ClearDDTS Administrator's Guide

RATIONAL
S O F T W A R E

Contents

6 Reconfiguring a ClearDDTS Network

7 Understanding the Master Template File

8 Customizing ClearDDTS

A Contents of a Defect Record

B Converting to ClearDDTS

C Sample Filter Command Script

D E-mail Submission API

E Creating Graphs with Graphbug

Preface

Rational's Distributed Defect Tracking system, ClearDDTS™, tracks and manages defects (bugs) and enhancement requests throughout the life cycle of a product or project. It is also an integral part of Rational's complete quality assurance (QA) solution.

Intended audience

The *ClearDDTS Administrator's Guide* is intended for system administrators and managers responsible for maintaining and customizing the ClearDDTS product. It describes how to set up and manage a ClearDDTS network as well as how to configure the system to suit your unique business requirements.

Although basic system administration is simple, it is necessary to be familiar with the following:

- UNIX operating system and common UNIX commands (or the POSIX shell if working in a POSIX-compliant environment)
- basic system administration commands for your platform
- requirements of your business environment
- web servers, client browsers, and your HTTP server configuration for use with the *webddts* interface

Using this manual

The *ClearDDTS Administrator's Guide* contains all of the information you need to manage ClearDDTS as shipped and provides the information you need to begin configuring the product to suit your unique business environment.

The manual is organized as follows:

- Chapter 1, Understanding ClearDDTS Operation, provides an introduction to ClearDDTS and an overview of the ClearDDTS distributed operation and network configuration.

- Chapter 2, Using Administration Utilities, describes the key utilities you use to manage the system.

- Chapter 3, Administering the Network, describes how to set up and maintain your ClearDDTS distributed network.

- Chapter 4, Managing Remote Access Between Multiple Installations, describes the use of import and export files for accessing projects across multiple ClearDDTS machines.

- Chapter 5, Maintaining Classes and Projects, describes commands for class and project creation, deletion, and maintenance.

- Chapter 6, Reconfiguring a ClearDDTS Network, describes how to reconfigure your ClearDDTS network when you need to move projects or machines.

- Chapter 7, Understanding the Master Template File, describes the structure of the master.tmpl file, field derivations, and how it is used by the *webddts* interface.

- Chapter 8, Customizing ClearDDTS, provides the information you need to customize various aspects of ClearDDTS.

- Chapter 9, Creating Custom ClearDDTS Reports, describes how to add your own custom reports to ClearDDTS.

- Chapter 10, Customizing Link Actions, describes defect linking and how to define the actions that you want applied based on those links.

- Chapter 11, Handling ClearDDTS Mail, provides detailed information about how ClearDDTS uses electronic mail and how you can customize that mail.

- Chapter 12, Managing ClearDDTS Security, describes how to control read and write access for defect records and how to set up a secure dial-in account for running ClearDDTS.

- Chapter 13, Managing and Customizing the ClearDDTS Database, describes the ClearDDTS database, how information is posted to the database, how to perform database maintenance, and how to modify the database schema.

- Chapter 14, Using the ClearDDTS SQL Interface, provides an introduction to SQL and to the ClearDDTS SQL command line interface, *ddtssql*.

- Chapter 15, Creating a Change Management System, describes the integration between ClearDDTS and various configuration management tools and how this integration allows you to establish a change management system.

Several appendices are included to provide specific technical information about selected topics, such as the content of defect records, how to convert an existing defect tracking system to ClearDDTS, and complete table and index definitions.

Where to go for more information

For special notes about this release of ClearDDTS, see the *Release Notes* file, [ddts_home]/RELEASE_NOTES. In addition to this guide, the ClearDDTS documentation set includes several other sources of information. Use the following roadmap to guide you to the documents that best suit your needs:

Installation and licensing

ClearDDTS Installation & Licensing Guide
- instructions for installing, upgrading, and licensing the product

ClearDDTS User's Guide
- getting started
- learning to use the tools
- performing key change management tasks

ClearDDTS features

ClearDDTS administration documentation in HTML (see http://<host>/wt/doc)

ClearDDTS HTML Documentation
- Man(1) and Man(5) pages
- Implementation Roadmap (steps you through the implementation process)
- All three reference guides

Questions or suggestions? Contact us

If you have suggestions for improving ClearDDTS or this manual, or if you have questions that are not answered here, contact the nearest Rational Technical Support Center.

Rational U.S.A.	**Rational Europe**	**Rational K.K.**
18880 Homestead Road	Siriusdreef 41	Kyowa Shinkawa Bldg.
Cupertino, CA 95014	2132 WT Hoofddorp	2020-8 Shinkawa
U.S.A	The Netherlands	Chuo-ku, Tokyo 104 Japan
Tel: (800) 433-5444	Tel: +31 (0)23 569 4310	Tel:+81 3 3551 9370
or (408) 863-4000	Fax: +31 (0)23 569 4302	Fax:+81 3 3551 9362
Fax: (408) 863-4590	e-mail: info@europe.rational.com	e-mail: info@japan.rational.com
e-mail: info@rational.com	support@europe.rational.com	support@japan.rational.com
support@rational.com		
URL: http://www.rational.com		

1

Understanding ClearDDTS Operation

This chapter provides a brief overview of the Distributed Defect Tracking system, ClearDDTS, and the ClearDDTS network. For more information about the operation and key components of the ClearDDTS system, you should read at least the first two chapters in the *ClearDDTS User's Guide*. These key concepts are only summarized in this manual. Topics covered in this chapter include:

▪ What is ClearDDTS?
▪ Distributed operation
▪ The ClearDDTS network
▪ How information is stored

What is ClearDDTS?

ClearDDTS, tracks and manages defects ("bugs") and enhancement requests throughout the life cycle of a project or product. Before you begin administering the system, you should be familiar with the basic principles of defect tracking and management used in ClearDDTS.

How defects are classified

ClearDDTS organizes defects into *classes* and *projects*. Every project is identified by a unique name and associated with a particular class. You can define your own classes or use the sample class, *software*, shipped with ClearDDTS. Once you have identified the classes you want to use, you need to define the projects that are members of each class.

Each project is associated with one *home system*. All project defects reside on that project's home system and users worldwide can log defects against it.

Understanding the defect life cycle

In most organizations, defects move through a number of common steps or *states*. This life cycle begins with the submission of a defect and generally ends with the resolution of the defect. In ClearDDTS, defect management involves keeping track of where a defect is in this life cycle. This movement from one state to another is called a *state transition.*

Distributed operation

ClearDDTS manages defect reports and related communications over the entire network of machines defined to ClearDDTS. This network of installed ClearDDTS sites is called the ClearDDTS network.

Because ClearDDTS is a distributed application, it deals very effectively with distributed project development and defect tracking. For example, a user could log a bug in Paris against a project in New York and an interested subscriber in Tokyo could be notified.

UNIX mail for flexible communication

ClearDDTS solves the communication problem by managing its distributed network entirely through ordinary UNIX mail messages. All that is required to network ClearDDTS systems together is an electronic mail link between the systems. It does not matter what gets the mail from one system to the other as long as it gets there correctly.

