, ALPHA

+ FCLNT_INTERN1_ER:
Comment = "FFDC Client Application - Internal Socket Creation Error"
Class = S
Log = True
Report = True
Alert = False

```

```

Err_Type = PERM
Err_Desc = {client, EMSG503, client_MSG503_DEFAULT}
Prob_Causes = {client, EMSG525, client_MSG525_DEFAULT}
Fail_Causes = {client, EMSG552, client_MSG552_DEFAULT}
Fail_Actions = {client, EMSG604, client_MSG604_DEFAULT}, \
                {client, EMSG605, client_MSG605_DEFAULT}
Detail_Data = 46, 00A2, ALPHA
Detail_Data = 42, EB2B, ALPHA
Detail_Data = 42, 0030, ALPHA
Detail_Data = 20, {client, EMSG657, client_MSG657_DEFAULT}, ALPHA
Detail_Data = 40, {client, EMSG658, client_MSG658_DEFAULT}, ALPHA
Detail_Data = 4, 8015, DEC

+ FCLNT_INTERN2_ER:
Comment = "FFDC Client Application - Internal Socket Connection Failure"
Class = S
Log = True
Report = True
Alert = False
Err_Type = PERM
Err_Desc = {client, EMSG503, client_MSG503_DEFAULT}
Prob_Causes = {client, EMSG526, client_MSG526_DEFAULT}
Fail_Causes = {client, EMSG578, client_MSG578_DEFAULT}
Fail_Actions = {client, EMSG604, client_MSG604_DEFAULT}, \
                {client, EMSG605, client_MSG605_DEFAULT}
Detail_Data = 46, 00A2, ALPHA
Detail_Data = 42, EB2B, ALPHA
Detail_Data = 42, 0030, ALPHA
Detail_Data = 20, {client, EMSG657, client_MSG657_DEFAULT}, ALPHA
Detail_Data = 8, {client, EMSG658, client_MSG658_DEFAULT}, ALPHA
Detail_Data = 4, 8015, DEC
Detail_Data = 4, {client, EMSG659, client_MSG659_DEFAULT}, DEC
Detail_Data = 4, {client, EMSG660, client_MSG660_DEFAULT}, DEC
Detail_Data = 4, {client, EMSG661, client_MSG661_DEFAULT}, DEC
Detail_Data = 4, {client, EMSG662, client_MSG662_DEFAULT}, DEC
Detail_Data = 4, {client, EMSG663, client_MSG663_DEFAULT}, DEC
Detail_Data = 4, {client, EMSG664, client_MSG664_DEFAULT}, DEC

+ FCLNT_INTERN3_ER:
Comment = "FFDC Client Application - Internal File Access Error"
Class = S
Log = True
Report = True
Alert = False
Err_Type = PERM
Err_Desc = {client, EMSG503, client_MSG503_DEFAULT}
Prob_Causes = {client, EMSG527, client_MSG527_DEFAULT}
Fail_Causes = {client, EMSG553, client_MSG553_DEFAULT}
Fail_Actions = {client, EMSG604, client_MSG604_DEFAULT}, \
                {client, EMSG605, client_MSG605_DEFAULT}
Detail_Data = 46, 00A2, ALPHA
Detail_Data = 42, EB2B, ALPHA
Detail_Data = 42, 0030, ALPHA
Detail_Data = 20, {client, EMSG657, client_MSG657_DEFAULT}, ALPHA
Detail_Data = 8, {client, EMSG658, client_MSG658_DEFAULT}, ALPHA
Detail_Data = 4, 8015, DEC
Detail_Data = 60, {client, EMSG665, client_MSG665_DEFAULT}, ALPHA
Detail_Data = 4, {client, EMSG666, client_MSG666_DEFAULT}, ALPHA

+ FCLNT_INTERN4_ER:
Comment = "FFDC Client Application - Internal Data Transmission Error"
Class = S
Log = True
Report = True
Alert = False
Err_Type = PERM
Err_Desc = {client, EMSG503, client_MSG503_DEFAULT}

```

```

Prob_Causes = {client, EMSG528, client_MSG528_DEFAULT}
Fail_Causes = {client, EMSG554, client_MSG554_DEFAULT}
Fail_Actions = {client, EMSG604, client_MSG604_DEFAULT}, \
               {client, EMSG605, client_MSG605_DEFAULT}
Detail_Data = 46, 00A2, ALPHA
Detail_Data = 42, EB2B, ALPHA
Detail_Data = 42, 0030, ALPHA
Detail_Data = 20, {client, EMSG657, client_MSG657_DEFAULT}, ALPHA
Detail_Data = 8, {client, EMSG658, client_MSG658_DEFAULT}, ALPHA
Detail_Data = 4, 8015, DEC
Detail_Data = 4, {client, EMSG667, client_MSG667_DEFAULT}, DEC
Detail_Data = 4, {client, EMSG668, client_MSG668_DEFAULT}, DEC
Detail_Data = 4, {client, EMSG669, client_MSG669_DEFAULT}, DEC
Detail_Data = 56, {client, EMSG670, client_MSG670_DEFAULT}, HEX

```

+ FCLNT_INTERN5_ER:

```

Comment = "FFDC Client Application - Internal Socket Query Failure"
Class = S
Log = True
Report = True
Alert = False
Err_Type = PERM
Err_Desc = {client, EMSG503, client_MSG503_DEFAULT}
Prob_Causes = {client, EMSG529, client_MSG529_DEFAULT}
Fail_Causes = {client, EMSG579, client_MSG579_DEFAULT}
Fail_Actions = {client, EMSG604, client_MSG604_DEFAULT}, \
               {client, EMSG605, client_MSG605_DEFAULT}
Detail_Data = 46, 00A2, ALPHA
Detail_Data = 42, EB2B, ALPHA
Detail_Data = 42, 0030, ALPHA
Detail_Data = 20, {client, EMSG657, client_MSG657_DEFAULT}, ALPHA
Detail_Data = 8, {client, EMSG658, client_MSG658_DEFAULT}, ALPHA
Detail_Data = 4, 8015, DEC
Detail_Data = 4, {client, EMSG659, client_MSG659_DEFAULT}, DEC
Detail_Data = 4, {client, EMSG671, client_MSG671_DEFAULT}, DEC
Detail_Data = 4, {client, EMSG672, client_MSG672_DEFAULT}, DEC

```

+ FCLNT_INTERN6_ER:

```

Comment = "FFDC Client Application - Internal Socket Read Failure"
Class = S
Log = True
Report = True
Alert = False
Err_Type = PERM
Err_Desc = {client, EMSG503, client_MSG503_DEFAULT}
Prob_Causes = {client, EMSG530, client_MSG530_DEFAULT}
Fail_Causes = {client, EMSG580, client_MSG580_DEFAULT}
Fail_Actions = {client, EMSG604, client_MSG604_DEFAULT}, \
               {client, EMSG605, client_MSG605_DEFAULT}
Detail_Data = 46, 00A2, ALPHA
Detail_Data = 42, EB2B, ALPHA
Detail_Data = 42, 0030, ALPHA
Detail_Data = 20, {client, EMSG657, client_MSG657_DEFAULT}, ALPHA
Detail_Data = 8, {client, EMSG658, client_MSG658_DEFAULT}, ALPHA
Detail_Data = 4, 8015, DEC
Detail_Data = 4, {client, EMSG659, client_MSG659_DEFAULT}, DEC
Detail_Data = 4, {client, EMSG667, client_MSG667_DEFAULT}, DEC
Detail_Data = 4, {client, EMSG673, client_MSG673_DEFAULT}, DEC
Detail_Data = 4, {client, EMSG674, client_MSG674_DEFAULT}, DEC
Detail_Data = 4, {client, EMSG675, client_MSG675_DEFAULT}, DEC

```

+ FCLNT_INTERN7_ER:

```

Comment = "FFDC Client Application - Internal dae_init Usage Failure"
Class = S
Log = True
Report = True
Alert = False

```

```

Err_Type = PERM
Err_Desc = {client, EMSG503, client_MSG503_DEFAULT}
Prob_Causes = {client, EMSG533, client_MSG533_DEFAULT}
Fail_Causes = {client, EMSG581, client_MSG581_DEFAULT}
Fail_Actions = {client, EMSG604, client_MSG604_DEFAULT}, \
                {client, EMSG605, client_MSG605_DEFAULT}
Detail_Data = 46, 00A2, ALPHA
Detail_Data = 42, EB2B, ALPHA
Detail_Data = 42, 0030, ALPHA
Detail_Data = 4, {client, EMSG677, client_MSG677_DEFAULT}, DEC
Detail_Data = 4, {client, EMSG680, client_MSG680_DEFAULT}, DEC
Detail_Data = 4, {client, EMSG681, client_MSG681_DEFAULT}, HEX
Detail_Data = 4, {client, EMSG682, client_MSG682_DEFAULT}, HEX

```

```

+ FCLNT_INTERN8_ER:
Comment = "FFDC Client Application - Internal File Creation Error"
Class = S
Log = True
Report = True
Alert = False
Err_Type = PERM
Err_Desc = {client, EMSG503, client_MSG503_DEFAULT}
Prob_Causes = {client, EMSG547, client_MSG547_DEFAULT}
Fail_Causes = {client, EMSG586, client_MSG586_DEFAULT}
Fail_Actions = {client, EMSG604, client_MSG604_DEFAULT}, \
                {client, EMSG605, client_MSG605_DEFAULT}
Detail_Data = 46, 00A2, ALPHA
Detail_Data = 42, EB2B, ALPHA
Detail_Data = 42, 0030, ALPHA
Detail_Data = 10, {client, EMSG658, client_MSG658_DEFAULT}, ALPHA
Detail_Data = 4, {client, EMSG666, client_MSG666_DEFAULT}, DEC
Detail_Data = 4, 8015, DEC

```

The ffdccInt.msg Message Catalog File

```

$ -----
$ Module Name:      ffdccexpl.msg
$
$ Purpose:   Used to demonstrate the use of First Failure Data Capture
$           (FFDC) command line and application programming interfaces.
$           The messages within this message catalog file are used to
$           report failure conditions to the FFDC Error Stack, and to
$           construct AIX Error Logging templates.
$ -----
$ %Z%M%   %I%   %W% %G% %U%
$ -----
$ double-quotes (") will be used to delimit messages in the catalog
$quote "
$ -----
$ Messages used to construct AIX Error Log templates in client software example
$set client
$ --
$ -- Error Descriptions --
$ --
EMSG501 "ffdcsvr Service Not Configured Properly"
EMSG502 "ffdcclnt Application Started"
EMSG503 "ffdcclnt Internal Failure - Terminating"
EMSG504 "ffdcclnt Instructed to Terminate"
EMSG505 "ffdcclnt Application Completed"
EMSG506 "ffdcclnt Unable to Start"
EMSG507 "ffdcclnt Configuration Failure - Terminating"
EMSG508 "ffdcclnt Failure - System Resources Unavailable"
EMSG509 "No ffdcsrv Servers Available"
EMSG510 "ffdcsvr Connected After Retry Attempts Made"
EMSG511 "ffdcsvr Connection Lost - Terminating ffdccInt"
EMSG512 "ffdcsvr Application Timed Out - Terminating ffdccInt"

```

```

EMSG513 "ffdcsvr Response Not Detected - Terminating ffdccInt"
EMSG514 "ffdcsvr Response Invalid - Terminating ffdccInt"
EMSG515 "ffdcInt Application Exiting Normally"
EMSG516 "ffdcsvr Failure - Terminating ffdccInt"
$ --
$ -- Probable Causes --
$ --
EMSG521 "ffdcsvr is not listed in the local system's /etc/services file"
EMSG522 "ffdcsvr protocol is not defined as 'tcp' in the /etc/services file"
EMSG523 "ffdcsvr service intentionally suspended on this system"
EMSG524 "ffdcsvr entry manually removed from /etc/services file"
EMSG525 "Unable to create a socket to contact the ffdcsrv application"
EMSG526 "Cannot connect SOCK_STREAM socket to ffdcsrv service port"
EMSG527 "Cannot open configuration file for ffdccInt application"
EMSG528 "Cannot transmit request to ffdcsrv application through socket"
EMSG529 "Cannot Query Socket Connection For Input"
EMSG530 "Cannot read information from ffdcsrv application through socket"
EMSG531 "Received termination signal SIGTERM"
EMSG532 "Operating system or user gave instructions to terminate ffdccInt"
EMSG533 "Daemon Initialization Subroutine dae_init() Failed"
EMSG534 "ffdcInt Started in an unsupported manner"
EMSG535 "System Resource Controller attempted to started ffdccInt"
EMSG536 "Cannot open or access configuration file /tmp/ffdc.conf"
EMSG537 "Cannot read data from file /tmp/ffdc.conf, or file is empty"
EMSG538 "Operating system detected memory buffer failure"
EMSG539 "Cannot establish socket connection to ffdcsrv application on all\n\
\t\t listed servers in the /tmp/ffdc.conf configuration file"
EMSG540 "Unable to connect to one or more ffdcsrv application servers"
EMSG541 "Socket connection with ffdcsrv application dropped by ffdcsrv"
EMSG542 "ffdcsvr failed to respond within the time permitted"
EMSG543 "Unable to determine if ffdcsrv application responded to request"
EMSG544 "Failure in dynamic memory allocation"
EMSG545 "Response from ffdcsrv application not in the anticipated format"
EMSG546 "ffdcsvr application reported a failure in response to a service\n\
\t\t request made by this application"
EMSG547 "Unable to create output file /tmp/ffdc.out"
$ --
$ -- Other Causes --
$ --
EMSG551 "Configuration process failed to define the ffdcsrv service"
EMSG552 "The socket() subroutine returned an unexpected failure response"
EMSG553 "The fopen() subroutine returned an unexpected failure response"
EMSG554 "The write() subroutine returned an unexpected failure response"
EMSG555 "The dae_init() routine indicated that it experienced a failure in\n\
\t\t performing its expected function"
EMSG556 "Consult the Detail Data section for further information"
EMSG557 "ffdcInt application incorrectly placed under System Resource\n\
\t\t Controller control"
EMSG558 "Configuration process failed to create the /tmp/ffdc.conf file"
EMSG559 "Configuration file is empty"
EMSG560 "Configuration file permissions have been changed to deny access\n\
\t\t to this file by this process"
EMSG561 "Directory permissions have been changed to deny access to the\n\
\t\t directory by this process"
EMSG562 "Owner of this process has too many files or sockets opened"
EMSG563 "Operating system memory buffer shortage"
EMSG564 "ffdcsvr Servers listed in /tmp/ffdc.conf are not available"
EMSG565 "ffdcsvr Application disabled on one or more servers"
EMSG566 "ffdcsvr Servers listed in /tmp/ffdc.conf do not exist"
EMSG567 "ffdcsvr Servers denied the ffdccInt application access"
EMSG568 "ffdcsvr application terminated prematurely on server"
EMSG569 "ffdcsvr application encountered a failure"
EMSG570 "Port for ffdcsrv application possibly used by a different application"
EMSG571 "ffdcsvr application may be contending for scare resources on\n\
\t\t server system"
EMSG572 "Network connections may be congested"
EMSG573 "The select() system call continually failed with an EAGAIN error"

```

```

EMSG574 "Application may have exceeded memory usage limits"
EMSG575 "Owner of this process may have exceeded user memory limits"
EMSG576 "ffdcsvr application on server may be an unsupported level of the\n\
\t\t application"
EMSG577 "ffdcsvr application experienced a failure condition"
EMSG578 "The connect() subroutine returned an unexpected failure response"
EMSG579 "The select() subroutine returned an unexpected failure response"
EMSG580 "The read() subroutine returned an unexpected failure response"
EMSG581 "The dae_init() subroutine returned an unexpected failure response"
EMSG582 "Operating system file table is full"
EMSG583 "No space available in the /tmp file system"
EMSG584 "The open() system call continually failed with an EINTR error"
EMSG585 "Owner of this process may have exceeded disk usage limits"
EMSG586 "The open() system call returned with an unexpected failure response"
$ --
$ -- Actions --
$ --
EMSG601 "Verify that the ffdcsvr service has not been intentionally\n\
\t\t suspended"
EMSG602 "Examine /etc/services on other systems and locate the ffdcsvr\n\
\t\t number"
EMSG603 "Manually update /etc/services on this system to define the ffdcsvr\n\
\t\t service and its port"
EMSG604 "Do not attempt to use this software until the problem is resolved"
EMSG605 "Contact the FFDC software vendor to report and resolve the problem"
EMSG606 "Report this contition to the system administrator"
EMSG607 "System administrators should perform any problem determination\n\
\t\t and resolution procedures associated with the libdae_bsd.a and\n\
\t\t libdae.a libraries"
EMSG608 "If conditions persist, contact the libdae.a library vendor to\n\
\t\t report and resolve the problem"
EMSG609 "ffdcclnt does not support the System Resource Controller"
EMSG610 "Remove ffdcclnt from the System Resource Controller control"
EMSG611 "Activate ffdcclnt using the supported methods"
EMSG612 "Consult the ffdcclnt documentation for further usage assistance"
EMSG613 "Manually create the /tmp/ffdc.conf configuration file\n\
\t\t Consult the ffdcclnt documentation for assistance on creating this file"
EMSG614 "Alter permissions on the configuration file to grant read-only\n\
\t\t access to the user of this process"
EMSG615 "Alter permissions on the directory to grant read and execute\n\
\t\t permissions to the user of this process"
EMSG616 "Owner of process must close unneeded files or sockets"
EMSG617 "Execute ffdcclnt when fewer network applications are active\n\
\t\t on this node"
EMSG618 "Examine other failure reports to determine if the system is\n\
\t\t frequently experiencing buffer shortage failures"
EMSG619 "Extend the memory used by the operating system for memory buffers"
EMSG620 "Verify that the servers listed in the /tmp/ffdc.conf configuration\n\
\t\t file are valid and operational"
EMSG621 "Correct any invalid server names in the /tmp/ffdc.conf file"
EMSG622 "If a file name is listed in Detailed Data, consult that file\n\
\t\t for specifics on the failures detected"
EMSG623 "No corrective action required, but investigation is recommended"
EMSG624 "Examine Error Log on ffdcsvr server system for ffdcsvr failures"
EMSG625 "Perform any recommended actions for any ffdcsvr failures listed"
EMSG626 "If port numbers do not agree with this system, manually update\n\
\t\t /etc/services on this system to define the correct port"
EMSG627 "Execute ffdcclnt when less network activity is present"
EMSG628 "Examine other processes owned by the same user for excessive\n\
\t\t memory usage"
EMSG629 "Terminate other applications for this user and try the application\n\
\t\t again"
EMSG630 "If conditions persist, contact the ffdcclnt application vendor for\n\
\t\t assistance in resolving this problem"
EMSG631 "Verify that the ffdcsvr application on the server system is a\n\
\t\t supported level"
EMSG632 "Contact the ffdcclnt application vendor for assistance if all\n\

```