Thus, ClearDDTS automatically takes advantage of whatever technology is determined to be the best way to communicate between each pair of systems.

How ClearDDTS handles its mail

Incoming mail for ClearDDTS usually contains copies of defect reports and automatic administration commands. A daemon reads this mail, then determines what should be done with the mail message. The daemon forwards any items it cannot process automatically to the person who has been designated the ClearDDTS administrator.

After reading and processing each mail message, ClearDDTS records the transaction and deletes the message. ClearDDTS also acknowledges defect arrival and retransmits any defects that may have been lost due to external mail problems, thereby guaranteeing that defects will not be lost in the mail system.

Note: ClearDDTS handles its own mail automatically. You should not delete, modify, or redirect ClearDDTS mail. See Chapter 11, Handling ClearDDTS Mail, for more information about how ClearDDTS uses e-mail.

Naming conventions

In order to operate on a network-wide basis, ClearDDTS requires a special naming convention to uniquely identify ClearDDTS machines and defect records. This naming convention takes the form *XXXyy*. These five characters uniquely identify a ClearDDTS system and are also used to identify the origin of a submitted defect.

The name identifying a defect record is generated by the machine that originates the defect. The defect name is the machine ID, followed by a five-digit sequence number. For example:

533rd bug submitted from this machine

```
ALBqa00533
```

ClearDDTS
machine ALBqa
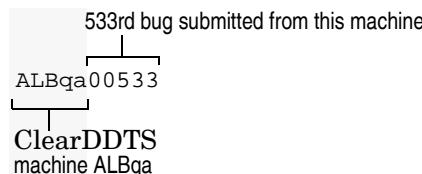
The ClearDDTS network

ClearDDTS makes extensive use of NFS (or RFS). A central server is typically used to store the ClearDDTS database and all local workstations access the system by NFS mounting the database. Thus, every local engineer deals with ClearDDTS as if it were installed on his local workstation.

Communication between local and remote installations

ClearDDTS also knows how to send and receive defects from remote systems through e-mail. As shown in the following figure, ClearDDTS will work over *uucp* or wide area networks.
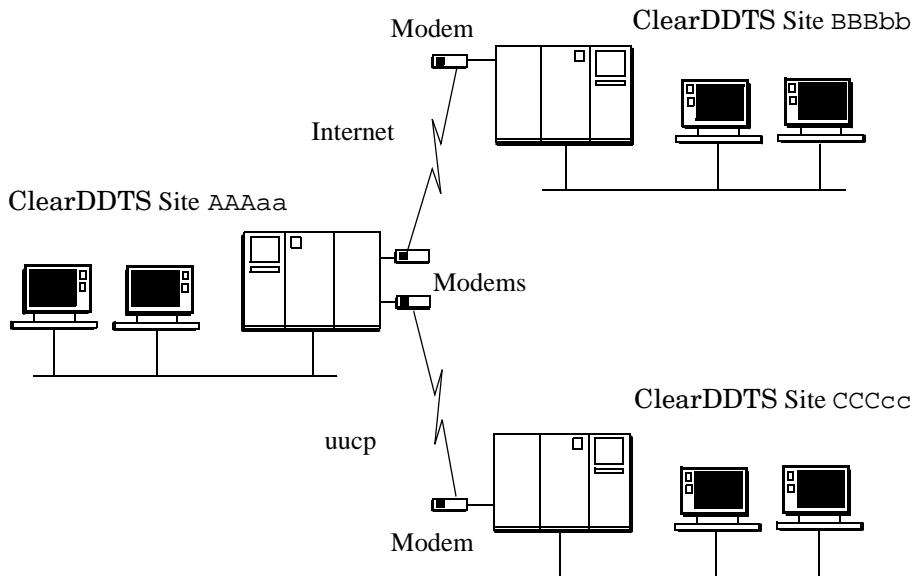
In this configuration there are three systems which are separate and distinct ClearDDTS installations:

- AAAaa
- BBBbb
- CCCcc

These systems are all locally customized to suit the local community's needs but are also completely interoperable. If a user local to AAAaa wants to submit a defect against a project on BBBbb, he just submits the defect normally. There is no difference between a local and remote submission. ClearDDTS takes care of all of the details of wrapping up the transaction in an e-mail message and sending it to machine BBBbb.

When the defect reaches BBBbb, the database transaction is run and an acknowledgment is sent from BBBbb back to AAAaa. Every time the defect is modified on BBBbb, a copy of the record is sent from BBBbb to AAAaa, AAAaa's database is updated, and e-mail is sent to the local submitter informing him of the change. This means that a local engineer can look at *any* defect owned on *any* machine and know that the defect is up-to-date.

Providing project subscriptions

Another distribution feature that ClearDDTS supports is called *subscription*. This feature works the same way as "replication" in large relational databases. ClearDDTS allows users to subscribe to projects on remote machines so that they are informed of all defects logged against projects in which they are interested.

For example, if users at machine CCCcc want to be informed about all defects for project *foo* located on machine BBBbb, they can subscribe to the *foo* project. All defect transactions that involve the *foo* project are sent to CCCcc, allowing an engineer at CCCcc to look at his local database for current information on the *foo* project.

When the mail system goes down

ClearDDTS also *guarantees* that no bugs will be lost in the mail system. Any lost mail is retransmitted. If a mail link totally goes down, the ClearDDTS administrator is notified, so that the link can be restored and the mail messages retransmitted.

How information is stored

In ClearDDTS, your defect records are stored in flat files in the `allbugs` directory. These files represent the "real" data and are used to post "copies" of the data to the SQL database. The SQL database acts as a cache to provide more efficient querying and performance.

The redundancy of storing data in both the flat files and the SQL database provides better security against potential loss or corruption of data and faster access to the information. It also allows for greater flexibility in changing the schema. Once changes have been made to a record, it is a simple procedure to rebuild the database to reflect the change.

2

Using Administration Utilities

There are several utility programs you will use to install, configure, and maintain the ClearDDTS system. This chapter includes an in-depth look at the most commonly used utility, *adminbug*, and provides an overview of the other key utilities you may use under special circumstances. Before reading this chapter, you should be familiar with the information in Chapter 1, Understanding ClearDDTS Operation. Topics covered in this chapter include:

- Basic ClearDDTS utilities
- Using adminbug
- More administrative utilities

Basic ClearDDTS utilities

Once you have set up your ClearDDTS environment, very little administration is usually required. Typically, you only need to add new projects as they are started, delete projects that are completed, or archive bugs that are old. These routine tasks ordinarily take about an hour per month, and are easy to perform even with little or no UNIX experience.

This chapter concentrates on the administration utilities available. Specific administrative tasks are described in detail in Chapter 3, Maintaining the Network, and Chapter 5, Maintaining Classes and Projects.

If you decide to customize ClearDDTS — for example, by modifying the contents of defect records, the format of output reports, or the way bug transitions are handled — system administration is somewhat more complex, and more specific

UNIX knowledge is required. Customization is described in Chapter 8, Customizing ClearDDTS.

The following utilities are used to perform the most common ClearDDTS administrative functions:

Utility	Description
adminbug	Performs most ClearDDTS administrative tasks.
batchbug	Makes a programmatic change to a defect.
ddtsclean	Performs housekeeping tasks, such as cleaning up log files.
ddtsversion	Prints the ClearDDTS version number.
newduser	Replaces one user with another in the database.
patchbug	Modifies the contents of one or more defect records.
projstat	Shows status of ClearDDTS projects.
rdtest	Detects syntax errors in template files.
rmbug	Deletes bugs from the ClearDDTS system.
tmpltest	Tests new or modified template files.

Because *adminbug* is the most frequently used administration command, it is explained in detail in this chapter. The other utilities are only rarely used and only touched on briefly in this chapter. Refer to the manual (man) pages for more complete information.

Using adminbug

The *adminbug* utility is used to administer ClearDDTS projects and the ClearDDTS network. It has commands for project creation, project deletion, remote site connection, reconfiguration, and so on. Most ClearDDTS administration is done through this program.

Entering commands

When invoked, *adminbug* displays the following:

```
*******************************************
**** ClearDDTS Administration Utility ****
*******************************************
DDTSHOME=/usr/testing
ClearDDTS is in maintenance mode.

Enter '?' for help at any time.

adminbug>
```

As indicated in this prompt, you can get help at any point by entering a question mark (?). When you do this, an informative message appears describing what is happening.

Once you have invoked *adminbug,* you can enter commands to perform various administrative tasks. Most of these commands prompt you for specific information and accept input interactively.

The following table lists the commands you can use in *adminbug.* In this table, the second column indicates who can run each command as follows:

- *ddts* — Only a user logged on as *ddts* can run this command.
- *PM* — Only the project manager or *ddts* user can run the command.
- *any* — Any user can run the command.

Command	Who	Description	Database rebuild required?	Page Reference
inst	*ddts*	Install this system.	Yes	3-2
dsbl	*ddts*	Disable (de-install) ClearDDTS on this machine.	No	3-4
mins	*ddts*	Modify installation parameters (such as administrator names).	No	3-5
dbms	*ddts*	Build the ClearDDTS database.	Yes	3-6
chdb	*ddts*	Change the database (RDBMS) used by ClearDDTS.	Yes	3-6
conn	*ddts*	Establish a connection between this site and another.	No	3-8

Command	Who	Description	Database rebuild required?	Page Reference
dcon	*ddts*	Remove the connection between this site and another.	No	3-9
clas	*ddts*	Add a new class of projects to the system.	No	5-1
dcls	*ddts*	Delete a class of projects from the system.	No	5-2
rcls	*ddts*	Rename a class.	Yes	5-3
meta	*ddts*	Add a meta-class to the system.	No	5-4
mmta	*ddts*	Modify a meta-class.	No	5-5
aprj	*ddts*	Add a new project to ClearDDTS.	No	5-5
cprj	*PM*	Close down a project temporarily, preserving defect data.	No	5-13
dprj	*ddts*	Delete a project from this machine, reclaiming disk space.	Yes	5-14
oprj	*PM*	Reopen a closed project.	No	5-14
mprj	*PM*	Modify project parameters (such as mail notification lists).	No	5-14
bprj	*PM*	Rebroadcast project information to the ClearDDTS network.	No	5-16
sprj	*PM*	Save a project. (Allows moving project to a different machine.)	No	5-16
rprj	*ddts*	Restore a project. (Installs saved project on a new machine.)	Yes	5-17
renm	*ddts*	Rename a project.	No	5-18
asub	*ddts*	Ask for subscription to a project.	No	5-18
dsub	*ddts*	Delete subscription to a project.	No	5-20
msub	*ddts*	Modify subscription parameters.	No	5-20
smnt	*ddts*	Shut down ClearDDTS on this machine for maintenance.	No	2-5
emnt	*ddts*	End maintenance and bring ClearDDTS back up.	No	2-5
lprj	*any*	List all project parameters for specified project.	No	5-21
lsit	*any*	List other systems connected to this system (with *conn* command).	No	3-10
ladm	*any*	List ClearDDTS administrators on this machine.	No	3-10

Command	Who	Description	Database rebuild required?	Page Reference
lbug	*any*	List projects to which defects may be submitted.	No	5-21
lown	*any*	List projects owned on this machine.	No	5-22
lsub	*any*	List projects subscribed to on this machine.	No	5-22
quit exit	*any*	Exit *adminbug*.	No	2-5
alic	*ddts*	Add evaluation licenses to ClearDDTS.	No	3-11
? help	*any*	Display context-sensitive help.	No	NA
? *cmnd*	*any*	Explain the specified command (for example, use *? aprj* to get help on the *aprj* command).	No	NA

Set up maintenance mode (smnt)

This command puts ClearDDTS on this machine into maintenance mode. In maintenance mode, all ClearDDTS daemons terminate and ClearDDTS software can be safely updated. During this period, ClearDDTS mail requests might still be arriving, but ClearDDTS takes no action on them until you enter the *emnt* command to exit maintenance mode.

Exit maintenance mode (emnt)

This command brings ClearDDTS out of maintenance mode. The ClearDDTS daemons are automatically restarted, and they begin processing any new or pending requests.

Quit adminbug (quit)

At the adminbug> prompt, this command exits *adminbug*. To abort individual *adminbug* functions at any interactive prompt, type your interrupt character to return to the adminbug> prompt.

More administrative utilities

There are a few additional administrative utilities that deserve mention and are summarized below. These are rarely used or only need to be set up once. For more information about any of these utilities, see the man pages.

ddtsclean

The *ddtsclean* utility cleans up log files and any problems that occur due to a system crash. It should be invoked once a day from the system's *crontab* file, or if your system allows user-specific crontab files, from the *crontab* file belonging to *ddts*. See the *ClearDDTS Installation and Licensing Guide* for details on how to do this. Once *ddtsclean* is set up in the *crontab* file, it requires no maintenance.

ddtsversion

This utility prints out the ClearDDTS version number. If you administer ClearDDTS on multiple systems, this is very helpful when it comes time to update them, so you can determine immediately which have been updated and which have not. (This utility is the same as the *whichddts* utility provided in previous releases.)

newduser

This utility allows you to transfer ownership of defects from one employee to another (for example if an employee leaves the organization and is replaced). You can use this script for both defect submitters and those that repair them.

patchbug

If you modify ClearDDTS to collect new information in defect reports, you can use *patchbug* to change all of the existing defect records to include the new information without updating them all by hand. This utility allows you to add, delete, or change fields in

existing defect records. The utility can only be run by the user *ddts*. Depending on how *patchbug* is run, you may need to rebuild the ClearDDTS database with *adminbug dbms*.

Note: The *patchbug* utility does not allow you to add or edit enclosures. Use the *batchbug* utility to do this.

batchbug

The *batchbug* utility is used for programmatic changes to individual defects and have those changes immediately posted to the database. The utility is different from *patchbug* in that it is not meant for large scale changes to the database. Rather, it is meant for changes to one or two bugs at a time.