```

\t\t previous instructions do not indicate any errors"
MSG633 "If the REFERENCE CODE field of the Detail Data is not blank,\n\
\t\t examine the failure report associated with the REFERENCE CODE"
MSG634 "Issue fcdecode command to interpret the REFERENCE CODE"
MSG635 "Alter permissions on the directory to grant write and execute\n\
\t\t permissions to the user of this process"
MSG636 "Make more space available in the /tmp/file system"
MSG637 "Close any unused and opened files on this system"
MSG638 "Remove any unused files owned by the owner of this process"
$ --
$ -- Detail Data --
$ --
MSG651 "Application Name"
MSG652 "Service Name Requested"
MSG653 "Protocol Requested"
MSG654 "Process Name"
MSG655 "Process Identifier"
MSG656 "Started by Parent Process Number"
MSG657 "Internal Routine Name"
MSG658 "Failed Function Call"
MSG659 "Socket Number"
MSG660 "Socket Structure Address Value"
MSG661 "Socket Structure Size Value"
MSG662 "Server Address Used in sockaddr_in Structure"
MSG663 "Service Port Used in sockaddr_in Structure"
MSG664 "Socket Internet Protocol Family Used in sockaddr_in Structure"
MSG665 "Configuration File Name Used"
MSG666 "Mode Used"
MSG667 "Loop Counter (number of iterations in transmit or receive request)"
MSG668 "Overall Size of Buffer Being Transmitted"
MSG669 "Number of Bytes Attempted in This Transmission"
MSG670 "Contents of Data Buffer (up to first 56 bytes)"
MSG671 "fc_set Structure Address Value"
MSG672 "First 4 Bytes of fd_set Structure Contents"
MSG673 "Overall Size of Input Being Received"
MSG674 "Number of Bytes Attempted in This Reception"
MSG675 "Storage Buffer Address Value"
MSG676 "Value of signal received"
MSG677 "Error code returned from dae_init subroutine"
MSG678 "Failing libdae.a subroutine name"
MSG679 "Failure message from libdae.a"
MSG680 "dae_init Parent Code Used"
MSG681 "Address provided for Parent Code parameter"
MSG682 "Address provided for Error Information parameter"
MSG683 "Process Owner User Identifier"
MSG684 "Failing routine return code"
MSG685 "Diagnostic Information available in the following file, if listed"
MSG686 "ffdcserv Server used at time of failure"
MSG687 "Number of bytes requested from the operating system"
MSG688 "Response information from ffdcerv"

```

Appendix B. First Failure Data Capture Messages — 2615

2615-001 **libct_ffdc.a Error: The parameter passed to the library routine `fc_init_stack` is not valid. The valid values are: `FC_STACK`, `FC_TRACE`, `FC_CREAT`, `FC_INHERIT`. This process cannot make recordings to the FFDC Error Stack.**

Explanation: The caller provided a value that was not valid as the parameter to the `fc_init_stack` routine.

User Response: Correct the source code to use the proper option, recompile the source code, and retry the call to this routine.

2615-002 **libct_ffdc.a Error: *library option* is not supported by this version of the *library routine name* routine.**

Explanation: The caller specified an option to the named routine which this version does not support at this time.

User Response: Upgrade the `libct_ffdc.a` library to a version that supports the specified option and recompile the application. If no upgrade is available, remove use of this option from the source code module, recompile the source code, and attempt the function again.

2615-003 **libct_ffdc.a Error: Options *symbolic name for option provided* and *libct_ffdc.a* cannot be combined as options to the *library routine name* routine. This process cannot make recordings to the FFDC Error Stack.**

Explanation: The caller used mutually exclusive options to the named library routine.

User Response: Correct the source code to use the proper combination of options, recompile the source code, and retry the call to this routine.

2615-004 **libct_ffdc.a Error: Cannot inherit an FFDC Environment because an FFDC Environment has not been previously established. This process cannot make recordings to the FFDC Error Stack.**

Explanation: The `fc_init_stack()` routine was used to inherit an FFDC Environment, but no ancestor of this process has previously established an FFDC Environment. Because of this, the attempt to inherit was not successful.

User Response: Do not attempt to record information to the FFDC Error Stack within this process. If the client wishes to have an FFDC Environment exist, modify the source code to create an FFDC Environment instead of

inheriting it. If the client only wishes to use an FFDC Environment if an ancestor has established an FFDC Environment, this failure can be ignored.

2615-005 **libct_ffdc.a Error: Cannot create or inherit an FFDC Environment because an FFDC Environment already exists for this process. `fc_init_stack` may have been attempted several times by this process. Examine the source code for errors.**

Explanation: The `fc_init_stack()` routine was used to create or inherit an FFDC Environment when the process already has an FFDC Environment established for itself. This session was established earlier in the process' execution, or earlier by another thread of this process.

User Response: Examine the source code for this process to ensure that multiple attempts to establish the FFDC Environment have not been made by accident. Correct the source code. The process can safely ignore this failure condition for now, but the source code should be corrected.

2615-006 **libct_ffdc.a Error: The FFDC Environment for this process appears to be corrupted. The FFDC Environment is not usable. This process cannot make recordings to the FFDC Error Stack.**

Explanation: The FFDC Environment is composed of four process environment variables: `FFDCSTACK`, `FFDCTRACE`, `FFDCORIG`, and `FFDCPID`. At least one of these variables have been removed or set to a null value. The FFDC Environment is not usable in this condition, meaning that the process cannot establish an FFDC Environment or record information to the FFDC Error Stack.

User Response: Do not attempt to record information to the FFDC Error Stack within this process. Check the system for users or processes that may be setting one or more of these process environment variables. Check the application source code for instructions that set the process environment, and ensure that they are not setting one or more of these process environment variables. Consult the FFDC documentation for further problem determination procedures.

2615-007 **libct_ffdc.a Error: Failure in allocating memory. An FFDC Environment cannot be established. This process cannot make recordings to the FFDC Error Stack. Check this process for memory leak problems, and check the system for processes hoarding memory.**

Explanation: Attempts to allocate memory for expanding the process environment were unsuccessful. This implies that either the process is allocating too much memory, or a process elsewhere in the system is allocating too much memory.

User Response: Do not attempt to record information to the FFDC Error Stack within this process. Examine the process for memory leaks and for overuse of memory. Examine the system for processes hoarding system memory, and request that the system administrator terminate these processes.

2615-008 **libct_ffdc.a Error: An FFDC Environment cannot be established for this process. Either the process execution environment could not be modified by the library, or temporary work space could not be obtained in the /tmp file system. This process cannot make recordings to the FFDC Error Stack. This process will also be unable to obtain FFDC Failure Identifiers for reports it makes to the AIX Error Log. Verify that space is available in the /tmp file system on this node, and notify the system administrator if /tmp appears to be near its capacity.**

Explanation: The `fc_init_stack` routine cannot modify the process environment. An FFDC Environment cannot be established.

User Response: Do not attempt to record information to the FFDC Error Stack within this process. Follow the instructions listed in the previous message.

2615-009 **libct_ffdc.a Error: The FFDC Environment could not be established. An FFDC Error Stack File appears to exist already for this process in the *directory name where FFDC Error Stacks reside on system* directory. This process cannot make recordings to the FFDC Error Stack.**

Explanation: The `fc_init_stack` routine attempted to reserve an FFDC Error Stack file for this process in the named directory, but a file already exists in that directory using the name that `fc_init_stack` tried to reserve. The file name reserved by `fc_init_stack` has the format of:
command_name.process_identifier.today's_date.

User Response: Do not attempt to record information to the FFDC Error Stack within this process. Use the `fcclear` command to remove any unneeded FFDC Error Stack files from this system. Run the application again after the unneeded FFDC Error Stack files are removed.

2615-010 **libct_ffdc.a Error: The FFDC Error Stack directory *directory name where FFDC Error Stacks reside on system* cannot be accessed. The directory may be missing, unmounted, or permissions may have been changed on the directory. Contact the system administrator and report this problem. This process cannot make recordings to the FFDC Error Stack at this time.**

Explanation: The named directory cannot be accessed. Either the directory has been removed, unmounted, or its permissions have been changed from the values created when the FFDC software was installed.

User Response: Do not attempt to record information to the FFDC Error Stack within this process. Contact the system administrator and report the problem. The system administrator should mount the directory if it is unmounted, create the directory if it has been removed, or change the permissions on the directory to permit access. Instructions for performing these repairs are given in the FFDC problem determination documentation.

2615-011 **libct_ffdc.a Error: An unexpected failure occurred in the *internal routine name* routine. This process cannot make recordings to the FFDC Error Stack at this time. Contact the IBM Customer Support Center for assistance in resolving this error.**

Explanation: An error occurred within the FFDC software that was not expected. This failure implies that a coding mistake was made in the FFDC software.

User Response: Do not attempt to record information to the FFDC Error Stack within this process. Report this problem to your system administrator, and have the system administrator contact the IBM Support Center. System administrators should consult the FFDC problem determination documentation prior to contacting the IBM Customer Support Center.

2615-012 **libct_ffdc.a Error: An FFDC Environment does not exist for this process. This process cannot record failure information to the FFDC Error Stack.**

Explanation: An FFDC Error Stack was not established by this process. The process is not able to record information to an FFDC Error Stack.

User Response: If this process wishes to record failure information to an FFDC Error Stack file, create or inherit an FFDC Environment using the `fc_init_stack` library routine. If this error is returned from the `fc_test_stack` routine, it is to be treated as an indication that the FFDC Environment does not exist, and the process should proceed accordingly; it is not an actual failure in this case, but status information.

2615-013 **libct_ffdc.a Error: The FFDC library routine name routine could not record information on this incident to the System Log. No incident report was filed by this routine.**

Explanation: This routine could not record information about this incident to the System Log. The System Log could not be accessed, or attempts to write to the System Log were unsuccessful. No other logging mechanism is available to record this information on this system.

User Response: Ask the system administrator to verify the operation of the System Log. System administrators should consult the appropriate operating system documentation to diagnose the status of the System Log.

2615-014 **libct_ffdc.a Error: The FFDC Error Stack file FFDC Error Stack file cannot be accessed. The file may be missing, corrupted, or, permissions may have been changed on the file to prohibit access. Contact the system administrator and report this problem. This process cannot make recordings to the FFDC Error Stack at this time.**

Explanation: The named file cannot be accessed. Either the file has been removed, unmounted, or its permissions have been changed from the values that were used by the FFDC library when the file was created.

User Response: Do not attempt to record information to the FFDC Error Stack within this process. Contact the system administrator and report the problem. The system administrator should verify the permissions on the file and correct them if necessary. Instructions for performing these repairs are given in the FFDC problem determination documentation.

2615-015 **libct_ffdc.a Error: The name reserved for the FFDC Error Stack file: FFDC Error Stack File is the name of an existing directory. Ask the system administrator to ensure that system users are not creating subdirectories in the directory: directory where FFDC Error Stacks reside. If this problem occurs in multiple applications, contact the system administrator to have the problem reported to the IBM Customer Support Center.**

Explanation: The file name reserved for the FFDC Error Stack file for this application is actually the name of a directory. Either a system user has created a directory of the same name on the system, or there is a problem with the `fc_init_stack()` routine that must be addressed by the IBM Customer Support Center.

User Response: Contact the system administrator and report the problem. System administrators should verify that the file name is indeed a directory, and was not created by a system user accidentally. Remove any subdirectories in the named directory to prevent the problem from happening again. If the problem is reported for multiple applications, the problem indicates an error in the `fc_init_stack()` or `fc_push_stack` library routines, and the IBM Customer Support Center should be contacted to resolve the problem.

2615-016 **libct_ffdc.a Error: The FFDC Error Stack file name of FFDC Error Stack File cannot be created. Permissions on the directory have been changed to prohibit creating files in this directory. Contact the system administrator and report this problem. This process cannot make recordings to the FFDC Error Stack at this time.**

Explanation: The named file cannot be created. The permissions on the directory have been changed from the permissions set when the FFDC software was installed.

User Response: Do not attempt to record information to the FFDC Error Stack within this process. Contact the system administrator and report the problem. The system administrator should change the permissions on the directory to permit processes to create files in this directory. Instructions for performing this repair are given in the FFDC problem determination documentation.

2615-017 **libct_ffdc.a Error: Insufficient space exists in the file system containing the directory: *directory name* to create an FFDC Error Stack file. This routine attempted to reserve an FFDC Error Stack file of a minimum size of *minimum size of FFDC Error Stack File* bytes while leaving at least five percent of the file system capacity available. The FFDC Error Stack file could not be reserved under these constraints. This application cannot record information to the FFDC Error Stack until more space becomes available in the file system.**

Explanation: An FFDC Error Stack cannot be created in the named directory within the safety precautions used by First Failure Data Capture. This application will be unable to record information to the FFDC Error Stack until more space becomes available in the file system containing this directory.

User Response: Report this problem to the system administrator. System administrators should remove any FFDC Error Stack files from this system that are no longer needed, using the `fcstkclear` command to remove them, or add more space to the file system containing this directory. System administrators should also check other directories in the same file system for other obsolete files that can be removed to make more space available.

2615-018 **libct_ffdc.a Error: Unable to lock the file: *FFDC Error Stack filename* for exclusive use. The file may have been removed by another application, or another application may have locked this file for use and become hung. This process is unable to record incident information to the FFDC Error Stack at this time.**

Explanation: Attempts to lock this file for exclusive use were not successful. The file may have been removed by another user or application, or another application also using this FFDC Error Stack file may have become hung while attempting to record information to the FFDC Error Stack. This report cannot be made to the FFDC Error Stack at this time. The application can attempt to make this report again at a later time.

User Response: Pause the application for a short period of time, and attempt to record the incident information again. If this condition persists, verify that the file exists and was not removed by another user or application. Check any ancestor or descendent processes of this application to determine if any of them have become hung. Terminate any ancestor or descendent process that have hung.

2615-019 **libct_ffdc.a Error: Unable to open the FFDC Error Stack file: *FFDC Error Stack filename*. The file may have been removed by another application, or the file permissions may have been altered on the file to prohibit this application from opening the file. Permissions on the directory containing this file may have changed to prohibit files within it to be modified. This application cannot record information to the FFDC Error Stack at this time. Error code from `open()` system call: *errno* from `open()` routine.**

Explanation: An attempt to open this file for modification were unsuccessful. The file may have been removed, or the permissions on the file may have been changed by another user or application to prohibit other applications from altering this file. Permissions on one or more of the directories within the file path name may have been altered by other users or applications to prohibit applications from modifying files. As a result, this application cannot record information to the FFDC Error Stack at this time.

User Response: Report the problem to the system administrator. System administrators should verify that the file exists, and that the file and the directories in the file path name permit applications to modify the file contents.

2615-020 **libct_ffdc.a Error: Unable to update the control information within the FFDC Error Stack file: *FFDC Error Stack filename*. The file may be corrupted or this problem can indicate an internal error in the FFDC library software. This process should consider the FFDC Error Stack unusable. Verify that the file exists and can be viewed using the `fcstkprpt` command. If `fcstkprpt` fails to indicate an error with the FFDC Error Stack file, contact the system administrator to have this problem reported to the IBM Customer Service Center. Error code from the `write()` library call: *errno* from `write()` routine.**

Explanation: The FFDC Error Stack internal control information could not be modified. The file may be corrupt, the file system may be experiencing problems, or an internal problem may exist in the FFDC library that requires the attention of the IBM Customer Support Center to rectify. The application should consider the FFDC Error Stack unusable.

User Response: Do not attempt to record further information to the FFDC Error Stack from this application. Contact the system administrator and report this problem. System administrators should verify that the file system containing this file is not experiencing

problems, and should also verify that the hardware supporting this file system is also not experiencing problems. If the file system and the hardware appear to be functioning properly, contact the IBM Customer Support Center to report this problem.

2615-021 **libct_ffdc.a Error: Unable to relinquish exclusive access to the FFDC Error Stack file: *FFDC Error Stack filename*. This process should consider the FFDC Error Stack unusable. Other ancestor and descendent processes may find it impossible to use the FFDC Error Stack if they also have FFDC Environments established. Report this problem to the system administrator. Do not attempt to use the FFDC Error Stack from this process. Error code from the lockf() library call: *errno from write() routine*.**

Explanation: The FFDC library could not release the lock it obtained for the named FFDC Error Stack file. This is most likely an indication of an internal error within the FFDC library that must be reported to the IBM Customer Support Center.

User Response: Do not attempt to use the FFDC Error Stack from this application. Expect ancestor or descendent processes that also use this FFDC Error Stack to report FC_STACK_LOCK failures while this application runs. Report this problem to the system administrator. System administrators should verify that the file system containing this file is not experiencing problems, and that the hardware supporting the file system is not experiencing problems. Once these have been verified, contact the IBM Customer Support Center to report this problem.

2615-022 **libct_ffdc.a Error: The FFDC Error Stack file: *FFDC Error Stack filename* appears to have been corrupted. The internal routine that detected this condition is: *internal routine*. This application should consider the FFDC Error Stack unusable, and should not attempt to make any more recordings to the FFDC Error Stack.**

Explanation: The contents of the FFDC Error Stack appear to be corrupted. File control information may contain values outside of the expected ranges, or previously recorded information may appear to be not valid. An application may have modified this file using an interface other than `fc_push_stack` to modify the contents of this file (such as an editor).

User Response: Do not attempt to use this FFDC Error Stack file from this application or any other. Examine the FFDC Error Stack file ownership values and permissions to determine if the file appears to have been modified by some outside source.