Other utilities that need to integrate to ClearDDTS should do so through this utility.

projck

The *projck* utility checks the health of the project and class database and reports errors to stdout. This utility is run every night by *ddtsclean*.

projstat

The *projstat* utility displays information about the ClearDDTS system. It lists all of the currently active projects, those projects owned on this machine, those projects subscribed to by this machine, and more.

refreshbug

The *refreshbug* utility can be used when you are subscribing to off-machine projects and the mail system has failed. If mail has failed or been lost, bugs from subscribed projects may have been lost. The *refreshbug* program will refresh your subscribed database from the machine where the project is owned. This utility is

automatically invoked once a day by *ddtsclean* to ensure the owned
database and the subscribed database conform.

If your system has mail problems, you may want to run this utility
more often.

rdtest

The *rdtest* utility is used to test for syntax errors in customized
template files. If you have customized ClearDDTS files, you can
run *rdtest* on these files to find any syntax errors before
implementing your customizations.

rmbug

The *rmbug* utility is used to remove a bug from the database.
Generally, it is better to resolve a bug than remove it; however, if
you do want to purge a bug from the database, you can use *rmbug*.
This utility removes the bug from the local machine, but does not
remove the bug from the network. To remove a bug from the
network, remove it from each machine individually. As mentioned,
it is generally better to just resolve the bug. A resolved bug is
resolved over the entire ClearDDTS network.

tmpltest

One way you can customize ClearDDTS is by defining your own
state transitions. The rules for moving from state to state are
defined in template files. If you create custom template files, you
can use *tmpltest* to execute these files and display the results.

3

Maintaining the Network

Once you have set up your ClearDDTS environment following the steps in the *ClearDDTS Installation and Licensing Guide*, little administration is required. You may, however, need to update your ClearDDTS network as changes in your environment occur, such as the addition of new machines or sites. This chapter describes the *adminbug* commands you can use to maintain your ClearDDTS network. For more information on running *adminbug* see Chapter 2, Using Administration Utilities.

This chapter includes the following topics:

- Install machine on ClearDDTS network (inst)
- Disable ClearDDTS machine (dsbl)
- Modify ClearDDTS installation parameters (mins)
- Build ClearDDTS database (dbms)
- Change database (chdb)
- Establish connection between sites (conn)
- Remove connection between sites (dcon)
- List ClearDDTS administrator names (ladm)
- List other sites connected to this system (lsit)
- Add licenses (alic)

Messages and prompts are displayed in a regular Courier font and responses are shown in **Bold Courier**. Some prompts have default responses. You can simply press RETURN with no input to accept the default.

Install machine on ClearDDTS network (inst)

You use this command to add a machine to the ClearDDTS network. This command is automatically invoked by the *ddtsinstall* script when you first install ClearDDTS. The following is a line-by-line explanation and example of the ensuing dialog:

```
Installing ClearDDTS on host <machine>.
This machine will be used as the ClearDDTS server.

Is this correct? (y/n): y
```

- By default, the *inst* command installs the ClearDDTS database on the machine that *adminbug* is currently running on, so normally the machine name displayed is correct and you can simply enter *y*. However, in an NFS or RFS environment, the name of the *server* must be provided. In this case, enter *n* and run this command from the server.

```
What are the mail address(es) of the ClearDDTS administrator(s)?

Mail address(es): fred mike chris@piper
```

- Enter the mail address(es) of the person(s) responsible for administering ClearDDTS on this machine. If an administrator reads mail on another machine, give a mail address that will reach that user from this machine. In the example, there are three administrators: fred, mike, and chris (on machine piper).

  The users listed are informed by mail of any network errors or ClearDDTS informational messages that are generated. Also, if any personal mail is sent to *ddts* by accident, ClearDDTS forwards this mail to the administrators.

Note: The names entered are *mail addresses*. They are not used for granting permissions. They are used strictly for sending informational e-mail messages. Note also that *ddts* is a special user and not a legal response to this prompt.

```
What is the five character site identifier used to uniquely identify
this site within the ClearDDTS network?

Site identifier (XXXyy): GPDqa
```

- Enter a site identifier to uniquely identify the machine in the ClearDDTS network. The first three characters must be upper case and the last two must be lower case. All must be letters of the alphabet with no numbers or special characters. You may want to include a business entity acronym (such as PSD for Portable Systems Division), but you should be sure that the site ID you choose is not being used by sites you want to connect to in the future. In the example, the site is identified by the acronym GPDqa (General Products Division, quality assurance).

  If you do not know what names other sites have chosen, call them and ask. If they do not know their own site ID, ask them to run the *ddtshostname* command. This will display the ClearDDTS site ID.

```
What is the name of your organization or division? This name is used
in the Submitter Information section of all records submitted from this
site.

Organization/division name: General Products Division Lab
```

- Enter the name of your business organization. In this example, the business organization is identified as *General Products Division Lab*. This string will appear on all bug reports submitted from the new machine and should be descriptive enough to uniquely identify the organization. Any kind of identifying string up to 30 characters can be used.

```
What is the mail command to use on this machine? If a '%s' is supplied
as part of the command, DDTS will substitute the subject of the mail
in its place. Be sure to quote the '%s' so spaces in the subject will
be handled correctly.

Mail command: /usr/ucb/mail -s '%s'
```

- Because all ClearDDTS networking depends on the mail system, you need to inform ClearDDTS how to send mail on your system. The default mail command varies depending on your architecture.

  Enter the **full** path name of the mailer program you use. On a Sun Microsystems or other BSD-oriented system, this is generally */usr/ucb/mail*. On a UNIX System V machine like Hewlett-Packard this is */usr/bin/mailx*. Other systems may use different mailers. If necessary, ask your system

administrator for the appropriate mail program and invocation parameters to use.

The *-s '%s'* parameters are used for mailers that support a subject line. If your mailer does not support a subject line, **do not** include the *-s '%s'* parameters in your response.

Note: When installing ClearDDTS for the first time, the *ddtsinstall* script automatically invokes the *adminbug alic* and *dbms* commands. If you are running this command after taking a ClearDDTS machine off the network (with *dsbl*), be sure to run the *alic* and *dbms* command to finish the reinstallation.

Disable ClearDDTS machine (dsbl)

Use the *dsbl* command to de-install the system and take it cleanly off the ClearDDTS network. After you run the *dsbl* command, users will not be able to log defects against projects on that machine. Typically, this command is used for moving ClearDDTS to another machine, or changing the ClearDDTS site ID.

This command starts off with a warning:

```
WARNING WARNING WARNING WARNING WARNING WARNING

This command will take this machine off the DDTS network.

Is this really what you want to do? (Y/N): y
```

▪ Enter *y* to continue. If you answer anything other than *y* or *Y*, the command is aborted.

```
Going into maintenance mode...
Killing the ClearDDTS daemons...
ClearDDTS is in maintenance mode.
Disabling the system...
ClearDDTS is now disabled.
```

This command does not destroy any data; it merely takes the system cleanly off the ClearDDTS network.

Note: The *dsbl* command also removes any subscriptions to or from projects on this machine. If the system is re-enabled later, any desired subscriptions to projects should be restored with the

adminbug asub command. To finish a ClearDDTS de-installation, delete the ClearDDTS *crontab* entries.