2615-023 **libct_ffdc.a Error: Unable to record incident information to the FFDC Error Stack file: *FFDC Error Stack filename*. The file may be corrupted, or this problem can indicate an internal error in the FFDC library software. This process should consider the FFDC Error Stack unusable. Verify that the file exists and can be viewed using the `fcstkrpt` command. If `fcstkrpt` fails to indicate an error with the FFDC Error Stack file, contact the system administrator to have this problem reported to the IBM Customer Service Center. Error code from the write() library call: *errno write() routine*.**

Explanation: This routine could not record information on the incident reported by the caller to the FFDC Error Stack file named in this message. The file may be corrupt, the file system may be experiencing problems, or an internal problem may exist in the FFDC library that requires the attention of the IBM Customer Support Center to rectify. The application should consider the FFDC Error Stack unusable.

User Response: Do not attempt to record further information to the FFDC Error Stack from this application. Contact the system administrator and report this problem. System administrators should verify that the file system containing this file is not experiencing problems, and should also verify that the hardware supporting this file system is also not experiencing problems. If the file system and the hardware appear to be functioning properly, contact the IBM Customer Support Center to report this problem.

2615-024 **libct_ffdc.a Error: Unable to set the read-write file pointer within the FFDC Error Stack file: *FFDC Error Stack filename* to offset position *file read-write offset position*. The file may be corrupted, or this problem can indicate an internal error in the FFDC library software. This process should consider the FFDC Error Stack unusable. Verify that the file exists and can be viewed using the `fcstkrpt` command. If `fcstkrpt` fails to indicate an error with the FFDC Error Stack file, contact the system administrator to have this problem reported to the IBM Customer Service Center. Error code from the write() library call: *errno from write() routine*.**

Explanation: The routine could not set the read-write pointer to the position within the named FFDC Error Stack file where information is to be recorded. The file may be corrupt, the file system may be experiencing problems, or an internal problem may exist in the FFDC

library that requires the attention of the IBM Customer Support Center to rectify. The application should consider the FFDC Error Stack unusable.

User Response: Do not attempt to record further information to the FFDC Error Stack from this application. Contact the system administrator and report this problem. System administrators should verify that the file system containing this file is not experiencing problems, and should also verify that the hardware supporting this file system is also not experiencing problems. If the file system and the hardware appear to be functioning properly, contact the IBM Customer Support Center to report this problem.

2615-025 libct_ffdc.a Error: Unable to create or inherit an FFDC Environment. This process environment has been set to prevent the creation or inheritance of FFDC Error Stacks. The system administrator or other privileged user has set this control. While this process remains capable of recording information to the AIX Error Log and the BSD System Log through the FFDC interfaces, this process is not capable of using an FFDC Error Stack. Contact the system administrator if an FFDC Error Stack is required by this process, and ask to have this FFDC function restored to the process environment.

Explanation: This routine was not able to inherit or create an FFDC Environment to use FFDC Error Stacks, because the system administrator activated a control that disabled the use of FFDC Error Stacks in this process environment. The process remains capable of recording information to the AIX Error Log and the BSD System Log.

User Response: Contact the system administrator to request that this restriction be lifted for this process environment. System administrators should consult the First Failure Data Capture administration documentation for instructions on restriction and selected enablement of the FFDC Error Stacks in various process environments.

2615-501 fcinit Error: The *option* option was specified more than once.

Explanation: The user specified the same option more than once to this command. An FFDC Environment was not established for this process.

User Response: This message is accompanied by a syntax message for the fcinit command. Select the appropriate option and attempt the command again. Use the -h option to learn more about this command.

2615-502 fcinit Error: Unknown option specified - *option specified-argument provided for the option.*

Explanation: The user specified an option that was not valid for this command. An FFDC Environment was not established for this process.

User Response: This message is accompanied by a syntax message for the fcinit command. Select the appropriate option and attempt the command again. Use the -h option to learn more about this command.

2615-503 fcinit Error: Cannot specify both the *option* and the *option* options to this command.

Explanation: The user specified mutually exclusive options to this command. An FFDC Environment was not established for this process.

User Response: This message is accompanied by a syntax message for the fcinit command. Select the appropriate option and attempt the command again. Use the -h option to learn more about this command.

2615-504 fcinit Error: Cannot obtain information about this process from the operating system. This indicates an internal problem with this command. Error code from getprocs() is *error code returned from getprocs C library routine.* Contact the IBM Customer Support Center to report this problem.

Explanation: The command was not successful in obtaining information about its own process from the operating system. The getprocs() routine was not successful within the command. This should not occur unless the command was written incorrectly, or unless the operating system is experiencing serious problems. An FFDC Environment was not established for this process.

User Response: Contact the system administrator to have the problem reported to the IBM Customer Support Center. System administrators should contact the IBM Customer Support center for assistance on resolving this problem.

2615-505 **fcinit Error: This command is being executed in its own shell. This command should be executed within the current shell, so that the command can alter the current shell's process environment. To do this in Korn or Bourne Shells, use this command as follows: `fcinit option` To do this from a C Shell, use the command as follows: `source fcinit option`.**

Explanation: The command detected that it was being executed in its own sub-shell, and not in the user's shell. The command cannot modify the user's shell to establish an FFDC Environment when used in this manner. An FFDC Environment was not established for this process.

User Response: Execute this command in the user's shell as described by the error message.

2615-506 **fcinit Error: Cannot inherit an FFDC Environment because an FFDC Environment has not been previously established. This process cannot make recordings to the FFDC Error Stack.**

Explanation: The fcinit command was used to inherit an FFDC Environment, but no ancestor of this process has previously established an FFDC Environment. Because of this, the attempt to inherit was not successful.

User Response: Do not attempt to record information to the FFDC Error Stack within this process. If the client wishes to have an FFDC Environment exist, modify the use of fcinit to create an FFDC Environment instead of inheriting it. If the client only wishes to use an FFDC Environment if an ancestor has established an FFDC Environment, this failure can be ignored.

2615-507 **fcinit Error: Cannot create or inherit an FFDC Environment, because an FFDC Environment already exists for this process. fcinit may have been attempted several times by this process. Examine the use of this command for possible repetitive use.**

Explanation: The fcinit command was used to create or inherit an FFDC Environment when the process already has an FFDC Environment established for itself. This session was established earlier in the process's execution.

User Response: Examine the source code for this process to ensure that multiple attempts to establish the FFDC Environment have not been made by accident. Correct the source code. The process can safely ignore this failure condition for now, but the source code should be corrected.

2615-508 **fcinit Error: The FFDC Environment for this process appears to be corrupted. The FFDC Environment is not usable. This process cannot make recordings to the FFDC Error Stack.**

Explanation: The FFDC Environment is composed of four process environment variables: FFDCSTACK, FFDCTRACE, FFDCORIG, and FFDCPID. At least one of these variables have been removed or set to a null value. The FFDC Environment is not usable in this condition, meaning that the process cannot establish an FFDC Environment or record information to the FFDC Error Stack.

User Response: Do not attempt to record information to the FFDC Error Stack within this process. Check the system for users or processes that may be setting one or more of these process environment variables. Check the application source code for instructions that set the process environment, and ensure that they are not setting one or more of these process environment variables. Consult the FFDC documentation for further problem determination procedures.

2615-509 **fcinit Error: Failure in allocating memory. An FFDC Environment cannot be established. This process cannot make recordings to the FFDC Error Stack. Check this process for memory leak problems, and check the system for processes hoarding memory.**

Explanation: Attempts to allocate memory for expanding the process environment were not successful. This implies that either the process is allocating too much memory, or a process elsewhere in the system is allocating too much memory.

User Response: Do not attempt to record information to the FFDC Error Stack within this process. Examine the process for memory leaks and for overuse of memory. Examine the system for processes hoarding system memory, and request that the system administrator terminate these processes.

2615-510 **fcinit Error: An FFDC Environment cannot be established for this process. Either the process execution environment could not be modified by the library, or temporary work space could not be obtained in the /tmp file system. This process cannot make recordings to the FFDC Error Stack. This process will also be unable to obtain FFDC Failure Identifiers for reports it makes to the AIX Error Log. Verify that space is available in the /tmp file system on this node, and notify the system administrator if /tmp appears to be near its capacity.**

Explanation: The fcinit command cannot modify the process environment. An FFDC Environment cannot be established.

User Response: Do not attempt to record information to the FFDC Error Stack within this process. Follow the instructions given in the above message.

2615-511 **fcinit Error: The FFDC Environment could not be established. An FFDC Error Stack File appears to exist already for this process in the *directory* where FFDC Error Stacks reside *directory*. This process cannot make recordings to the FFDC Error Stack.**

Explanation: The fcinit command attempted to reserve an FFDC Error Stack file for this process in the named directory, but a file already exists in that directory using the name that fc_init_stack tried to reserve. The file name reserved by fc_init_stack has the format of: *command name.process identifier.today's date*.

User Response: Do not attempt to record information to the FFDC Error Stack within this process. Use the fcclear command to remove any unneeded FFDC Error Stack files from this system. Run the application again after the unneeded FFDC Error Stack files are removed.

2615-512 **fcinit Error: The FFDC Error Stack directory *directory name* cannot be accessed. The directory may be missing, unmounted, or permissions may have been changed on the directory. Contact the system administrator and report this problem. This process cannot make recordings to the FFDC Error Stack at this time.**

Explanation: The named directory cannot be accessed. Either the directory has been removed, unmounted, or its permissions have been changed from the values created when the FFDC software was installed.

User Response: Do not attempt to record information

to the FFDC Error Stack within this process. Contact the system administrator and report the problem. The system administrator should mount the directory if it is unmounted, create the directory if it has been removed, or change the permissions on the directory to permit access. Instructions for performing these repairs are given in the FFDC problem determination documentation.

2615-513 **fcinit Error: An unexpected failure occurred in the routine *routine name*. This process cannot make recordings to the FFDC Error Stack at this time. Contact the IBM Customer Support Center for assistance in resolving this failure.**

Explanation: An error occurred within the FFDC software that was not expected. This failure implies that a coding mistake was made in the FFDC software.

User Response: Do not attempt to record information to the FFDC Error Stack within this process. Report this problem to your system administrator, and have the system administrator contact the IBM Support Center. System administrators should consult the FFDC problem determination documentation prior to contacting the IBM Customer Support Center.

2615-514 **fcinit Error: Unable to create or inherit an FFDC Environment. This process environment has been set to prevent the creation or inheritance of FFDC Error Stacks. The system administrator or other privileged user has set this control. While this process remains capable of recording information to the AIX Error Log and the BSD System Log through the FFDC interfaces, this process is not capable of using an FFDC Error Stack. Contact the system administrator if an FFDC Error Stack is required by this process, and ask to have this FFDC function restored to the process environment.**

Explanation: This command was not able to inherit or create an FFDC Environment to use FFDC Error Stacks, because the system administrator activated a control that disabled the use of FFDC Error Stacks in this process environment. The process remains capable of recording information to the AIX Error Log and the BSD System Log.

User Response: Contact the system administrator to request that this restriction be lifted for this process environment. System administrators should consult the First Failure Data Capture administration documentation for instructions on restriction and selected enablement of the FFDC Error Stacks in various process environments.

2615-551 fcteststk Error: Too many options specified.

Explanation: The user specified too many options to the fcteststk command. The user is not told whether an FFDC Environment exists for this process or not, so the user cannot assume that the FFDC Environment is established.

User Response: Check the usage information for the command, and issue the command again using the appropriate options.

2615-552 fcteststk Error: Unknown option specified - option.

Explanation: The user specified an option that was not valid to this command. This command could not test if an FFDC Environment was established by this process.

User Response: This message is accompanied by a syntax message for the fcteststk command. Select the appropriate option and attempt the command again.

2615-553 fcteststk Error: The FFDC Environment for this process appears to be corrupted. The FFDC Environment is not usable. This process cannot make recordings to the FFDC Error Stack.

Explanation: The FFDC Environment is composed of four process environment variables: FFDCSTACK, FFDCTRACE, FFDCCORIG, and FFDCCPID. At least one of these variables have been removed or set to a null value. The FFDC Environment is not usable in this condition, meaning that the process cannot establish an FFDC Environment or record information to the FFDC Error Stack.

User Response: Do not attempt to record information to the FFDC Error Stack within this process. Check the system for users or processes that may be setting one or more of these process environment variables. Check the application source code for instructions that set the process environment, and ensure that they are not setting one or more of these process environment variables. Consult the FFDC documentation for further problem determination procedures.

2615-554 fcteststk Error: An unexpected failure occurred in the routine routine name. This process should assume that the FFDC Environment has not been established for this process, and that the process may be unable to establish an FFDC Environment. Contact the IBM Customer Support Center for assistance in resolving this failure.

Explanation: An error occurred within the FFDC software that was not expected. This failure implies that

a coding mistake was made in the FFDC software.

User Response: Do not attempt to record information to the FFDC Error Stack within this process. Report this problem to your system administrator, and have the system administrator contact the IBM Support Center. System administrators should consult the FFDC problem determination documentation prior to contacting the IBM Customer Support Center.

2615-601 fclogerr Error: Unknown option specified - option.

Explanation: The user specified an option that was not valid to this command. This command did not attempt to record information to the AIX Error Log or the BSD System Log.

User Response: This message is accompanied by a syntax message for the fclogerr command. Select the appropriate option and attempt the command again.

2615-602 fclogerr Error: The option option has been specified more than once. The option can only be specified once to this command. Information will not be recorded to the System Log or to the AIX Error Log.

Explanation: The user provided multiple instances of the same option to this command. This command permits options to be used only once. Information will not be recorded to the System Log and the AIX Error Log.

User Response: The caller must correct the usage of this command in order for the command to successfully record information to the System Log and the AIX Error Log. Correct the command to specify options only once. Reissue the command after correcting its usage.

2615-603 fclogerr Warning: The 'i' option was not provided to this command, or the file name provided as an argument to the 'i' option is not a valid file name. fclogerr cannot obtain the code number of the AIX Error Log template to be used in this recording. The fclogerr command will use its own AIX Error Log template to record this information: AIX Error Log Template Label: AIX Error Logging Template used by First failure Data Capture, Identifier of this template:code number or identifier for the AIX Error Logging template. .

Explanation: The fclogerr command has to be provided with the name of a header file that defines both the symbolic and numeric codes for the AIX Error Logging Template used to record information to the AIX Error Log. Without this header file, fclogerr cannot obtain the numeric code to use for the template. Without

this code, the command can only record a generic entry to the AIX Error Log using its own AIX Error Logging template.

User Response: Provide the name of the header file to the `fclogerr` command using the `-i` option, and issue the command again.

2615-604 **fclogerr Warning: Unable to open the file:** *name of header file specified as an argument to the `—i` option. Because the file cannot be opened, fclogerr will use the errlogger command to record information to the AIX Error Log. Error from `fopen()` system call. A description of the error encountered by the `fopen()` system call follows this message.*

Explanation: `fclogerr` could not access the file indicated in the `-i` option. Because this header file cannot be opened, `fclogerr` cannot obtain the numeric code to use for the template. Without this code, the command can only record a generic operator information message to the AIX Error Log using the `errlogger` command.

User Response: Ensure that the correct file name was used as an argument to the `-i` option, and ensure that the file exists. Examine the error message following this message, and perform the appropriate corrective action to grant this command access to the `fclogerr` command. Issue the `fclogerr` command again after the corrective action has been taken.

2615-605 **fclogerr Warning: Unable to locate the symbolic name:** *Symbolic name of the AIX Error Log Template provided as an argument to the `-t` option in the file: name of the file that was supposed to have defined this name, provided as an argument to the `-i` option. fclogerr cannot obtain the code number of the AIX Error Log template to be used in this recording. The errlogger command will be used to record information to the AIX Error Log.*

Explanation: `fclogerr` could not locate the symbolic name of the AIX Error Logging Template in the header file specified by the user. As a result, the command can only record a generic operator information message to the AIX Error Log using the `errlogger` command.

User Response: Ensure that the correct file name was used as an argument to the `-i` option. Ensure that the correct symbolic name was provided to the command as an argument to the `-t` option. Verify that the header file has not been modified or corrupted.

2615-606 **fclogerr Warning: A detail data item in the `'-d'` argument list does not have a corresponding data type listed in the `'-x'` argument list. The remaining detail data items will be treated as ALPHA data.**

Explanation: Each item in the `-d` argument list is to have a matching data type indicator in the `-x` argument list. The `-d` option specifies at least one additional detail data item that does not have a corresponding data type indicator in the `-x` argument list. `fclogerr` cannot determine what data type should be used to represent this information in the Detail Data section of the AIX Error Log entry, so the information will not be interpreted at all. The extra items of detail data are recorded as ALPHA formatted information to the AIX Error Log entry. This can lead to a confusing entry in the AIX Error Log, if the corresponding detail data item is not meant to be ALPHA formatted data.

User Response: Ensure that the number of detail data items and the number of detail data types agree. If the `-d` argument list contains imbedded white space, enclose the `-d` argument list in quotes to prevent the shell from interpreting the argument list as two separate argument lists.

2615-607 **fclogerr Warning: A detail data item in the `'-d'` argument list does not have a corresponding length listed in the `'-y'` argument list. The remaining detail data items will be treated as ALPHA data.**

Explanation: Each item in the `-d` argument list is to have a matching data length indicator in the `-y` argument list. The `-d` option specifies at least one additional detail data item that does not have a corresponding data length indicator in the `-y` argument list. `fclogerr` cannot determine how much space this item uses in the Detail Data section of the AIX Error Log entry, so the command will record the information unformatted. The extra items of detail data are recorded as ALPHA formatted information to the AIX Error Log entry. This can lead to a confusing entry in the AIX Error Log, if the corresponding detail data item is not meant to be ALPHA formatted data.

User Response: Ensure that the number of detail data items and the number of detail data lengths agree. If the `-d` argument list contains imbedded white space, enclose the `-d` argument list in quotes to prevent the shell from interpreting the argument list as two separate argument lists.

2615-608 **fclogerr Warning: The data type indicator value used as detail data type indicator is not a supported data type for this command. The supported data type indicators are: DEC HEX ALPHA. The detail data item for this data type will be recored as ALPHA formatted information.**

Explanation: The -x argument list contains an unsupported data type indicator. fclogerr cannot determine how the corresponding detail data item should be interpreted, so the command interprets it as ALPHA formatted data. This can lead to a confusing entry in the AIX Error Log, if the corresponding detail data item is not meant to be ALPHA formatted data.

User Response: Correct the usage of the fclogerr command to specify the correct data type.

2615-609 **fclogerr Warning: An unknown Log Event Type was provided to this command: Log event type specifeid as the argument to the '-e' option. Valid Log Event Types are: FFDC_EMERG FFDC_ERROR FFDC_STATE FFDC_PERF FFDC_TRACE FFDC_RECOV FFDC_DEBUG The value FFDC_DEBUG was substituted for this value in the report generated by this routine, giving this record the lowest severity indication available.**

Explanation: A value that is not valid or is unknown was provided as the Log Event Type to this routine. As a result, this command assigned the lowest severity it supports, FFDC_DEBUG, to the record. This may result in high severity indicents receiving a low severity indicator, and can result in high severity indicents being overlooked in a casual analysis of the System Log.

User Response: Correct this command to provide a valid log event type to this routine, to ensure that the System Log entry is labelled with a severity suited to the incident being reported.

2615-610 **fclogerr Warning: The caller provided both a detail data field using the '-d' option, and a detail data file using the '-f' option. This command only accepts either a detail data field or a detail data file. The detail data file will be ignored in the record generated by this routine, and the detail data field will be used instead.**

Explanation: The caller specified both the -d and the -f options to this command. One or the other must be specified.

User Response: Repair the use of this command to

provide either a detail data field or a detail data file, not both.

2615-611 **fclogerr Warning: No System Log message was provided to this command. No message was recorded in the System Log report generated by this routine, but a record was made in the System Log.**

Explanation: The user did not provide a message as an argument to the -b option of this command, or the -b option was omitted. As a result, no message was recorded in the System Log record, but the record was made to the System Log. The record contains the information on the detecting file only.

User Response: Repair the command to provide a meaningful System Log message to this command, for assisting future problem determination efforts.

2615-612 **fclogerr Warning: This command could not record information on this incident to the System Log.**

Explanation: This routine could not record information about this incident to the System Log. The System Log could not be accessed, or attempts to write to the System Log failed.

User Response: The system administrator should verify the operation of the System Log and should consult the appropriate operating system documentation to diagnose the status of the System Log.

2615-613 **fclogerr Warning: This command was unable to copy the contents of the following file: Name of detail data file provided to this routine to this directory: Directory where this routine intended to copy the file. Do not discard the original copy of this file.**

Explanation: The command could not make a copy of the file. The directory may not be available or accessible, space may be insufficient to store the file in the directory, or the routine may not be able to access the original.

User Response: Retain the original copy of the file for later use in resolving the incident being reported. In the future, ensure that the detail data file can be read by processes of this user. If conditions persist, contact the system administrator and report the problem. System administrators should consult the FFDC problem determination documentation for assistance in resolving this problem.

2615-614 **fclogerr Error: An unexpected failure occurred in the routine *internal routine name*. Error code value is *return code*. Contact the IBM Customer Support Center for assistance in resolving this problem.**

Explanation: An error occurred within the FFDC software that was not expected. This failure implies that a coding mistake was made in the FFDC software.

User Response: Report this problem to your system administrator, and have the system administrator contact the IBM Support Center. System administrators should consult the FFDC problem determination documentation prior to contacting the IBM Customer Support Center.

2615-615 **fclogerr Warning: Incomplete information was provided for the source code file reporting this incident. Default information was provided by this command on behalf of the user.**

Explanation: This command accepts a licensed program product name, a source code file name, a line of code position, and a file version number as input parameters. At least one of these items was not provided, or was not a valid value. This command provided default information in place of the missing information, but that information is of little use to potential problem investigators.

User Response: Fix the usage of this command to provide more complete detecting file information, for assisting future problem determination efforts.

2615-616 **fclogerr Warning: No resource name was provided to this command, or the command user failed to provide the 'r' option to this command. No resource name was recorded in the report generated by this routine, but the record was made.**

Explanation: The command user provided an empty string as the "resource" name to this command. As a result, no resource name was recorded in the BSD System Log record as a prefix or in the AIX Error Log record, but the record was made.

User Response: Repair the use of this command to provide a meaningful resource name to this command, for assisting future problem determination efforts.

2615-617 **fclogerr Warning: The command user provided a value for detail data size that appears invalid. As a result, this routine used the maximum length permitted for detail data. This may result in some extraneous and unrelated information being recorded in the record generated by this routine.**

Explanation: This command was provided with detailed information about an incident, but the caller either failed to provide the size of this information, or the size appears to be incorrect. In order to record part of the incident details, this command assumed that the detail data size was the maximum allowed. This may have caused some unrelated information for unrelated memory areas to be copied into the report generated by this routine.

User Response: Repair the usage of this command to provide a valid detail data size to this command, to ensure that correct and pertinent information is recorded to the report.

2615-618 **fclogerr Warning: No details on the incident were provided to this command. The detailed data section of the report made by this routine has been omitted, but the report was recorded.**

Explanation: The user failed to provide any "detail data" to this routine. As a result, no details about the incident were recorded in the report generated by this routine, although a report was recorded.

User Response: Revise the command to provide meaningful details on the incident to this command, for assisting future problem determination efforts.

2615-619 **fclogerr Warning: An unique FFDC Failure Identifier could not be generated for the report filed by this command. fclogerr created the incident report anyway.**

Explanation: This command was unable to generate a valid FFDC Error Identifier for this incident report. No FFDC Error Identifier was recorded as part of the report, and the caller does not have an FFDC Error Identifier to provided to its client to indicate where the report was made.

User Response: If this problem occurs more than once, it may indicate a possible problem within the FFDC software itself. Report multiple instances of this warning to the system administrator. System administrators should contact the IBM Customer Support Center for assistance on resolving this potential problem.

2615-651 fcdispfid Error: An option or an FFDC Error Identifier argument is required.

Explanation: This command requires either an option or an FFDC Error Identifier as an argument. The user did not specify either of these on the command line.

User Response: This message is accompanied by a syntax message for the fcdispfid command. Correct usage of this command and attempt the command again. Use the -h option to learn more about this command.

2615-652 fcdispfid Error: Unknown option specified -option:

Explanation: The user specified an option that was not valid to this command. The command did not display an FFDC Error Identifier to standard output.

User Response: This message is accompanied by a syntax message for the fcdispfid command. Correct usage of this command and attempt the command again. Use the -h option to learn more about this command.

2615-653 fcdispfid Error: An invalid FFDC Error Identifier was provided to this command. No FFDC Error Identifier is displayed to standard output.

Explanation: An FFDC Error Identifier value that was not valid, or pointer to an FFDC Error Identifier, was provided to this command.

User Response: Ensure that an FFDC Error Identifier exists before attempting to use this command. Verify that the application is passing a valid FFDC Error Identifier to this command.

2615-654 fcdispfid Error: An unexpected failure occurred in the routine *internal routine name*. Contact the IBM Customer Support Center for assistance in resolving this failure.

Explanation: An error occurred within the FFDC software that was not expected. This failure implies that a coding mistake was made in the FFDC software.

User Response: Report this problem to your system administrator, and have the system administrator contact the IBM Support Center. System administrators should consult the FFDC problem determination documentation prior to contacting the IBM Customer Support Center.

2615-701 fcstkprpt Error: Memory allocation failure. Exiting.

Explanation: An error occurred while the command was allocating memory to store incident information. The command assumes that the information is vital to

understanding the contents of the FFDC Error Stack, and without this information, the FFDC Error Stack contents cannot be correctly understood. Processing of the FFDC Error Stack ceases at this point, and the command terminates. The memory allocation failure is most likely the result of a process on the system hoarding virtual memory.

User Response: Issue the command again. If the command fails again with the same message, contact the system administrator. System administrators should check the system for processes hoarding memory and possibly remove these processes.

2615-702 fcstkprpt Error: Unexpected failure detected in the *internal routine name* routine. Error code is *error code*. Note the full text of this message, and contact the IBM Customer Support Center to report the problem.

Explanation: An unexpected failure occurred, and this failure may indicate a problem in the fcstkprpt command itself. Report this problem to the IBM Customer Support Center.

User Response: Report the problem to the system administrator, providing the full text of this error message. System administrators should report this error to the IBM Customer Support Center to resolve the problem.

2615-703 fcstkprpt Error: At least one command option must be specified.

Explanation: This command requires at least one option. The user did not specify an option to the command. The command has terminated.

User Response: This message is accompanied by a syntax message for the fcstkprpt command. Correct usage of this command and attempt the command again. Use the -h option to learn more about this command.

2615-704 fcstkprpt Error: The *option* option has been specified more than once. This option can only be specified once, and accepts only one argument. Processing of this command has ceased. Please correct the usage of this command and reissue the command with the proper options.

Explanation: The caller specified an option to this command multiple times. This specific option can only be specified once to the command, and takes only a single argument. The command has terminated.

User Response: Correct the command to specify options only once. Reissue the command after correcting its usage.

2615-705 **fcstkrpt Error: Unknown option specified - *option*.** Processing of this command has ceased. Please correct the usage of this command and reissue the command with the proper options.

Explanation: The user specified an option that was not valid to this command.

User Response: This message is accompanied by a syntax message for the fcstkrpt command. Select the appropriate option and attempt the command again.

2615-706 **fcstkrpt Error: Both the *option* and *option* options have been specified to this command. These options cannot be used together. Processing of this command has ceased. Please correct the usage of this command to specify only one of these two options, and reissue the command with the proper option.**

Explanation: The user specified mutually exclusive options to this command. Processing of the command has terminated.

User Response: This message is accompanied by a syntax message for the fcstkrpt command. Select the appropriate option and attempt the command again. Use the -h option to learn more about this command.

2615-707 **fcstkrpt Error: The *option* option was specified to this command, but the *option* option was not specified. Both options must be provided when using the *option*' option. Processing of this command has ceased. Please correct the usage of this command to provide both options, and reissue this command.**

Explanation: The user did not provide an option required with another option specified by the user. Processing of the command has terminated.

User Response: This message is accompanied by a syntax message for the fcstkrpt command. Select the appropriate option and attempt the command again. Use the -h option to learn more about this command.

2615-708 **fcstkrpt Error: The FFDC Error Identifier *FFDC Error Identifier* as created by a later version of the FFDC software. To examine the information stored for this identifier, use an 'fcstkrpt' command of the same or later version.**

Explanation:

User Response:

2615-709 **fcstkrpt Error: The FFDC Error Identifier *FFDC Error Identifier* specifies an entry stored in the following node's AIX Error Log: Cluster Name: *cluster name* where the *FFDC Error Identifier* was originally generated Node Number: *node number* where the *FFDC Error Identifier* was originally generated.**

Explanation: The caller provided an FFDC Error Identifier to this command that was generated for an AIX Error Log entry, not an FFDC Error Stack entry. The fcstkrpt command only examines the FFDC Error Stacks on this node, and therefore cannot obtain any information for this FFDC Error Identifier.

User Response: Go to the cluster name and the node indicated, and issue the appropriate command to obtain the incident information from the AIX Error Log on that node.

2615-710 **fcstkrpt Error: The FFDC Error Identifier *FFDC Error Identifier* provided to this command does not reference an FFDC Error Stack entry. Please verify that the data item above is indeed an FFDC Failure Identifier and has been entered correctly. To obtain more information about the FFDC Failure Identifier, issue the following command: **fcdecode****

Explanation: The caller provided an FFDC Error Identifier to this command that was not generated from an FFDC Error Stack entry. The fcstkrpt command only examines the FFDC Error Stacks on this node, and therefore cannot obtain any information for this FFDC Error Identifier.

User Response:

2615-711 **fcstkrpt Error: The FFDC Error Identifier *FFDC Error Identifier* provided to this command specifies an FFDC Error Stack in the following location: Cluster Name: *cluster name* where the *FFDC Error Identifier* was originally generated Node Number: *node number* where the *FFDC Error Identifier* was originally generated.**

Explanation: The caller provided an FFDC Error Identifier to this command that was generated for an FFDC Error Stack entry on a remote node. The fcstkrpt command only examines the FFDC Error Stacks on this node, and therefore cannot obtain any information for this FFDC Error Identifier.

User Response: Go to the cluster name and the node indicated, and issue the appropriate command to obtain the incident information from the AIX Error Log on that node.

2615-712 fcstkrpt Error: Unable to locate or open FFDC Error Stack file *FFDC Error Stack file name*.

Explanation:

User Response:

2615-713 fcstkrpt Error: The command experienced an unexpected failure in the *Name of the internal routine that detected a failure condition routine*. Failure code from this routine was *error code from the internal routine*. **Note the conditions present when this command was executed, and contact the system administrator. System administrators should contact the IBM Customer Support Center to report this failure.**

Explanation: An internal routine of the fcstkrpt command experienced an unexpected failure condition, one that should not exist if the command is working properly. This condition requires the attention of the IBM Customer Support Center to be resolved.

User Response: Note the conditions that were present at the time the command was executed, and contact the system administrator. System administrators should contact the IBM Customer Support Center for assistance on resolving this failure.

2615-714 fcstkrpt Error: The FFDC Error Identifier *FFDC Error Identifier value provided by the user*. **did not provide a valid name for an FFDC Error Stack. Please verify that the FFDC Error Identifier was specified correctly. Correct any error in the specification of the FFDC Error Identifier, and attempt this command again.**

Explanation: The FFDC Error Identifier, when decoded by the command, did not provide a valid FFDC Error Stack reference. The most likely cause of this failure is that the user made a mistake in typing or specifying the FFDC Error Identifier to the command.

User Response: Verify that the FFDC Error Identifier is typed correctly.

2615-715 fcstkrpt Error: The FFDC Error Identifier *FFDC Error Identifier value provided by the user*. **references an FFDC Error Stack that does not exist on this node. The FFDC Error Stack may have existed at one time, but the stack may have been removed since the time that the FFDC Error Identifier was created. fcstkrpt cannot provide any information on the incident associated with this FFDC Error Identifier.**

Explanation: The FFDC Error Identifier does not reference an FFDC Error Stack that is currently present on the node. The associated FFDC Error Stack was most likely removed by the fcstclear command.

User Response: Contact the system administrator to determine if a backup exists of the FFDC Error Stack directory, ask for the backup to be restored, and retry the command once the backup is restored. If no backup exists, nothing can be done to obtain incident information for this FFDC Error Identifier. In the future, obtain the incident information as soon as an FFDC Error Identifier is reported, and preserve this information for later problem determination efforts.

2615-716 fcstkrpt Error: **This command requires that either the *The set of required options to the fcstkrpt command* option or the *The set of required options to the fcstkrpt command* option be specified. Please correct the usage of this command and execute the command again with the proper options.**

Explanation: The caller did not provide one of the required options to this command. This message is followed by a usage message for the fcstkrpt command.