Modify ClearDDTS installation parameters (mins)

From time to time, you may want to modify some of the ClearDDTS installation parameters. This can be done with the *mins* command. This command allows you to change the ClearDDTS administrator names, the invocation string for mail, or the organization name.

The dialog is similar to portions of the *inst* command except that the current answers to the questions are also displayed along with the prompt (for more information about these prompts, see "Install machine on ClearDDTS network (inst)" on page 3-2). To retain the current information, just press RETURN. If you need to make a change, backspace over incorrect items and enter the new response.

```
What are the mail address(es) of the ClearDDTS administrator(s)?

Mail address(es): fred chris@piper
```

▪ Change the mail addresses of the persons who will be administering ClearDDTS on this machine.

```
What is the mail command to use on this machine? If a '%s' is supplied
as part of the command, DDTS will substitute the subject of the mail
in its place. Be sure to quote the '%s' so spaces in the subject will
be handled correctly.

Mail command: /usr/ucb/mail -s '%s'
```

▪ Change the mailer command used for your system. For example, on an HP-UX system this may be */usr/bin/mailx*.

```
What is the name of your organization or division? This name is used
in the Submitter Information section of all records submitted from this
site.

Organization/division name: Software Business Division
```

▪ Change the name of the business organization (up to 30 characters).

Build ClearDDTS database (dbms)

This command rebuilds the ClearDDTS database. In general, the *dbms* command is used:

- during installation or reinstallation of ClearDDTS
- when moving a project to or from another machine
- after making changes to your database (for example, with *patchbug*)

Running this command displays the following:

```
Do you want to build/rebuild the ClearDDTS database? (y/n): Y

If there are many bugs this may take a while.

Going into maintenance mode...
Killing the ClearDDTS daemons...
ClearDDTS is in maintenance mode.
Building the ClearDDTS database on the internal database...
Creating table defects...done.
Creating table enclosures...done.
Creating table change_history...done.
Loading defect records from /nfs/testing/allbugs/00
.
.
.
Loading defect records from /nfs/testing/allbugs/99...
Exiting maintenance mode...
Starting the ClearDDTS daemons...
Rebuild of the ClearDDTS pure database completed successfully.
```

Change database (chdb)

Use this command to change your database vendor. After installation you can change from the internal ClearDDTS database to an external Oracle database. Running this command displays the following:

```
Current database information:

    Vendor           : pure
    Instance         : ddts
    ClearDDTS login  : none/none
    Query only login : none/none

Which relational database system do you want to use with ClearDDTS?

Database vendor name: oracle
```

- Enter the name of your database vendor.

```
What is the name of the oracle instance where the ClearDDTS
tables will be stored?

Instance: dbinst
```

- Enter the *database instance* associated with this database. The
  instance acts as a network identifier for locating the appropriate
  database to use. With an Oracle database, you need to identify
  the instance associated with this database before the database
  can be mounted.

```
What is the ORACLE_HOME value needed to connect to this instance?

ORACLE_HOME: /usr/oracle
```

- Enter the path to the Oracle home directory.

```
What is the database login name of the owner
of the ClearDDTS tables? This user should be the only
user that is able to modify the ClearDDTS tables.

ClearDDTS login: ddts
Password:
```

- Enter the *database user* name you want designated as the owner
  of ClearDDTS tables. (Use your database vendor's tools to create
  this user.) For security reasons, only this database user can
  modify the database. In most cases you should use the login *ddts*
  to identify this user. For the internal ClearDDTS database, no
  entry is needed.

```
What is the login name of a database user that is
used only for queries? This user should not be able
to modify any information in the database.

Query only login: readonly
Password:
```

- Enter the *database user* name to use to provide read-only access
  to the database. The "readonly" database login allows you to give
  your general user population permission to run queries without
  allowing them to modify the database. Use your database
  vendor's tools to create this user. No entry is needed for the
  internal ClearDDTS database.

```
Vendor: oracle
Instance: dbinst
ClearDDTS login: ddts/ddts
Query only login: readonly/readonly
```

```
Are these values correct (y/n)? y
```

▪ Verify your settings and enter *y* if they are correct.

```
Updating the database configuration files...

Running the dbms command to rebuild the database.

Do you want to build/rebuild the ClearDDTS database? (y/n): y
```

▪ Enter *y* to build the ClearDDTS database using default table parameters. You can modify these parameters if necessary using tools from the database vendor.

For more information about the file created with this command, see Appendix G, Using an Oracle Database. For information about creating database users and instances, refer to your database vendor's documentation.

Establish connection between sites (conn)

The *conn* command connects one site to another site. This command sets up a communication pathway so that two machines can exchange information about projects. When this command is successfully executed, users on either system can log defects against projects located on the other machine.

Note: Only one site needs to execute this command, not both.

For security and administrative reasons, project data (actual bugs) are not exported from or imported into remote systems unless explicitly authorized by the content of special *import* and *export* files. Project information, such as name, class, and description, is always exported. Read Chapter 4, Managing Remote Access Between Multiple Installations, and edit the import and export files appropriately *before* you issue this command.

Note: The *conn* command does not exchange Class template files. This must be done manually by the ClearDDTS administrators on each system.

Running this command displays the following:

```
What is the site id of the remote site to establish a connection with?

Site id (XXXyy): DSDaa
```

- Enter the ClearDDTS site ID of the system to which to connect. If you do not know the remote site's machine name, ask the remote ClearDDTS administrator or have someone on the remote site run the *ddtshostname* command.

```
What is the e-mail address of the DDTS user at the remote site?

Remote DDTS e-mail address: ddts@jupiter.com
```

- Enter the mail address to the other site.

```
Enter the e-mail addresses of the persons to inform by mail when
the connection with the remote site has been established.

Mail addresses: fred bill
```

- When the connection request arrives on the remote site, the remote site ClearDDTS administrators are informed by mail that a connection request has been made. When the connection request is established, the users listed above, fred and bill, and the local ClearDDTS administrators are notified that a connection has been established.

  An error message or no confirmation message after a long period means that the installation request failed. If the mail communication link to the remote system is a modem or other slow connection, you may have to wait until mail is delivered to ClearDDTS and a confirmation is returned. If communication between systems is over a fast LAN such as Ethernet, confirmation should only take a few minutes.

Remove connection between sites (dcon)

Use this command to sever the ClearDDTS network connection between this site and a remote site. After this command is executed, users at each site are no longer able to submit bugs against projects on the other site. Bugs already in transit are not affected.

Note: Before disconnecting from a remote site, use the *dsub* command to delete any subscriptions to projects on that site. To find the projects to which your site subscribes, use the *lsub* command. If you reconnect to a site, use the *asub* command to reestablish subscriptions to projects on that site.