User Response: Provide one of the required options for the command and retry the fcstkrpt command. For assistance on using the fcstkrpt command, use the '-h' option to obtain assistance information for the command.

2615-717 fcstkrpt Error: The FFDC Error Identifier *FFDC Error Identifier value* **references an FFDC Error Stack that was created on a remote system. fcstkrpt cannot provide any information on the incident associated with this FFDC Error Identifier. Use the 'fcdecode' command to determine where the FFDC Error Stack for this identifier was created.**

Explanation: The FFDC Error Identifier references an FFDC Error Stack that was created on a remote system. The FFDC Error Stack for this identifier does not exist on this node, therefore 'fcstkrpt' cannot obtain

the FFDC Error Stack information for this identifier. The user must go to the remote system where the FFDC Error Identifier was generated to obtain the report associated with this identifier.

User Response: Use the 'fcdecode' command to determine on which remote system this FFDC Error Identifier was generated. Go to that system and issue the 'fcstkprpt' command on that node to obtain the report associated with this identifier.

2615-718 **fcstkprpt Error: Unable to lock the FFDC Error Stack file for exclusive use. The file may have been removed by another application, or another application may have locked this file for use and become hung. Unable to generate a report for this FFDC Error Stack file at this time.**

Explanation: Attempts to lock this file for exclusive use were not successful. The file may have been removed by another user or application, or another application also using this FFDC Error Stack file may have become hung while attempting to record information to the FFDC Error Stack.

User Response: Attempt to issue this report command again at a later time. If this condition persists, verify that the file exists and was not removed by another user or application.

2615-719 **fcstkprpt Error: The FFDC Error Stack file appears to be corrupted. Ending the report of the FFDC Error Stack file contents at this point.**

Explanation: The 'fcstkprpt' command detected at least one corrupted record within the FFDC Error Stack file, or found the file control information to be corrupted. Consider the FFDC Error Stack file unusable for recording further incident information.

User Response: Do not attempt to use this FFDC Error Stack in future incident recordings.

2615-720 **fcstkprpt Error: The FFDC Error Identifier *FFDC Error Identifier value* references an FFDC Error Stack that cannot be accessed by this user. The FFDC Error Stack file may have been created by another system user, and the user running the fcstkprpt command does not have sufficient system privileges to access files created by that user. To obtain information for this FFDC Failure Identifier, have a more privileged system user attempt this command. fcstkprpt cannot provide any information on the incident associated with this FFDC Error Identifier.**

Explanation: The user has specified an FFDC Failure Identifier that is stored in an FFDC Error Stack file created by another user. The fcstkprpt user does not have sufficient privilege to access this file, probably because the user that created the FFDC Error Stack file has greater system authority than the user that issued the command.

User Response: Have a more privileged system user attempt this command.

2615-721 **fcstkprpt Error: The FFDC Error Stack file *FFDC Error Stack file name value* cannot be accessed by this user. The FFDC Error Stack file may have been created by another system user, and the user running the fcstkprpt command does not have sufficient system privileges to access files created by that user. To obtain information for this FFDC Failure Identifier, have a more privileged system user attempt this command. fcstkprpt cannot provide any information on the incident associated with this FFDC Error Identifier.**

Explanation: The user has specified an FFDC Error Stack file created by another system user. The user that issued fcstkprpt does not have sufficient privilege to access this file, most likely because the user that created the FFDC Error Stack file has greater system authority than the user issuing the command.

User Response: Have a more privileged system user attempt this command.

2615-751 **fcdecode Error: Unknown option specified *-option*.**

Explanation: The user specified an option that was not valid to this command. The command did not attempt to obtain information from any FFDC Error Identifiers that were specified by the user of the command.

User Response: This message is accompanied by a

syntax message for the `fcdecode` command. Select the appropriate option and attempt the command again.

2615-752 **fcdecode Error: No options or First Failure Data Capture failure identifiers were provided as arguments to this command.**

Explanation: This command expects one or more FFDC Error Identifiers to be provided as arguments, or for the '-h' option to be specified as an option. The command was issued without either item of information. `fcdecode` ceased functioning and exited.

User Response: This message is accompanied by a syntax message for the `fcdecode` command. Select the appropriate option and attempt the command again.

2615-753 **fcdecode Error: More than one list of First Failure Data Capture failure identifiers was detected. This command accepts a single list of FFDC failure identifiers only. The initial list of FFDC failure identifiers detected was: *The initial list of FFDC Error Identifiers detected by fcdecode in the command's argument list.* The next list of failure identifiers detected was: *the next list of FFDC Error Identifiers detected by the command.* The most common reason for this failure is that white space is accidentally inserted by the command user between the FFDC failure identifiers. When specifying more than one FFDC failure identifier to `fcdecode`, these identifiers must be presented in a single list, separated by commas with no white space between the failure identifiers and the commas. Please correct the usage of this command and issue the command again.**

Explanation: Multiple lists of FFDC Error Identifiers were provided as arguments to this command, or multiple FFDC Error Identifiers were not correctly specified by the user. The message explains the most common reasons for why the command fails in this fashion.

User Response: Repair the usage of the command and issue the command again.

2615-754 **fcdecode Error: The following identifier is not in the proper format for a First Failure Data Capture failure Identifier: *identifier.* Verify that the identifier was entered correctly, and includes all characters from the First Failure Data Capture failure identifier. Issue this command again with the correct identifier.**

Explanation: `fcdecode` determined that the identifier provided by the user was not a valid First Failure Data Capture failure identifier. The command ensures that the identifier is of a specific length, and this identifier is not of the proper length. The user probably entered the identifier incorrectly, or did not provide all characters of the identifier.

User Response: Verify that the failure identifier was entered correctly, and reissue the command with the correct failure identifier.

2615-755 **fcdecode Error: The following First Failure Data Capture failure identifier cannot be interpreted: *identifier that cannot be interpreted.* This identifier was generated by an application using a later version of the First Failure Data Capture utilities, which has revised the format used within First Failure Data Capture failure identifiers. This `fcdecode` command is part of an earlier version of the First Failure Data Capture utilities, and cannot interpret the revised format of the identifier. Contact the system administrator to ensure that all systems within this operational cluster are executing the same version of the RSCT Cluster Utilities software.**

Explanation: `fcdecode` discovered that the identifier was generated by a more recent version of First Failure Data Capture than the version containing this `fcdecode` command. The identifier contained information which the `fcdecode` command could not interpret. If this failure identifier was generated from a different operational cluster than the one containing the local system, it is impossible to determine where the failure report resides.

User Response: Contact the system administrator and ask the administrator to investigate whether all nodes in the local node's operational cluster are all using the same level of RSCT Cluster Utilities software. If all nodes are executing at the same level, then the only conclusion to make is that the identifier was generated on a system outside the current operational cluster, and the failure cannot be traced to its location.

2615-756 **fcdecode Error: An unexpected failure occurred within the routine: *internal routine name*. Failure code from this routine: *failure code*. This error indicates a problem with the First Failure Data software. IBM Customer Support should be informed of this failure, and the IBM Customer Support Center should be contacted to resolve this problem. Make a note of the command that was issued, including all options and arguments, and save a copy of this command's output. Contact the system administrator to report this problem to the IBM Customer Support Center. System administrators should consult the problem determination information for First Failure Data Capture for assistance on reporting this failure.**

Explanation: fcdecode encountered an unexpected error which indicates a problem within the FFDC source code itself, and must be reported to the IBM Customer Support Center for resolution.

User Response: Follow the instructions listed in the message.

2615-801 **fcpushstk Error: Unknown option specified -*option*.**

Explanation: The user specified an option that was not valid to this command. This command did not attempt to record information to the FFDC Error Stack System Log.

User Response: This message is accompanied by a syntax message for the fcpushstk command. Select the appropriate option and enter the command again.

2615-802 **fcpushstk Error: The *option* option has been specified more than once. The option can only be specified once to this command. Information will not be recorded to the FFDC Error Stack.**

Explanation: The user provided multiple instances of the same option to this command. This command permits options to be used only once. Information will not be recorded to the FFDC Error Stack.

User Response: Correct the usage of this command in order for it to successfully record information to the FFDC Error Stack. Correct the command to specify options only once. Reissue the command after correcting its usage.

2615-803 **fcpushstk Warning: The caller provided both a detail data field using the -d option, and a detail data file using the -f option. This command only accepts either a detail data field or a detail data file. The detail data file will be ignored in the record generated by this routine, and the detail data field will be used instead.**

Explanation: The caller specified both the -d and the -f options to this command. One or the other must be specified.

User Response: Revise the command to provide either a detail data field or a detail data file, not both.

2615-804 **fcpushstk Warning: Incomplete information was provided for the source code file reporting this incident. Default information was provided by this command on behalf of the user.**

Explanation: This command accepts a licensed program product name, a source code file name, a line of code position, and a file version number as input parameters. At least one of these items was not provided, or was not a valid value. This command provided default information in place of the missing information, but that information is of little use to potential problem investigators.

User Response: Revise the command to provide more complete detecting file information, for assisting future problem determination efforts.

2615-805 **fcpushstk Error: An FFDC Environment does not exist for this process. This process cannot record failure information to the FFDC Error Stack.**

Explanation: An FFDC Error Stack was not established by this process. The process is not able to record information to an FFDC Error Stack.

User Response: If the process wishes to make use of an FFDC Error Stack for recording information about potential problems it detects, the process must establish an FFDC Environment using the fcinitstk command. All uses of the fcpushstk command will not succeed until an FFDC Environment is established.

2615-806 **fcpushstk Error: The FFDC Environment for this process appears to be corrupted. The FFDC Environment is not usable. This process cannot make recordings to the FFDC Error Stack.**

Explanation: The FFDC Environment is composed of four process environment variables: FFDCSTACK, FFDCTRACE, FFDCORIG, and FFDCPID. At least one

of these variables have been removed or set to a null value. The FFDC Environment is not usable in this condition, meaning that the process cannot establish an FFDC Environment or record information to the FFDC Error Stack.

User Response: Do not attempt to record information to the FFDC Error Stack within this process. Check the system for users or processes that may be setting one or more of these process environment variables. Check the application source code for instructions that set the process environment, and ensure that they are not setting one or more of these process environment variables. Consult the FFDC documentation for further problem determination procedures.

2615-807 **fcpushstk Error: The FFDC Error Stack directory** *directory name where FFDC Error Stacks reside on this system* **cannot be accessed. The directory may be missing, unmounted, or permissions may have been changed on the directory. Contact the system administrator and report this problem. This process cannot make recordings to the FFDC Error Stack at this time.**

Explanation: The named directory cannot be accessed. Either the directory has been removed, unmounted, or its permissions have been changed from the values created when the FFDC software was installed.

User Response: Do not attempt to record information to the FFDC Error Stack within this process. Contact the system administrator and report the problem. The system administrator should mount the directory if it is unmounted, create the directory if it has been removed, or change the permissions on the directory to permit access. Instructions for performing these repairs are given in the FFDC problem determination documentation.

2615-808 **fcpushstk Error: The FFDC Error Stack file** *filename of the FFDC Error Stack file* **cannot be accessed. The file may be missing, corrupted, or, permissions may have been changed on the file to prohibit access. Contact the system administrator and report this problem. This process cannot make recordings to the FFDC Error Stack at this time.**

Explanation: The named file cannot be accessed. Either the file has been removed, unmounted, or its permissions have been changed from the values that were used by the FFDC utilities when the file was created.

User Response: Do not attempt to record information to the FFDC Error Stack within this process. Contact the system administrator and report the problem. The

system administrator should verify the permissions on the file and correct them if necessary. Instructions for performing these repairs are given in the FFDC problem determination documentation.

2615-809 **fcpushstk Error: The name reserved for the FFDC Error Stack file:** *filename of the FFDC Error Stack file* **is the name of an existing directory. Ask the system administrator to ensure that system users are not creating subdirectories in the directory:** *name of the directory where FFDC Error Stacks reside on this system.* **If this problem occurs in multiple applications, contact the system administrator to have the problem reported to the IBM Customer Support Center.**

Explanation: The file name reserved for the FFDC Error Stack file for this application is actually the name of a directory. Either a system user has created a directory of the same name on the system, or there is a problem with the fcpushstk command that must be addressed by the IBM Customer Support Center.

User Response: Contact the system administrator and report the problem. System administrators should verify that the file name is indeed a directory, and was not created by a system user accidentally. Remove any subdirectories in the named directory to prevent the problem from reoccurring. If the problem is reported for multiple applications, the problem indicates a flaw in the fcpushstk command or the FFDC library routines, and the IBM Customer Support Center should be contacted to resolve the problem.

2615-810 **fcpushstk Error: An unexpected failure occurred in the routine** *routine name.* **This process cannot make recordings to the FFDC Error Stack at this time. Contact the IBM Customer Support Center for assistance in resolving this failure.**

Explanation: An error occurred within the FFDC software that was not expected. This error implies that a coding mistake was made in the FFDC software.

User Response: Do not attempt to record information to the FFDC Error Stack within this process. Report this problem to your system administrator, and have the system administrator contact the IBM Support Center. System administrators should consult the FFDC problem determination documentation prior to contacting the IBM Customer Support Center.

2615-811 **fcpushstk Error: Insufficient space exists in the file system containing the directory:** *name of the directory where FFDC Error Stacks reside on this system* to create an FFDC Error Stack file. This routine attempted to reserve and FFDC Error Stack file of a minimum size of *minimum size of an FFDC Error Stack file* bytes while leaving at least five percent of the file system capacity available. The FFDC Error Stack file could not be reserved under these constraints. This application cannot record information to the FFDC Error Stack until more space becomes available in the file system.

Explanation: An FFDC Error Stack cannot be created in the named directory within the safety precautions used by First Failure Data Capture. This application will be unable to record information to the FFDC Error Stack until more space becomes available in the file system containing this directory.

User Response: Report this problem to the system administrator. System administrators should remove any FFDC Error Stack files from this system that are no longer needed, using the `fcstkclear` command to remove them, or add more space to the file system containing this directory. System administrators should also check other directories in the same file system for other obsolete files that can be removed to make more space available.

2615-812 **fcpushstk Error: Unable to lock the file:** *name of the FFDC Error Stack file* for exclusive use. The file may have been removed by another application, or another application may have locked this file for use and become hung. This process is unable to record incident information to the FFDC Error Stack at this time.

Explanation: Attempts to lock this file for exclusive use were not successful. The file may have been removed by another user or application, or another application also using this FFDC Error Stack file may have become hung while attempting to record information to the FFDC Error Stack. This report cannot be made to the FFDC Error Stack at this time. The application can attempt to make this report again at a later time.

User Response: Attempt to record the same information to the FFDC Error Stack again. If this condition persists, verify that the file exists and was not removed by another user or application. Check any ancestor or descendent processes of this application to determine if any of them have become hung. Terminate any ancestor or descendent process that have hung.

2615-813 **fcpushstk Error: Unable to update the control information within the FFDC Error Stack file:** *name of the FFDC Error Stack file*. The file may be corrupted, or this problem can indicate an internal error in the FFDC software. This process should consider the FFDC Error Stack unusable. Verify that the file exists and can be viewed using the `fcstkprpt` command. If `fcstkprpt` fails to indicate an error with the FFDC Error Stack file, contact the system administrator to have this problem reported to the IBM Customer Service Center. Error code from the `write()` library call: *errno* from `write()` routine .

Explanation: The FFDC Error Stack internal control information could not be modified. The file may be corrupt, the file system may be experiencing problems, or an internal problem may exist in the FFDC library that requires the attention of the IBM Customer Support Center to rectify. The application should consider the FFDC Error Stack unusable.

User Response: Do not attempt to record further information to the FFDC Error Stack from this application. Contact the system administrator and report this problem. System administrators should verify that the file system containing this file is not experiencing problems, and should also verify that the hardware supporting this file system is also not experiencing problems. If the file system and the hardware appear to be functioning properly, contact the IBM Customer Support Center to report this problem.

2615-814 **fcpushstk Warning: No resource name was provided to this command through the *Option* to the `fcpushstk` command option. No resource name was recorded to the FFDC Error Stack report generated by this routine, but the incident was recorded.**

Explanation: The caller did not provide the option to indicate a resource name to this command. As a result, no resource name was recorded in the FFDC Error Stack entry, but the entry was made.

User Response: Repair the source code to pass a meaningful resource name to this routine, for assisting future problem determination efforts.

2615-815 **fcpushstk Warning: This command was not provided with a default message to use as a description for this incident in cases where the cataloged message description cannot be obtained. If the FFDC Error Stack contents are viewed on a node where the message does not exist, or if the FFDC Error Stack contents are viewed when the message catalog is missing, no description information will be presented for this entry. Users may not be able to understand the nature of the incident without a description, which limits the usefulness of the entry to the system administrator.**

Explanation: A default message to describe this incident was not provided to this command, or the message that was provided is not valid. However, a cataloged message was provided to this command. Without the default message information, no description information will be presented when viewing the FFDC Error Stack when the message catalog cannot be located or used. This can make the report unusable in these cases to any user that does not understand the internal implementation of the software, making the report a generic incident report of little value.

User Response: Supply description information, otherwise the information becomes useless to users who do not understand the implementation of the software. Verify that the default message text was provided as a parameter to this routine. Correct the application to provide this information.

2615-816 **fcpushstk Warning: This command was not provided with complete information to identify a cataloged message that describes the nature of the incident being recorded to the FFDC Error Stack. As a result, this description will be presented using the default message text. Users viewing the FFDC Error Stack contents in other locales will not be able to translate this message into the language used for that locale, meaning that the user may not be able to understand the information recorded by this application.**

Explanation: The information provided for the message catalog, message set within the catalog, and message number within the set, are either missing or are incomplete. This routine could not record message catalog information to the FFDC Error Stack. As a result, end-users using languages other than that used by the application programmer may not be able to understand the information being recorded in the FFDC Error Stack for this incident.

User Response: Supply message catalog information, otherwise the information becomes useless to users who do not understand the application programmer's language. Verify that the message catalog name, the message set number, and the message number were provided as parameters to this routine. Correct the application to provide this information.

2615-817 **fcpushstk Warning: The caller provided both a detail data field and a detail data file to this command. This command only accepts either a detail data field or a detail data file. The detail data file was ignored in the record generated by this command, and the detail data field was used instead.**

Explanation: The caller specified both a detail data string on the incident being reported, and a file containing details on the incident. fcpushstk could not determine if the file of information was more important to the report than the detail data string. Since fcpushstk would record a file name in the place of a detail data string, the command decided that the detail data string should not be lost and used the detail data string in the FFDC Error Stack instead of the file.

User Response: Repair the source code to provide either a detail data field or a detail data file this routine, not both.

2615-818 **fcpushstk Warning: The command was not able to generate a unique FFDC Error Identifier for the report it filed. The report has been made successfully.**

Explanation: The FFDC software was unable to generate a valid FFDC Error Identifier for this incident report. An FFDC Error Identifier was not recorded as part of the report, and the caller does not have an FFDC Error Identifier provided to its client to indicate where the report was made. However, a report of the incident was made to the FFDC Error Stack, and can be observed later using the fcstkpr command on the FFDC Error Stack.

User Response: If this problem occurs more than once, it may indicate a possible problem within the FFDC software itself. Report multiple instances of this warning to the system administrator. System administrators should contact the IBM Customer Support Center for assistance on resolving this potential problem.

2615-819 **fcpushstk Warning: The command was unable to copy the contents of the file: name of detail data file provided in the -f option to the directory: directory where this routine intended to copy the file. Do not discard the original copy of this file. The name of the original copy of has been recorded to the FFDC Error Stack, and future problem determination efforts may require access to this file.**

Explanation: The command could not make a copy of the file. The directory may not be available or accessible, space may be insufficient to store the file in the directory, or the command may not be able to access the original.

User Response: Retain the original copy of the file for later use in resolving the incident being reported. In the future, ensure that the detail data file can be read by processes of this user. If conditions persist, contact the system administrator and report the problem. System administrators should consult the FFDC problem determination documentation for assistance in resolving this problem.

2615-820 **fcpushstk Warning: The information submitted for recording to the FFDC Error Stack exceeds the record length of FFDC Error Stack record size maximum length bytes. The information has been truncated to fit the record size limit. Some important diagnostic information may have been lost in the process. Errors may occur when this information is displayed later with the fcstkrpt command. This record size limit includes the detecting file information, the message catalog information, the description message, substitutional parameters to the description message, as well as the details of the condition. In the future, record less information, or make use of the detail data file option.**

Explanation: The FFDC client attempted to record too much information to a single record in the FFDC Error Stack. The FFDC code truncated this information to fit the record size length.

User Response: Modify the use of fcpushstk routine to use less information, or to make use of the detail data file option to move information out of the error stack entry and into another location.

2615-821 **fcpushstk Warning: No details about the incident were provided to this command, either in the form of a detail data string or in the form of a detail data file. The detailed data section of this report was omitted, but the report was successfully recorded. Records that do not contain details can be difficult to understand or use effectively in problem analysis efforts. Ensure that the command was used correctly, and that the '-d' or '-f' options were not accidentally omitted.**

Explanation: The fcstkrpt command was issued without the -d and -f options. No details about the incident were provided to the command. fcstkrpt successfully recorded an FFDC Error Stack record for the incident, but the detailed data section of this record will be blank when examined later. Without details, the incident report can only indicate that a specific incident occurred, and cannot provide any reasons why the incident occurred, limiting the usefulness of the entry.

User Response: Ensure that the fcstkrpt command was used correctly. Correct the command to include details about the failure with either the -d or the -f options.

2615-851 **fcreport Error: Unknown option specified - option.**

Explanation: The user specified an option that was not valid to this command. The command did not attempt to build a report.

User Response: This message is accompanied by a syntax message for the fcreport command. Select the appropriate option and enter the command again.

2615-852 **fcreport Error: The following identifier is not in the proper format for a First Failure Data Capture failure Identifier: identifier. Verify that the identifier was entered correctly, and includes all characters from the First Failure Data Capture failure identifier. Issue this command once again with the correct identifier.**

Explanation: fcdecode determined that the identifier provided by the user of the command was not a valid First Failure Data Capture failure identifier. The command ensures that the identifier is of a specific length, and this identifier is not of the proper length. The user probably entered the identifier incorrectly, or did not provide all characters of the identifier.

User Response: Verify that the failure identifier was entered correctly, and reissue the command with the correct failure identifier.

2615-853 **fcreport Error: The following First Failure Data Capture failure identifier cannot be interpreted: *identifier that cannot be interpreted*. This identifier was generated by an application using a later version of the First Failure Data Capture utilities, which has revised the format used within First Failure Data Capture failure identifiers. This `fcdecode` command is part of an earlier version of the First Failure Data Capture utilities, and cannot interpret the revised format of the identifier. Contact the system administrator to ensure that all systems within this operational cluster are executing the same version of the RSCT Cluster Utilities software.**

Explanation: `fcdecode` discovered that the identifier was generated by a more recent version of First Failure Data Capture than the version containing this `fcdecode` command. The identifier contained information which the `fcdecode` command could not interpret. If this failure identifier was generated from a different operational cluster than the one containing the local system, it is impossible to determine where the failure report resides.

User Response: Contact the system administrator and ask the administrator to investigate whether all nodes in the local node's operational cluster are all using the same level of RSCT Cluster Utilities software. If all nodes are executing at the same level, then the only conclusion to make is that the identifier was generated on a system outside the current operational cluster, and the failure cannot be traced to its location.

2615-854 **fcreport Error: The First Failure Data Capture failure identifier: *FFDC Failure Identifier name* was recorded to an FFDC Error Stack file which no longer exists on this system. This command is unable to generate a report for this FFDC Failure Identifier. The FFDC Error Stack file was most likely removed as part of normal system maintenance. In the future, try to obtain information on FFDC Failure Identifiers in a more timely fashion.**

Explanation: The FFDC Error Stack file containing this FFDC Failure Identifier no longer exists on the local system. The file was most likely removed as part of routine system maintenance, which includes the removal of old files and old FFDC Error Stacks using the `'fcclear'` command.

User Response: In the future, obtain information about FFDC Failure Identifiers in a more timely fashion, and ensure that needed FFDC Error Stacks are retained on the system.

2615-855 **fcreport Error: An unexpected failure occurred in the routine *internal routine name*. Unexpected return code: *return code*. This process cannot make recordings to the FFDC Error Stack at this time. Contact the IBM Customer Support Center for assistance in resolving this failure.**

Explanation: An error occurred within the FFDC software that was not expected. This error implies that a coding mistake was made in the FFDC software.

User Response: Report this problem to your system administrator, and have the system administrator contact the IBM Support Center. System administrators should consult the FFDC problem determination documentation prior to contacting the IBM Customer Support Center.

2615-856 **fcreport Status: The following First Failure Data Capture Failure Identifier: *First Failure Data Capture Failure Identifier provided to the fcreport command* was generated on a node using the following Internet address: *internet address obtained from the FFDC Failure Identifier*. This node cannot be contacted to obtain the information recorded for the specified identifier. The node may not be active at this moment, or the Internet address may no longer be valid. Another possible reason for this condition is that the First Failure Data Capture Failure Identifier may not have been stated correctly to this command. A typographical error in specifying this value may cause the `'fcreport'` command to incorrectly interpret the identifier, causing it to yield an erroneous Internet address. Verify that the First Failure Data Capture Failure Identifier was specified correctly.**

Explanation: The Internet address obtained from the FFDC Failure Identifier cannot be reached by the `'fcreport'` command. Because the node cannot be reached, information for this FFDC Failure Identifier cannot be obtained for this report. The `'fcreport'` command ends its processing at this point.

User Response: Follow the instructions listed in this message.

2615-857 **fcreport Error: Failure in allocating memory. This command cannot obtain further information on incidents related to the First Failure Data Capture Failure Identifier: *First Failure Data Capture Failure Identifier* provided to the *fcreport* command. If this command is being invoked from a script, check the script source code for possible memory leaks. Also check the system for processes hoarding memory, and report this problem to the system administrator.**

Explanation: Attempts to allocate memory for expanding the process environment were unsuccessful. This implies that either the process is allocating too much memory, or a process elsewhere in the system is allocating too much memory.

User Response: Consult the message text for instructions.

2615-901 **fccheck Error: Unknown command option *option*.**

Explanation: The listed option is not an option recognized by the 'fccheck' command. This message is followed by a usage message for the command which lists the valid command options.

User Response: Enter the command again using the correct options.

2615-902 **fccheck Error: An unexpected internal failure was detected within the 'fccheck' command while it was executing its tests. This failure must be reported to the IBM Customer Support Center for resolution. Please make a note of the conditions present on the local system at the time this failure occurred, make a permanent record of the output generated by this command, and contact the system administrator to have the problem reported to the IBM Customer Support Center. System administrator should consult the FFDC problem determination documentation for assistance on reporting this problem to the IBM Customer Support Center. Name of internal routine: *fccheck* internal routine Return code from internal routine: *return code*.**

Explanation: 'fccheck' encountered an unexpected error while performing its tests. This error indicates a problem in the actual implementation of the 'fccheck' command, and must be reported to the IBM Customer Support Center to be resolved.

User Response: Follow the instructions listed in this message.

2615-903 **fccheck Error: This command is not able to query the operating system for information about the file system containing the following directory: *directory*. This information should be available for a local file system through the operating system. If this directory resides on a local file system, this may indicate a problem with the operating system. Contact the system administrator and report this problem.**

Explanation: 'fccheck' is unable to query the operating system for information on the file system containing this directory. The 'statfs()' C library routine did not return any information on this file system. This error may indicate a problem with the operating system.

User Response: Report the problem to the system administrator. System administrators should perform problem determination procedures on the operating system to check for any errors. Contact the IBM Customer Support Center if the situation cannot be resolved.

2615-951 **fcclear Error: At least one option is required to execute this command.**

Explanation: The user did not specify at least one of the required options for this command. This message is followed by the usage message for the fcclear command.

User Response: Enter the command again using the required options.

2615-952 **fcclear Error: Unknown command option *option*.**

Explanation: The fcclear command does not recognize the listed option. This message is followed by a usage message for the command which lists the valid command options.

User Response: Enter the command again using the correct options.

2615-953 **fcclear Error: The *option* option has been specified more than once.**

Explanation: The user specified the same option more than once to this command. The command considers this a fatal error and has ceased its processing. This message is followed by the usage for the fcclear command.

User Response: Enter the command again using the correct options.

2615-954 **fcclear Error: A numeric value is required as an argument to the *option* option**

Explanation: A numeric value is required as an argument to this option. The user specified a value that was not numeric. The command has terminated without removing any files from the system. This message is followed by a usage message for the command which lists the valid command options.

User Response: Enter the command again using the correct options.

2615-955 **fcclear Error: The options provided to this command have specified that the same file should be retained and removed from the system. To avoid taking the wrong action, 'fcclear' will terminate without removing any files from the local system. The most common source of this failure is that the command user provided an FFDC Failure Identifier that is stored in an FFDC Error Stack file that was specified by another FFDC Failure Identifier, or referenced by name to another command option. Test all FFDC Failure Identifiers with the 'fcdecode' command. Ensure that none of the FFDC Failure Identifiers to be removed are stored in FFDC Error Stack files that are to be retained, and vice versa. Also ensure that all of the file names or FFDC Failure Identifiers provided as command arguments are entered correctly.**

Explanation: Options to the command have specified the same file name to be removed and retained. To avoid taking the wrong action, the command terminated without removing any files.

User Response: Follow the instructions listed in this message.

2615-956 **fcclear Error: An unexpected internal failure was detected within the 'fcclear' command. This failure must be reported to the IBM Customer Support Center for resolution. Please make a note of the conditions present on the local system at the time this failure occurred, make a permanent record of the output generated by this command, and contact the system administrator to have the problem reported to the IBM Customer Support Center. System administrator should consult the FFDC problem determination documentation for assistance on reporting this problem to the IBM Customer Support Center. Name of internal routine: *internal routine name*. Return code from internal routine: *return code*.**

Explanation: 'fcclear' encountered an unexpected error which indicates a problem in the actual implementation of the 'fcclear' command, and must be reported to the IBM Customer Support Center to be resolved.

User Response: Follow the instructions listed in this message.

2615-957 **fcclear Error: The following directory has been detected to be in error: *directory name*. The directory either does not have the correct access permissions, been removed, or references a file instead of a directory. This command is unable to remove any files from this directory. Contact the system administrator to report this problem. System administrators should verify that the file system containing this directory is mounted and that the directory does exist. If the directory does exist, ensure that the permissions on the directory are set to permission code 1777, and any further security on this directory permits access to the directory for normal users. Normal users should have sufficient permissions granted to allow them to remove First Failure Data Capture Error Stacks and detail data files that they have created in this directory.**

Explanation: 'fcclear' is unable to access the named directory. The directory may be unmounted, may have been removed, or permissions may have been altered to prohibit access to this directory. The 'fcclear' command is unable to remove any files from this directory.

User Response:

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Department LJEB/P905
2455 South Road

Poughkeepsie, NY 12601-5400
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Trademarks

The following terms are trademarks of the International Business Machines Corporation in the United States or other countries or both:

- AIX
- AIX/6000
- DATABASE 2
- DB2
- ES/9000
- ESCON
- HACMP/6000
- IBM
- IBMLink
- LoadLeveler
- Micro Channel
- Netfinity
- Netfinity Manager
- POWERparallel
- POWERserver
- RS/6000
- RS/6000 Scalable POWERparallel Systems
- Scalable POWERparallel Systems
- SP
- System/370
- System/390
- TURBOWAYS

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Microsoft, Windows, Windows NT, BackOffice, MS-DOS, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Tivoli Enterprise Console is a trademark of Tivoli Systems Inc. in the United States, other countries, or both.

UNIX is a registered trademark in the United States and other countries licensed exclusively through The Open Group.

Other company, product, and service names may be the trademarks or service marks of others.

Publicly Available Software

PSSP includes software that is publicly available:

expect

Programmed dialogue with interactive programs

Perl Practical Extraction and Report Language

SUP Software Update Protocol

Tcl Tool Command Language