The *dcon* command starts by listing all remote machine IDs and paths, then asks for the name of the system to disconnect. For example:

```
The following are the remote sites that you are connected to:
SPDqa  path
GSDbc  path
DSDbc  path

Which site do you want to disconnect from?

Site Id(XXXyy): GSDbc

Please wait, this will take a while...
```

List other sites connected to this system (lsit)

This command lists the ClearDDTS machine names of the other ClearDDTS systems that are connected to the system using the *conn* command. For example:

```
GSDqa  ddts@larry
PSDqa  ddts@moe
PPDxx  ddts@curley
```

List ClearDDTS administrator names (ladm)

This command lists the ClearDDTS administrator names. For example:

```
Administrator: mike bill bob
```

Add licenses (alic)

Use this command to add evaluation licenses to ClearDDTS. Have a copy of your license certificate available.

```
Are you entering an evaluation license? (y/n): Y

Enter the information EXACTLY as it appears on your license certificate.

   Hostname                          :<machine>
   Hostid                            :ANY
   Number of licenses                :4
   Expiration Date                   :30-Apr-1998
   Password                          :JSJ8XCJD834MJ9JS78Y2


Adding the license...
```

For information about licensing options and entering permanent licenses, see the *ClearDDTS Installation and Licensing Guide*.

4 Managing Remote Access Between Multiple Installations

This chapter describes how to use ClearDDTS import and export files to manage remote access between multiple ClearDDTS installations. You do not need to read this chapter if:

- your ClearDDTS software is only installed on one system
- you are using NFS or RFS and ClearDDTS is installed on only one system

This chapter covers the following topics:

- How the import and export files are used
- The export file
- The import file

How the import and export files are used

In ClearDDTS, the *adminbug* utility is used for almost all of your administration needs. Therefore, you generally do not need remote administrators to agree on a whole set of files to edit and then force the joint editing of those files. There are, however, two files that need to be properly edited if you are using ClearDDTS in a distributed fashion. These are the *export* and *import* files located in *~ddts/conf*.

- The *export* file defines the projects other systems can know about and the projects remote systems can log bugs against. It is used to restrict what ClearDDTS information goes to remote sites.

  For example, assume you have several projects that you want remote systems to know about and log bugs against, but you also have some projects that are strictly for local use. You can use the *export* file to prevent remote sites from logging bugs against these specific local projects.

- The *import* file defines the projects a system will accept from external sites.

Note: The import and export files determine what projects can be sent or accepted at a site. To actually establish a connection between sites you must use the *adminbug* project subscription commands subscribe to a project (*asub*), modify a subscription parameters (*msub*), list project subscriptions (*lsub*), or delete a subscription (*dsub*). See Chapter 5, Maintaining Classes and Projects, for more information.

The *export* file

The *~ddts/conf/export* file is used to decide which projects remote systems are able to log bugs against. The file contains a list of remote machines and the projects that those remote machines are allowed to see. The mechanism used for this is simple and is best described through an example.

Examining a sample file

A sample *export* file is shown below:

```
CMMqa: compiler
KMM*: * !compiler
SVR*: *.4.3
*: *
```

- Characters to the *left* of the colon (:) are ClearDDTS site IDs, that is, the name by which ClearDDTS knows remote machines.
- Characters to the *right* of the colon represent the projects on your system that users on the selected remote machines can log bugs against.
- Shell metacharacters (*, ?, [and]) can be used to describe both the sites and the projects. In addition, the unary negation operator '!' can be used with projects.
- Use a space (space bar) as the separator to list more than one item on either side of the colon.

In this example, the first line says machine CMMqa can access the compiler project and nothing else. The second line says for all

machines with the KMM prefix (KMM*), export all local projects except the compiler project. The next line says site SVR can access only projects with a suffix of '.4.3'. The last line says for all other systems export all projects.

How the file is read

It is important to note that only those projects specifically named will be exported. An empty file or no file means that nothing is to be exported. If the rest of a line after a colon is empty, nothing is sent to the machine specified on the left of the colon. If a site ID is matched, the rest of the file is ignored.

Important: The order of lines in this file matters, because in deciding whether to export a particular project, ClearDDTS reads the file in order until a line is found whose left side matches the machine in question. Consider the following example:

```
CMM*: FRM*
CMMqa: *
```

The first line here indicates that you want to export only projects prefixed with FRM to all CMM sites. Since any site ID that starts with a CMM is matched in the first line, the second line is never executed. If these lines were reversed, CMMqa would get all projects, and other CMM machines (CMMdd for example) would only get projects prefixed with FRM.

When ClearDDTS finds a line in the export file that applies to the machine in question, that line is examined completely. For each item in the list, if it does not begin with a !, each matched project is added to the set of projects to be broadcast to that machine. If the item begins with a !, each project matched by the remainder of the item is deleted from the set of projects (if it is in the set). When the whole item list has been processed, those project names remaining in the set are the ones distributed to that machine.

Further examples

Because of the importance of this file, here are more examples and what would result from each:

Lines in export file	Result
`*: *`	Allow all projects to be exported to all remote machines (projects available to everyone). This is the default *export* file.
`CMM*: * !os*`	Export all local projects except projects with an `os` prefix to all CMM remote sites.
`CMMqa: [A-Z]* framus`	Export only projects that start with a capital letter and the `framus` project to remote site CMMqa.
`CMMqa: compiler !*`	Incorrect usage. This line indicates you want to export the `compiler` project and to export nothing (`!*`).
`*: !*`	Do not export any projects from the local ClearDDTS system to any remote system. This is the same as an empty *export* file.
`*:`	Do not export any projects from the local ClearDDTS system to any remote system. This is the same as an empty or missing *export* file.

Importance of project naming conventions

Because ClearDDTS searches for the first matching site ID in the *export* file and sequentially looks at each pattern on that line to see if a project should be exported, the *export* file works best if you come up with a convention for project names.

It is strongly recommended that you use a project naming convention with agreed-upon prefixes and suffixes. For example, a prefix for organizational groups and a suffix for release levels often works well. Thus, an operating system project might look like `OS.4.0`, `OS.4.1`, `OS.4.3`, and so forth.

Another convention could be used where the first character designates whether the project is released yet. Thus an 'R' prefix would designate released projects and a 'D' prefix would designate projects still in development. You could then edit the *export* file to only export projects that were prefixed with an 'R'.

Applying changes to the export file

If you make a change to the *export* file, you can force the change to be applied against remote systems by again issuing the *adminbug conn* command against that site. Alternately, if only one project needs changing, the *adminbug bprj* command can be used.

The *import* file

The *import* file works like the export file and has exactly the same syntax and semantics. The difference is that the *export* file controls projects that are exported from the local system and the *import* file controls what projects are allowed to be installed on the local system. The import file keeps unnecessary projects off your system.

Some examples:

Lines in import file	Result
`*: *`	Allow any project to be installed in your system.
`DAL*: !*` `*: *`	Accept any projects except projects that come from the `DAL` site.
`DAL*: RTE*` `*: *`	Accept any project except projects from the `DAL` site. From the `DAL` site, only accept projects that begin with the project name `RTE`.
`*:`	Do not accept any projects from any remote system. This is the same as an empty or missing *import* file.

5

Maintaining Classes and Projects

This chapter describes the *adminbug* commands you can use to maintain your classes and projects. For more information on running *adminbug* see Chapter 2, Using Administration Utilities. This chapter covers the following topics:

- Maintaining classes
- Maintaining projects

Messages and prompts are displayed in a regular Courier font and responses are shown in **Bold Courier**. Some prompts have default responses. You can simply press RETURN with no input to accept the default.