TclX Tool Command Language Extended

Tk Tcl-based Tool Kit for X-windows

This book discusses the use of these products only as they apply specifically to the RS/6000 SP system. The distribution for these products includes the source code and associated documentation.

/usr/lpp/ssp/public contains the compressed **tar** files of the publicly available software. (IBM has made minor modifications to the versions of Tcl and Tk used in the SP system to improve their security characteristics. Therefore, the IBM-supplied versions do not match exactly the versions you may build from the compressed **tar** files.) All copyright notices in the documentation must be respected. You can find version and distribution information for each of these products that are part of your selected install options in the **/usr/lpp/ssp/README/ssp.public.README** file.

Software Update Protocol

IBM has provided modifications to this software. The resulting software is provided to you on an "AS IS" basis and WITHOUT WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Glossary of Terms and Abbreviations

A

ACL. Access Control List. A list that defines who has permission to access certain services; that is, for whom a server may perform certain tasks. This is usually a list of principals with the type of access assigned to each.

adapter. An adapter is a mechanism for attaching parts. For example, an adapter could be a part that electrically or physically connects a device to a computer or to another device. In the SP system, network connectivity is supplied by various adapters, some optional, that can provide connection to I/O devices, networks of workstations, and mainframe networks. Ethernet, FDDI, token-ring, HiPPI, SCSI, FCS, and ATM are examples of adapters that can be used as part of an SP system.

address. A character or group of characters that identifies a register, a device, a particular part of storage, or some other data source or destination.

AFS. A distributed file system that provides authentication services as part of its file system creation.

AIX. Abbreviation for Advanced Interactive Executive, IBM's licensed version of the UNIX operating system. AIX is particularly suited to support technical computing applications, including high function graphics and floating point computations.

Amd. Berkeley Software Distribution automount daemon.

API. Application Programming Interface. A set of programming functions and routines that provide access between the Application layer of the OSI seven-layer model and applications that want to use the network. It is a software interface.

application. The use to which a data processing system is put; for example, a payroll application, an airline reservation application.

application data. The data that is produced using an application program.

ARP. Address Resolution Protocol.

ATM. Asynchronous Transfer Mode. (See *TURBOWAYS 100 ATM Adapter*.)

authentication. The process of validating the identity of either a user of a service or the service itself. The process of a principal proving the authenticity of its identity.

authorization. The process of obtaining permission to access resources or perform tasks. In SP security services, authorization is based on the principal identifier. The granting of access rights to a principal.

authorization file. A type of ACL (access control list) used by the IBM AIX remote commands and the IBM PSSP Sysctl and Hardmon components.

B

batch processing. * (1) The processing of data or the accomplishment of jobs accumulated in advance in such a manner that each accumulation thus formed is processed or accomplished in the same run. * (2) The processing of data accumulating over a period of time. * (3) Loosely, the execution of computer programs serially. (4) Computer programs executed in the background.

BMCA. Block Multiplexer Channel Adapter. The block multiplexer channel connection allows the RS/6000 to communicate directly with a host System/370 or System/390; the host operating system views the system unit as a control unit.

BOS. The AIX Base Operating System.

C

call home function. The ability of a system to call the IBM support center and open a PMR to have a repair scheduled.

CDE. Common Desktop Environment. A graphical user interface for UNIX.

charge feature. An optional feature for either software or hardware for which there is a charge.

CLI. Command Line Interface.

client. * (1) A function that requests services from a server and makes them available to the user. * (2) A term used in an environment to identify a machine that uses the resources of the network.

Client Input/Output Sockets (CLIO/S). A software package that enables high-speed data and tape access between SP systems, AIX systems, and ES/9000 mainframes.

CLIO/S. Client Input/Output Sockets.

CMI. Centralized Management Interface provides a series of SMIT menus and dialogues used for defining and querying the SP system configuration.

Concurrent Virtual Shared Disk. A virtual shared disk that can be concurrently accessed by more than one server.

connectionless. A communication process that takes place without first establishing a connection.

connectionless network. A network in which the sending logical node must have the address of the receiving logical node before information interchange can begin. The packet is routed through nodes in the network based on the destination address in the packet. The originating source does not receive an acknowledgment that the packet was received at the destination.

control workstation. A single point of control allowing the administrator or operator to monitor and manage the SP system using the IBM AIX Parallel System Support Programs.

credentials. A protocol message, or part thereof, containing a ticket and an authenticator supplied by a client and used by a server to verify the client's identity.

css. Communication subsystem.

D

daemon. A process, not associated with a particular user, that performs system-wide functions such as administration and control of networks, execution of time-dependent activities, line printer spooling and so forth.

DASD. Direct Access Storage Device. Storage for input/output data.

DCE. Distributed Computing Environment.

DFS. distributed file system. A subset of the IBM Distributed Computing Environment.

DNS. Domain Name Service. A hierarchical name service which maps high level machine names to IP addresses.

E

Error Notification Object. An object in the SDR that is matched with an error log entry. When an error log entry occurs that matches the Notification Object, a user-specified action is taken.

ESCON. Enterprise Systems Connection. The ESCON channel connection allows the RS/6000 to communicate directly with a host System/390; the host operating system views the system unit as a control unit.

Ethernet. (1) Ethernet is the standard hardware for TCP/IP local area networks in the UNIX marketplace. It is a 10-megabit per second baseband type LAN that

allows multiple stations to access the transmission medium at will without prior coordination, avoids contention by using carrier sense and deference, and resolves contention by collision detection (CSMA/CD).

(2) A passive coaxial cable whose interconnections contain devices or components, or both, that are all active. It uses CSMA/CD technology to provide a best-effort delivery system.

Ethernet network. A baseband LAN with a bus topology in which messages are broadcast on a coaxial cabling using the carrier sense multiple access/collision detection (CSMA/CD) transmission method.

event. In Event Management, the notification that an expression evaluated to true. This evaluation occurs each time an instance of a resource variable is observed.

expect. Programmed dialogue with interactive programs.

expression. In Event Management, the relational expression between a resource variable and other elements (such as constants or the previous value of an instance of the variable) that, when true, generates an event. An example of an expression is $X < 10$ where X represents the resource variable `IBM.PSSP.aixos.PagSp.%totalfree` (the percentage of total free paging space). When the expression is true, that is, when the total free paging space is observed to be less than 10%, the Event Management subsystem generates an event to notify the appropriate application.

F

failover. Also called failover, the sequence of events when a primary or server machine fails and a secondary or backup machine assumes the primary workload. This is a disruptive failure with a short recovery time.

fall back. Also called fallback, the sequence of events when a primary or server machine takes back control of its workload from a secondary or backup machine.

FDDI. Fiber Distributed Data Interface.

FFDC. First Failure Data Capture.

Fiber Distributed Data Interface (FDDI). An American National Standards Institute (ANSI) standard for 100-megabit-per-second LAN using optical fiber cables. An FDDI local area network (LAN) can be up to 100 km (62 miles) and can include up to 500 system units. There can be up to 2 km (1.24 miles) between system units and concentrators.

file. * A set of related records treated as a unit, for example, in stock control, a file could consist of a set of invoices.

file name. A CMS file identifier in the form of 'filename filetype filemode' (like: TEXT DATA A).

file server. A centrally located computer that acts as a storehouse of data and applications for numerous users of a local area network.

File Transfer Protocol (FTP). The Internet protocol (and program) used to transfer files between hosts. It is an application layer protocol in TCP/IP that uses TELNET and TCP protocols to transfer bulk-data files between machines or hosts.

First Failure Data Capture (FFDC). A set of utilities used for recording persistent records of failures and significant software incidents. It provides a means of associating failures to one another, thus allowing software to link effects of a failure to their causes and thereby facilitating discovery of the root cause of a failure.

foreign host. Any host on the network other than the local host.

FTP. File transfer protocol.

G

gateway. An intelligent electronic device interconnecting dissimilar networks and providing protocol conversion for network compatibility. A gateway provides transparent access to dissimilar networks for nodes on either network. It operates at the session presentation and application layers.

H

HACMP. High Availability Cluster Multi-Processing for AIX.

HACWS. High Availability Control Workstation function, based on HACMP, provides for a backup control workstation for the SP system.

HAL. Hardware Abstraction Layer, a communication device interface that provides communication channels for processes.

Hashed Shared Disk (HSD). The data striping device for the IBM Virtual Shared Disk. The device driver lets application programs stripe data across physical disks in multiple IBM Virtual Shared Disks, thus reducing I/O bottlenecks.

help key. In the SP graphical interface, the key that gives you access to the SP graphical interface help facility.

High Availability Cluster Multi-Processing. An IBM facility to cluster nodes or components to provide high availability by eliminating single points of failure.

HiPPI. High Performance Parallel Interface. RS/6000 units can attach to a HiPPI network as defined by the ANSI specifications. The HiPPI channel supports burst rates of 100 Mbps over dual simplex cables; connections can be up to 25 km in length as defined by the standard and can be extended using third-party HiPPI switches and fiber optic extenders.

home directory. The directory associated with an individual user.

host. A computer connected to a network, and providing an access method to that network. A host provides end-user services.

I

instance vector. Obsolete term for resource identifier.

Intermediate Switch Board. Switches mounted in the switch expansion frame.

Internet. A specific inter-network consisting of large national backbone networks such as APARANET, MILNET, and NSFnet, and a myriad of regional and campus networks all over the world. The network uses the TCP/IP protocol suite.

Internet Protocol (IP). (1) A protocol that routes data through a network or interconnected networks. IP acts as an interface between the higher logical layers and the physical network. This protocol, however, does not provide error recovery, flow control, or guarantee the reliability of the physical network. IP is a connectionless protocol. (2) A protocol used to route data from its source to its destination in an Internet environment.

IP address. A 32-bit address assigned to devices or hosts in an IP internet that maps to a physical address. The IP address is composed of a network and host portion.

ISB. Intermediate Switch Board.

K

Kerberos. A service for authenticating users in a network environment.

kernel. The core portion of the UNIX operating system which controls the resources of the CPU and allocates them to the users. The kernel is memory-resident, is said to run in "kernel mode" and is protected from user tampering by the hardware.

Kernel Low-Level Application Programming Interface (KLAPI). KLAPI provides transport service for communication using the SP Switch.

L

LAN. (1) Acronym for Local Area Network, a data network located on the user's premises in which serial transmission is used for direct data communication among data stations. (2) Physical network technology that transfers data a high speed over short distances. (3) A network in which a set of devices is connected to another for communication and that can be connected to a larger network.

local host. The computer to which a user's terminal is directly connected.

log database. A persistent storage location for the logged information.

log event. The recording of an event.

log event type. A particular kind of log event that has a hierarchy associated with it.

logging. The writing of information to persistent storage for subsequent analysis by humans or programs.

M

mask. To use a pattern of characters to control retention or elimination of portions of another pattern of characters.

menu. A display of a list of available functions for selection by the user.

Motif. The graphical user interface for OSF, incorporating the X Window System. Also called OSF/Motif.

MTBF. Mean time between failure. This is a measure of reliability.

MTTR. Mean time to repair. This is a measure of serviceability.

N

naive application. An application with no knowledge of a server that fails over to another server. Client to server retry methods are used to reconnect.

network. An interconnected group of nodes, lines, and terminals. A network provides the ability to transmit data to and receive data from other systems and users.

NFS. Network File System. NFS allows different systems (UNIX or non-UNIX), different architectures, or vendors connected to the same network, to access remote files in a LAN environment as though they were local files.

NIM. Network Installation Management is provided with AIX to install AIX on the nodes.

NIM client. An AIX system installed and managed by a NIM master. NIM supports three types of clients:

- Standalone
- Diskless
- Dataless

NIM master. An AIX system that can install one or more NIM clients. An AIX system must be defined as a NIM master before defining any NIM clients on that system. A NIM master manages the configuration database containing the information for the NIM clients.

NIM object. A representation of information about the NIM environment. NIM stores this information as objects in the NIM database. The types of objects are:

- Network
- Machine
- Resource

NIS. Network Information System.

node. In a network, the point where one or more functional units interconnect transmission lines. A computer location defined in a network. The SP system can house several different types of nodes for both serial and parallel processing. These node types can include thin nodes, wide nodes, 604 high nodes, as well as other types of nodes both internal and external to the SP frame.

Node Switch Board. Switches mounted on frames that contain nodes.

NSB. Node Switch Board.

NTP. Network Time Protocol.

O

ODM. Object Data Manager. In AIX, a hierarchical object-oriented database for configuration data.

P

parallel environment. A system environment where message passing or SP resource manager services are used by the application.

Parallel Environment. A licensed IBM program used for message passing applications on the SP or RS/6000 platforms.

parallel processing. A multiprocessor architecture which allows processes to be allocated to tightly coupled multiple processors in a cooperative processing environment, allowing concurrent execution of tasks.

parameter. * (1) A variable that is given a constant value for a specified application and that may denote

the application. * (2) An item in a menu for which the operator specifies a value or for which the system provides a value when the menu is interpreted. * (3) A name in a procedure that is used to refer to an argument that is passed to the procedure. * (4) A particular piece of information that a system or application program needs to process a request.

partition. See system partition.

Perl. Practical Extraction and Report Language.

perspective. The primary window for each SP Perspectives application, so called because it provides a unique view of an SP system.

pipe. A UNIX utility allowing the output of one command to be the input of another. Represented by the | symbol. It is also referred to as filtering output.

PMR. Problem Management Report.

POE. Formerly Parallel Operating Environment, now Parallel Environment for AIX.

port. (1) An end point for communication between devices, generally referring to physical connection. (2) A 16-bit number identifying a particular TCP or UDP resource within a given TCP/IP node.

predicate. Obsolete term for expression.

Primary node or machine. (1) A device that runs a workload and has a standby device ready to assume the primary workload if that primary node fails or is taken out of service. (2) A node on the switch that initializes, provides diagnosis and recovery services, and performs other operations to the switch network. (3) In IBM Virtual Shared Disk function, when physical disks are connected to two nodes (twin-tailed), one node is designated as the primary node for each disk and the other is designated the secondary, or backup, node. The primary node is the server node for IBM Virtual Shared Disks defined on the physical disks under normal conditions. The secondary node can become the server node for the disks if the primary node is unavailable (off-line or down).

Problem Management Report. The number in the IBM support mechanism that represents a service incident with a customer.

process. * (1) A unique, finite course of events defined by its purpose or by its effect, achieved under defined conditions. * (2) Any operation or combination of operations on data. * (3) A function being performed or waiting to be performed. * (4) A program in operation. For example, a daemon is a system process that is always running on the system.

protocol. A set of semantic and syntactic rules that defines the behavior of functional units in achieving communication.

R

RAID. Redundant array of independent disks.

rearm expression. In Event Management, an expression used to generate an event that alternates with an original event expression in the following way: the event expression is used until it is true, then the rearm expression is used until it is true, then the event expression is used, and so on. The rearm expression is commonly the inverse of the event expression (for example, a resource variable is on or off). It can also be used with the event expression to define an upper and lower boundary for a condition of interest.

rearm predicate. Obsolete term for rearm expression.

remote host. See *foreign host*.

resource. In Event Management, an entity in the system that provides a set of services. Examples of resources include hardware entities such as processors, disk drives, memory, and adapters, and software entities such as database applications, processes, and file systems. Each resource in the system has one or more attributes that define the state of the resource.

resource identifier. In Event Management, a set of elements, where each element is a name/value pair of the form name=value, whose values uniquely identify the copy of the resource (and by extension, the copy of the resource variable) in the system.

resource monitor. A program that supplies information about resources in the system. It can be a command, a daemon, or part of an application or subsystem that manages any type of system resource.

resource variable. In Event Management, the representation of an attribute of a resource. An example of a resource variable is IBM.AIX.PagSp.%totalfree, which represents the percentage of total free paging space. IBM.AIX.PagSp specifies the resource name and %totalfree specifies the resource attribute.

RISC. Reduced Instruction Set Computing (RISC), the technology for today's high performance personal computers and workstations, was invented in 1975. Uses a small simplified set of frequently used instructions for rapid execution.

rlogin (remote LOGIN). A service offered by Berkeley UNIX systems that allows authorized users of one machine to connect to other UNIX systems across a network and interact as if their terminals were connected directly. The rlogin software passes information about the user's environment (for example, terminal type) to the remote machine.

RPC. Acronym for Remote Procedure Call, a facility that a client uses to have a server execute a procedure call. This facility is composed of a library of procedures plus an XDR.

RSH. A variant of RLOGIN command that invokes a command interpreter on a remote UNIX machine and passes the command line arguments to the command interpreter, skipping the LOGIN step completely. See also *rlogin*.

S

SCSI. Small Computer System Interface.

Secondary node. In IBM Virtual Shared Disk function, when physical disks are connected to two nodes (twin-tailed), one node is designated as the primary node for each disk and the other is designated as the secondary, or backup, node. The secondary node acts as the server node for the IBM Virtual Shared disks defined on the physical disks if the primary node is unavailable (off-line or down).

server. (1) A function that provides services for users. A machine may run client and server processes at the same time. (2) A machine that provides resources to the network. It provides a network service, such as disk storage and file transfer, or a program that uses such a service. (3) A device, program, or code module on a network dedicated to providing a specific service to a network. (4) On a LAN, a data station that provides facilities to other data stations. Examples are file server, print server, and mail server.

shell. The shell is the primary user interface for the UNIX operating system. It serves as command language interpreter, programming language, and allows foreground and background processing. There are three different implementations of the shell concept: Bourne, C and Korn.

Small Computer System Interface (SCSI). An input and output bus that provides a standard interface for the attachment of various direct access storage devices (DASD) and tape drives to the RS/6000.

Small Computer Systems Interface Adapter (SCSI Adapter). An adapter that supports the attachment of various direct-access storage devices (DASD) and tape drives to the RS/6000.

SMIT. The System Management Interface Toolkit is a set of menu driven utilities for AIX that provides functions such as transaction login, shell script creation, automatic updates of object database, and so forth.

SNMP. Simple Network Management Protocol. (1) An IP network management protocol that is used to monitor attached networks and routers. (2) A TCP/IP-based protocol for exchanging network management

information and outlining the structure for communications among network devices.

socket. (1) An abstraction used by Berkeley UNIX that allows an application to access TCP/IP protocol functions. (2) An IP address and port number pairing. (3) In TCP/IP, the Internet address of the host computer on which the application runs, and the port number it uses. A TCP/IP application is identified by its socket.

standby node or machine. A device that waits for a failure of a primary node in order to assume the identity of the primary node. The standby machine then runs the primary's workload until the primary is back in service.

subnet. Shortened form of subnetwork.

subnet mask. A bit template that identifies to the TCP/IP protocol code the bits of the host address that are to be used for routing for specific subnetworks.

subnetwork. Any group of nodes that have a set of common characteristics, such as the same network ID.

subsystem. A software component that is not usually associated with a user command. It is usually a daemon process. A subsystem will perform work or provide services on behalf of a user request or operating system request.

SUP. Software Update Protocol.

switch capsule. A group of SP frames consisting of a switched frame and its companion non-switched frames.

Sysctl. Secure System Command Execution Tool. An authenticated client/server system for running commands remotely and in parallel.

syslog. A BSD logging system used to collect and manage other subsystem's logging data.

System Administrator. The user who is responsible for setting up, modifying, and maintaining the SP system.

system partition. A group of nonoverlapping nodes on a switch chip boundary that act as a logical SP system.

T

tar. Tape ARchive, is a standard UNIX data archive utility for storing data on tape media.

TaskGuides. SP TaskGuides are a form of advanced online assistance designed to walk you through complex or infrequently performed tasks. Each TaskGuide does not simply list the required steps. It actually performs the steps for you, automating the steps to the highest degree possible and prompting you for input only when absolutely necessary. You might recognize them as *wizards*.

Tcl. Tool Command Language.

TclX. Tool Command Language Extended.

TCP. Acronym for Transmission Control Protocol, a stream communication protocol that includes error recovery and flow control.

TCP/IP. Acronym for Transmission Control Protocol/Internet Protocol, a suite of protocols designed to allow communication between networks regardless of the technologies implemented in each network. TCP provides a reliable host-to-host protocol between hosts in packet-switched communications networks and in interconnected systems of such networks. It assumes that the underlying protocol is the Internet Protocol.

Telnet. Terminal Emulation Protocol, a TCP/IP application protocol that allows interactive access to foreign hosts.

ticket. An encrypted protocol message used to securely pass the identity of a user from a client to a server.

Tk. Tcl-based Tool Kit for X Windows.

TMPCP. Tape Management Program Control Point.

token-ring. (1) Network technology that controls media access by passing a token (special packet or frame) between media-attached machines. (2) A network with a ring topology that passes tokens from one attaching device (node) to another. (3) The IBM Token-Ring LAN connection allows the RS/6000 system unit to participate in a LAN adhering to the IEEE 802.5 Token-Passing Ring standard or the ECMA standard 89 for Token-Ring, baseband LANs.

transaction. An exchange between the user and the system. Each activity the system performs for the user is considered a transaction.

transceiver (transmitter-receiver). A physical device that connects a host interface to a local area network, such as Ethernet. Ethernet transceivers contain electronics that apply signals to the cable and sense collisions.

transfer. To send data from one place and to receive the data at another place. Synonymous with move.

transmission. * The sending of data from one place for reception elsewhere.

TURBOWAYS 100 ATM Adapter. An IBM high-performance, high-function intelligent adapter that provides dedicated 100 Mbps ATM (asynchronous transfer mode) connection for high-performance servers and workstations.

U

UDP. User Datagram Protocol.

UNIX operating system. An operating system developed by Bell Laboratories that features multiprogramming in a multiuser environment. The UNIX operating system was originally developed for use on minicomputers, but has been adapted for mainframes and microcomputers. **Note:** The AIX operating system is IBM's implementation of the UNIX operating system.

user. Anyone who requires the services of a computing system.

User Datagram Protocol (UDP). (1) In TCP/IP, a packet-level protocol built directly on the Internet Protocol layer. UDP is used for application-to-application programs between TCP/IP host systems. (2) A transport protocol in the Internet suite of protocols that provides unreliable, connectionless datagram service. (3) The Internet Protocol that enables an application programmer on one machine or process to send a datagram to an application program on another machine or process.

user ID. A nonnegative integer, contained in an object of type *uid_t*, that is used to uniquely identify a system user.

V

Virtual Shared Disk, IBM. The function that allows application programs executing at different nodes of a system partition to access a raw logical volume as if it were local at each of the nodes. In actuality, the logical volume is local at only one of the nodes (the server node).

W

workstation. * (1) A configuration of input/output equipment at which an operator works. * (2) A terminal or microcomputer, usually one that is connected to a mainframe or to a network, at which a user can perform applications.

X

X Window System. A graphical user interface product.

Bibliography

This bibliography helps you find product documentation related to the RS/6000 SP hardware and software products.

You can find most of the IBM product information for RS/6000 SP products on the World Wide Web. Formats for both viewing and downloading are available.

PSSP documentation is shipped with the PSSP product in a variety of formats and can be installed on your system. The man pages for public code that PSSP includes are also available online.

You can order hard copies of the product documentation from IBM. This bibliography lists the titles that are available and their order numbers.

Finally, this bibliography contains a list of non-IBM publications that discuss parallel computing and other topics related to the RS/6000 SP.

Information Formats

Documentation supporting RS/6000 SP software licensed programs is no longer available from IBM in hardcopy format. However, you can view, search, and print documentation in the following ways:

- On the World Wide Web
- Online (from the product media or the SP Resource Center)

Finding Documentation on the World Wide Web

Most of the RS/6000 SP hardware and software books are available from the IBM RS/6000 Web site at:

<http://www.rs6000.ibm.com>

You can view a book or download a Portable Document Format (PDF) version of it. At the time this manual was published, the Web address of the "RS/6000 SP Product Documentation Library" page was:

http://www.rs6000.ibm.com/resource/aix_resource/sp_books

However, the structure of the RS/6000 Web site can change over time.

Accessing PSSP Documentation Online

On the same medium as the PSSP product code, IBM ships PSSP man pages, HTML files, and PDF files. In order to use these publications, you must first install the **ssp.docs** file set.

To view the PSSP HTML publications, you need access to an HTML document browser such as Netscape. The HTML files and an index that links to them are installed in the **/usr/lpp/ssp/html** directory. Once installed, you can also view the HTML files from the RS/6000 SP Resource Center.

If you have installed the SP Resource Center on your SP system, you can access it by entering the **/usr/lpp/ssp/bin/resource_center** command. If you have the SP Resource Center on CD-ROM, see the **readme.txt** file for information about how to run it.

To view the PSSP PDF publications, you need access to the Adobe Acrobat Reader. The Acrobat Reader is shipped with the AIX Version 4.3 Bonus Pack and is also freely available for downloading from the Adobe Web site at:

<http://www.adobe.com>

To successfully print a large PDF file (approximately 300 or more pages) from the Adobe Acrobat reader, you may need to select the "Download Fonts Once" button on the Print window.

Manual Pages for Public Code

The following manual pages for public code are available in this product:

SUP /usr/lpp/ssp/man/man1/sup.1

Perl (Version 4.036)

/usr/lpp/ssp/perl/man/perl.man

/usr/lpp/ssp/perl/man/h2ph.man

/usr/lpp/ssp/perl/man/s2p.man

/usr/lpp/ssp/perl/man/a2p.man

Manual pages and other documentation for **Tcl**, **TclX**, **Tk**, and **expect** can be found in the compressed **tar** files located in the **/usr/lpp/ssp/public** directory.

RS/6000 SP Planning Publications

This section lists the IBM product documentation for planning for the IBM RS/6000 SP hardware and software.

IBM RS/6000 SP:

- *Planning, Volume 1, Hardware and Physical Environment, GA22-7280*
- *Planning, Volume 2, Control Workstation and Software Environment, GA22-7281*

RS/6000 SP Hardware Publications

This section lists the IBM product documentation for the IBM RS/6000 SP hardware.

IBM RS/6000 SP:

- *Planning, Volume 1, Hardware and Physical Environment, GA22-7280*
- *Planning, Volume 2, Control Workstation and Software Environment, GA22-7281*
- *Installation and Relocation, GA22-7441*
- *System Service Guide, GA22-7442*
- *SP Switch Service Guide, GA22-7443*
- *SP Switch2 Service Guide, GA22-7444*
- *Uniprocessor Node Service Guide, GA22-7445*
- *604 and 604e SMP High Node Service Guide, GA22-7446*
- *SMP Thin and Wide Node Service Guide, GA22-7447*
- *POWER3 SMP High Node Service Guide, GA22-7448*

RS/6000 SP Switch Router Publications

The RS/6000 SP Switch Router is based on the Ascend GRF switched IP router product from Lucent Technologies. You can order the SP Switch Router as the IBM 9077.

The following publications are shipped with the SP Switch Router. You can also order these publications from IBM using the order numbers shown.

- *Ascend GRF GateD Manual, GA22-7327*
- *Ascend GRF 400/1600 Getting Started, GA22-7368*
- *Ascend GRF Configuration and Management, GA22-7366*
- *Ascend GRF Reference Guide, GA22-7367*
- *SP Switch Router Adapter Guide, GA22-7310*

RS/6000 SP Software Publications

This section lists the IBM product documentation for software products related to the IBM RS/6000 SP. These products include:

- IBM Parallel System Support Programs for AIX (PSSP)
- IBM LoadLeveler for AIX (LoadLeveler)
- IBM Parallel Environment for AIX (Parallel Environment)
- IBM General Parallel File System for AIX (GPFS)
- IBM Engineering and Scientific Subroutine Library (ESSL) for AIX
- IBM Parallel ESSL for AIX
- IBM High Availability Cluster Multi-Processing for AIX (HACMP)
- IBM Client Input Output/Sockets (CLIO/S)
- IBM Network Tape Access and Control System for AIX (NetTAPE)

PSSP Publications

IBM RS/6000 SP:

- *Planning, Volume 2, Control Workstation and Software Environment, GA22-7281*

PSSP:

- *Installation and Migration Guide, GA22-7347*
- *Administration Guide, SA22-7348*
- *Managing Shared Disks, SA22-7349*
- *Performance Monitoring Guide and Reference, SA22-7353*
- *Diagnosis Guide, GA22-7350*
- *Command and Technical Reference, SA22-7351*
- *Messages Reference, GA22-7352*

RS/6000 Cluster Technology (RSCT):

- *Event Management Programming Guide and Reference, SA22-7354*
- *Group Services Programming Guide and Reference, SA22-7355*
- *First Failure Data Capture Programming Guide and Reference, SA22-7454*

LoadLeveler Publications

LoadLeveler:

- *Using and Administering, SA22-7311*
- *Diagnosis and Messages Guide, GA22-7277*

GPFS Publications

GPFS:

- *Problem Determination Guide, GA22-7434*
- *Data Management API Guide, GA22-7435*
- *Guide and Reference, GA22-7452*

- *Installation and Tuning Guide*, GA22-7453

Parallel Environment Publications

Parallel Environment:

- *Installation Guide*, GA22-7418
- *Messages*, GA22-7419
- *DPCL Programming Guide*, SA22-7420
- *DPCL Class Reference*, SA22-7421
- *MPI Programming Guide*, SA22-7422
- *MPI Subroutine Reference*, SA22-7423
- *Hitchhiker's Guide*, SA22-7424
- *Operation and Use, Volume 1*, SA22-7425
- *Operation and Use, Volume 2*, SA22-7426
- *MPL Programming and Subroutine Reference*, GC23-3893

Parallel ESSL and ESSL Publications

- *ESSL Products: General Information*, GC23-0529
- *Parallel ESSL: Guide and Reference*, SA22-7273
- *ESSL: Guide and Reference*, SA22-7272

HACMP Publications

HACMP:

- *Concepts and Facilities*, SC23-4276
- *Planning Guide*, SC23-4277
- *Installation Guide*, SC23-4278
- *Administration Guide*, SC23-4279
- *Troubleshooting Guide*, SC23-4280
- *Programming Locking Applications*, SC23-4281
- *Programming Client Applications*, SC23-4282
- *Master Index and Glossary*, SC23-4285
- *HANFS for AIX Installation and Administration Guide*, SC23-4283
- *Enhanced Scalability Installation and Administration Guide*, SC23-4284

CLIO/S Publications

CLIO/S:

- *General Information*, GC23-3879
- *User's Guide and Reference*, GC28-1676

NetTAPE Publications

NetTAPE:

- *General Information*, GC23-3990
- *User's Guide and Reference*, available from your IBM representative

AIX and Related Product Publications

For the latest information on AIX and related products, including RS/6000 hardware products, see *AIX and Related Products Documentation Overview*, SC23-2456. You can order a hard copy of the book from IBM. You can also view it online from the “AIX Online Publications and Books” page of the RS/6000 Web site at:

http://www.rs6000.ibm.com/resource/aix_resource/Pubs

DCE Publications

The DCE library consists of the following books:

- *IBM DCE 3.1 for AIX: Administration Commands Reference*
- *IBM DCE 3.1 for AIX: Administration Guide—Introduction*
- *IBM DCE 3.1 for AIX: Administration Guide—Core Components*
- *IBM DCE 3.1 for AIX: DFS Administration Guide and Reference*
- *IBM DCE 3.1 for AIX: Application Development Guide—Introduction and Style Guide*
- *IBM DCE 3.1 for AIX: Application Development Guide—Core Components*
- *IBM DCE 3.1 for AIX: Application Development Guide—Directory Services*
- *IBM DCE 3.1 for AIX: Application Development Reference*
- *IBM DCE 3.1 for AIX: Problem Determination Guide*
- *IBM DCE 3.1 for AIX: Release Notes*

You can view a DCE book or download a Portable Document Format (PDF) version of it from the IBM DCE Web site at:

<http://www.ibm.com/software/network/dce/library>

Red Books

IBM's International Technical Support Organization (ITSO) has published a number of redbooks related to the RS/6000 SP. For a current list, see the ITSO Web site at:

<http://www.redbooks.ibm.com>

Non-IBM Publications

Here are some non-IBM publications that you may find helpful.

- Almasi, G., Gottlieb, A., *Highly Parallel Computing*, Benjamin-Cummings Publishing Company, Inc., 1989.
- Foster, I., *Designing and Building Parallel Programs*, Addison-Wesley, 1995.
- Gropp, W., Lusk, E., Skjellum, A., *Using MPI*, The MIT Press, 1994.
- Message Passing Interface Forum, *MPI: A Message-Passing Interface Standard, Version 1.1*, University of Tennessee, Knoxville, Tennessee, June 6, 1995.
- Message Passing Interface Forum, *MPI-2: Extensions to the Message-Passing Interface, Version 2.0*, University of Tennessee, Knoxville, Tennessee, July 18, 1997.
- Ousterhout, John K., *Tcl and the Tk Toolkit*, Addison-Wesley, Reading, MA, 1994, ISBN 0-201-63337-X.
- Pfister, Gregory, F., *In Search of Clusters*, Prentice Hall, 1998.

Index

Numerics

2615 messages 187

A

about this book xi
administration 20
AIX error log
 environment 35, 37
 making effective use of 20, 26, 36
 recording information to 37
application programming interface
 fc_display_fid 44, 55
 fc_eid_init 52
 fc_eid_is_set 53
 fc_init 41, 48
 fc_log_err 42
 fc_log_error 66
 fc_push_stack 43, 59
 fc_test_stack 42, 57
audience of this book xii

B

BSD system log
 recording information to 37

C

commands
 fccheck 45, 108
 fcclear 45, 105
 fcdecode 44, 97
 fcdispfid 44, 95
 fcfilter 44, 99
 fcinit 41, 74
 fclogerr 42, 87
 fcpushstk 43, 80
 fcreport 44, 103
 fcstkrpt 44, 101
 fcteststk 42, 78
creating an FFDC environment
 basic 5
 error stack 5

D

diagnosing
 complex failure situation 2
diagnostics
 basic tests 21
directory
 /var/adm/ffdc 19
 dumps 19
 stacks 19
 permissions 19
dumps, location of 19

E

end-user commands, overview
 fccheck 45
 fcclear 45
 fcdecode 44
 fcreport 44
 fcstkrpt 44
environment, variable
 FFDCSDISABLE 20
error
 log, AIX 23
 stack, FFDC 23
error-stack
 allocated 19
 consumption 19
 disabling 20
 removal 20
 safeguard 19
errsave
 for kernel extensions 23
examples, source-code
 daemon 148
 client.c 148
 ffdcInt.err.E 175
 ffdcInt.msg 183
 limitations of xiii
 parent/child 111
 child1.c 119
 child2.sh 125
 ffdcexpl.msg 145
 grandp.c 133
 Makefile 147
 parent.sh 111
 utils.c 140

F

fc_display_fid Subroutine 55
fc_eid_init Subroutine 52
fc_eid_is_set Subroutine 53
fc_init Subroutine 48
fc_log_error Subroutine 66
fc_push_stack Subroutine 59
fc_test_stack Subroutine 57
fccheck Command 108
fcclear Command 105
fcdecode Command 97
fcdispfid Command 95
fcfilter Command 99
fcinit Command 74
fclogerr Command 87
fcpushstk Command 80
fcreport Command 103
fcstkrpt Command 101
fcteststk Command 78
FFDC environment 4
 basic 5

FFDC environment 4 (*continued*)
 creating 5
 recording in 12
error stack 5
 creating 5
 inheriting 6
 recording in 13
 scope of 9

FFDC error stack
 allocated 19
 disabling 20
 environment 38
 making effective use of 38
 recording information to 39

FFDC failure identifier
 fid token described 35
 passing fid to callers 36, 39
 used in error stack 39

FFDCSDISABLE
 environment variable 20

file
 header 46
 removal from directory 20

First Failure Data Capture messages 187

H

header file, <rsct/ct_ffdc.h> 46

I

inheriting an FFDC environment
 error stack 6

installation
 directory 19
 package 19

interfaces, overview 41

L

log, AIX error 23

M

maintenance
 removal of error stack 20
man (manual) pages 45
manual pages for public code 226
messages
 First Failure Data Capture messages 187

P

path
 /var 19
 FFDC directory 19
permissions
 directory 19
prerequisite knowledge for this book xii

R

recording facility
 log, AIX 23

recording facility (*continued*)
 stack, FFDC 23
 usage, by software type 24
 application 26
 daemon 25
 device driver 24
 library 25
 script 26

removal
 error stack 20
reporting results 14
root cause
 determining 3

S

scope, of error stack 9
space, file
 consumption 19
stack, FFDC error 23
stacks, location of 19
subroutines
 fc_display_fid 55
 fc_eid_init 52
 fc_eid_is_set 53
 fc_init 48
 fc_log_error 66
 fc_push_stack 59
 fc_test_stack 57

T

templates
 constructing 26
terminology xiv
trademarks 214

U

utilities, FFDC 41

V

var directory 19

Readers' Comments — We'd Like to Hear from You

RS/6000 Cluster Technology
First Failure Data Capture Programming Guide and Reference

Publication No. SA22-7454-00

Overall, how satisfied are you with the information in this book?

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Overall satisfaction	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

How satisfied are you that the information in this book is:

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Accurate	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Complete	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Easy to find	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Easy to understand	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Well organized	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Applicable to your tasks	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Please tell us how we can improve this book:

Thank you for your responses. May we contact you? Yes No

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

Name

Address

Company or Organization

Phone No.



Fold and Tape

Please do not staple

Fold and Tape



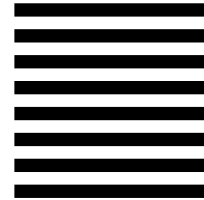
NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

IBM Corporation
Department 55JA, Mail Station P384
2455 South Road
Poughkeepsie, NY 12601-5400



Fold and Tape

Please do not staple

Fold and Tape



Program Number: 5765-D51 (PSSP); 5765-D28 (HACMP)



Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

SA22-7454-00