Maintaining classes

This section describes the commands you can use to maintain classes. This includes adding, deleting, and renaming classes, as well as creating and modifying meta-classes.

Add a new class (clas)

Use this command to add a new class of projects to ClearDDTS. A class is a collection of projects that share common attributes or are treated in the same manner. For example, you may want to have a common bug tracking methodology for all of your compiler projects and a different methodology for networking projects. All of the projects in a class share the same state transition definition files and I/O template files. They also all share the same set of

Management Reports and Metrics that may be applied to the projects.

When adding a new class of projects, the following dialog occurs:

```
Enter new class name:     networking
```

- Enter the new class. Do not include spaces in the class name. In this example, the user is creating a class of projects for networking.

```
Enter a one line description of this class:
These are networking projects.
```

- Enter a one-line description used for the *xddts* and *bugs* help files.

```
Enter Class prototype directory name:     software
```

- Enter the name of a current class that is most like the new class desired. You should enter a class name that already exists and that treats projects in a way similar to what you desire.

```
Copying proto dir to networking. One moment please ...

Creating shared query: By Identifier

Updating DBMS.  This will take a while...
```

When you run this command, *adminbug* creates a new class directory, *~ddts/class/<new_class_name>*. The class directory is where most ClearDDTS customization occurs. You should take a look at that directory now to see the files used in customization. A complete description of ClearDDTS customization is described in Chapter 8, Customizing ClearDDTS.

Delete a class (dcls)

This command deletes a class or meta-class from the system and removes the class directory associated with it. You should be

extremely careful with this command; it can destroy all of your customizations.

Note: When you delete a meta-class, the classes grouped under the meta-class are not deleted. Only their association as a meta-class and the meta-class directory are removed.

When invoked the *dcls* command presents a warning:

```
WARNING WARNING WARNING WARNING WARNING WARNING

This command removes the class from the system and reclaims the
disk space. All class customizations will be lost.


Is this really what you want to do?   Y


Enter class name: networking


This will take a few seconds...

Updating DBMS.  This will take a while...
```

If any open projects still exist in this class, the command issues an error and aborts. You must remove all existing projects in the class or assign them to a different class (with *mprj*) before executing this command.

Rename a class (rcls)

This command will rename an existing class. The dialog is:

```
Enter current Class name: foo

Enter new Class name: bar
```

▪ Enter the new class name. Do not include spaces in the class name.

The following message is displayed:

```
This will take a while . . .

You MUST run the adminbug(1) 'dbms' command to
incorporate the changes into the database.
```

To finish the task, run the *adminbug dbms* command to rebuild the database.

Create a meta-class (meta)

Use this command to create a meta-class. A meta-class allows you to group classes together for the purpose of metrics and link semantics. For example, you can use a meta-class to report information about all of the records in several different classes at once. Although ClearDDTS treats the meta-class the same as a regular class, no projects are directly associated with the meta-class.

When creating a new meta-class, the following dialog occurs:

```
Enter new metaclass name: Computer
```

- Enter the new meta-class name. Do not includes spaces in the name.

```
Enter a one line description of this class.
Hardware-Software-Tools Group
```

- Enter a one-line description used for *xddts* and *bugs* help files in the user interface.

```
Which classes should be included in this metaclass?
software hardware
```

- Enter the classes you want grouped under this meta-class.

```
Creating metaclass Computer.  One moment please ...

Updating DBMS.  This will take a while ...
```

When you run this command, *adminbug* creates a new class directory, *~ddts/class/<new_metaclass_name>*. This is the directory where you customize the metrics and link semantics. Meta-classes do not have records or projects directly associated with them. the

master.tmpl file for meta-classes is used only to determine what fields should be displayed for querying in the meta-class.

Modify a meta-class (mmta)

Use this command to modify information for a meta-class. For example, you can use this command to add classes to the meta-class.

When modifying a meta-class, the following dialog occurs:

```
Enter existing metaclass name: Computer

One line description of this metaclass:
Hardware-Software-Tools Group

Which classes should be included in this metaclass?
software hardware network

Updating metaclass Computer.  One moment please ...


Updating DBMS.  This will take a while ...
```

Maintaining projects

This section describes the commands you can use to maintain projects. You can perform a variety of activities including adding, deleting, renaming, and many others.

Add a new project (aprj)

Use the *aprj* command to install a new project on the system and on the ClearDDTS network. This command broadcasts to the entire ClearDDTS network your intention to accept and resolve defect reports pertaining to the new project.

Note: You can regulate which systems receive information about projects using the ClearDDTS *import* and *export* files. See Chapter 4, Managing Remote Access Between Multiple Installations, for more information about these files.

Considering project naming conventions

Because every project in ClearDDTS has a unique name and this name is used to group defect reports, we **strongly suggest** that you create a corporate naming convention for projects before continuing. If you exchange defects with other organizations or companies, or run metrics on your projects, you should define a logical naming convention. You may want to use the organization names (or abbreviations for them) as part of the names for projects in those organizations.

Entering project information

When adding a new project the following dialog appears:

```
Enter the project name:    compiler
```

- Enter any name that does not contain slashes, question marks, asterisks, or other special characters (+, ^, etc.). The dot character (.) can be used and, in fact, can be useful.

  To be able to exchange defects with older AT&T UNIX Systems, System V-based systems and SCO systems, you must restrict the total length of the project name to 14 characters. For modern systems, this restriction does not apply as long as the file system on which ClearDDTS resides supports long file names.

```
Enter a one-line description of project:
Defects against the C compiler.
```

- Enter a short description of the project, up to 45 characters. Try to be clear and concise. Remember that others will use this description to decide if this is the project against which they should log a defect report. With a poor description, defects may be logged against the wrong project or not logged at all.

```
Enter project part number:    960-153A
```

- If the project has a part number or other such code, enter it here. This can be left blank.

```
To which class should this project belong?    software
```

- A class is a collection of projects that are treated the same way. For example, you might have one class of projects called "software" and another class called "hardware". All projects in a class use the same state transitions and rules for moving a defect from state to state.

```
List remote site identifiers for those allowed to
modify bugs for this project.  A blank line indicates that
no remote site can modify bugs.
```

- If you want to allow remote sites to modify defects for this project, enter the ClearDDTS identifiers (XXXyy) for those sites. Users at the sites you specify will be able to modify this project's defect records.

  Note that when the remote site subscribes to this project, that site can have its own local write access security for the project. Determining who should have write access depends on the policies agreed to by each site administrator.

  You should also note that it is not possible to lock records remotely, however it is possible for ClearDDTS to detect when users at different sites have made conflicting changes to the same record. When this happens, ClearDDTS sends mail to the appropriate users explaining that their changes were not made and that a copy of their changes has been saved. The users can then re-submit the changes.

```
Inherit parameters from another project? [Y/N]   Y

Enter the project name: CC
```

- If you want a new project to inherit parameters from an existing project, enter *y* and specify the name of the existing project. Entering an existing project name initializes default answers for all questions that follow. This can save considerable time in repetitive typing. In this example, the compiler project will inherit parameters from an existing project called "CC".

Setting up project notification

One of the most helpful features of ClearDDTS is its closed-loop notification service. Whenever a bug makes a transition *into* a state mentioned below, the users on the notification list will be

notified. For example, at Rational, project managers are on notification lists for New (N) bugs entering the system and Resolved (R) bugs that have been repaired. This notification mechanism helps managers estimate the amount of work remaining on the projects for which they are responsible.

Mail aliases can also be used in place of individual user addresses to send mail to many users at once.

An informative message is printed next, followed by a prompt.

Note: The prompts in this section may be different from what you see. This command should be customized to reflect the state model selected for this project. See Chapter 8, Customizing ClearDDTS for information on customizing your state model.

```
The following questions relate to who to notify when a
record changes state (for instance, moving a record from
open to resolved).

Enter mail addresses of those to notify upon the arrival
of a new bug:
fred bob bill larry!moe!curley!mike fred@thunder
```

▪ When a new bug arrives at this project fred, bob, bill, and mike will be informed by mail of the new bug.

```
Enter mail addresses of those to notify when a new bug is
assigned to an engineer:
```

▪ When a bug is assigned to an engineer, the assigned engineer automatically receives mail about the assignment. Generally, no other users are notified.

```
Enter mail addresses of those to notify when a bug is
opened by the assigned engineer:
mike bill bob
```

▪ When a bug is opened, the assigned engineer acknowledges responsibility for an assigned bug, gives an estimated date of resolution, and will often include a workaround for the problem. In this example, the users mike, bill, and bob will be informed of who has opened the bug and when it will be repaired.

```
Enter mail addresses of those to notify when a bug has
been resolved:
fred
```

- Whenever a bug is resolved, both the submitter and `fred` will be notified by mail.

```
Enter mail addresses of those to notify when a bug fix
has been verified:
```

- Enter the mail addresses of those that should be informed when a bug has moved into the verified state.

```
Enter mail addresses of those to notify when a bug fix has
been postponed:
fred
```

- When a bug fix is postponed, it means that the fix probably will not go out in the next release. This is also likely to be a controversial decision and might require management review. If this decision does require management review, put the manager's mail address on this line.

```
Enter mail addresses of those to notify when a bug has
been declared to be a duplicate of another bug:
```

- When a bug is declared a duplicate, the users on this list will be notified.

```
List LOGIN names of those that have project management
responsibility:
bill
```

- *This is a very important question.* It defines who is allowed to make changes to the project just defined. Enter the project manager's login name. It is also useful to enter the senior engineer's login name. Only users listed as project managers (and the user *ddts*) are allowed to use the *cprj*, *oprj*, *mprj*, or *bprj* commands to modify project information.

```
Are others allowed to subscribe to this project? y
```

- Indicate whether remote machines are allowed to subscribe to this project. If you answer *y* and a remote user subscribes to your project, copies of all the project's bugs are also maintained on that remote system. This allows remote QA organizations to run comparative metrics and predict schedules based on company-wide historical information. These metrics can also be used to estimate staffing needs in support organizations.

Customer support organizations also might need active bug information on your project.

Unless there is a security issue involved, you should answer *y* to this question. We recommend that you do *not* restrict this information.

Whenever a subscription request is honored by ClearDDTS, the project managers and ClearDDTS administrators receive mail about the subscription. If a subscription request is honored and you change your mind, the subscription can be removed by the project manager with the *mprj* command (see below). In addition, selective subscription is also possible. To do this, run the *mprj* command to temporarily allow a remote subscription and then run it again to disallow general subscription. See the *asub* and *dsub* commands below.

```
What Configuration Management System does this project use? clearcase
```

▪ Enter the configuration management system that you are using if any.

Setting up project security

The next sequence of prompts sets permissions on the project database and determines who in the organization is allowed to move defects from state to state, who can modify defects in that state, and who is allowed to manage the project currently being defined. (For more information on write access control, see Chapter 12, Managing ClearDDTS Security.)

The section starts with the following informative message:

```
The following relates to WRITE ACCESS CONTROL.

The following questions relate to who may and may not
modify a bug's state [e.g. resolve bugs, assign bugs, etc].
If ANYONE is allowed to make the state change do NOT enter
a login or group name, just hit RETURN.  If you enter a
login or group name in answer to the questions below, then
any other user or any other group NOT listed will be denied
the ability to make that state change.
```

In general, we recommend that you put as few restrictions as possible on the bug process. However, you must decide for yourself what security or access control requirements make sense for your

needs. We assume that no restrictions are placed on defect submissions, so SUBMIT and NEW are not prompted for here.

Below are the access control questions:

```
List LOGIN names of users allowed to ASSIGN (A state) a
bug. Just hit return if anyone is allowed to ASSIGN bugs.
      mmanley

List GROUP names of groups allowed to ASSIGN (A state) a
bug. Just hit return if any group is allowed to ASSIGN bugs.
      proj_managers
List LOGIN names of users allowed to OPEN (O state) a
bug. Just hit return if anyone is allowed to OPEN bugs.
      $Engineer
List GROUP names of groups allowed to OPEN (O state) a
bug. Just hit return if any group is allowed to OPEN bugs.
      design_team

List LOGIN names of users allowed to RESOLVE (R state) a
bug. Just hit return if anyone is allowed to RESOLVE bugs.

List GROUP names of groups allowed to RESOLVE (R state) a
bug. Just hit return if any group is allowed to RESOLVE bugs.

List LOGIN names of users allowed to VERIFY (V state) a
bug. Just hit return if anyone is allowed to VERIFY bugs.

List GROUP names of groups allowed to VERIFY (V state) a
bug. Just hit return if any group is allowed to Verify bugs.

List LOGIN names of users allowed to declare a bug to
be a DUPLICATE (D state).
Just hit return if anyone is allowed to declare a bug to
be a DUPLICATE.

List GROUP names of groups allowed to declare a bug to
be a DUPLICATE (D state).
Just hit return if any group is allowed to declare a bug to be a
DUPLICATE.

List LOGIN names of users allowed to POSTPONE (P state)
fixing a bug. Just hit return if anyone is allowed to
POSTPONE fixing bugs.

List GROUP names of groups allowed to POSTPONE (P state)
fixing a bug. Just hit return if any group is allowed
to POSTPONE fixing bugs.
```

In this example, the user *mmanley* and users in the group *proj_managers* are allowed to assign defects for repair. They are the only users allowed to move a bug into the Assigned (A) state and the only users allowed to modify a defect once in that state. Only the assigned engineer (value of $Engineer) and users in the group *design_team* may move the bug to the Opened (O) state and only

they may modify defects in that state. For more information on entering users, groups, and variables to control write access, see Chapter 12, Managing ClearDDTS Security.

The ClearDDTS administrator or the project manager can run the *adminbug mprj* command to change any of these parameters.

The above mechanism is the primary way that ClearDDTS implements Write Access Control. See Chapter 12, Managing ClearDDTS Security, for other ways to implement write access control. The next section deals with Read Access Control.

```
The following relates to READ ACCESS CONTROL.

List LOGIN names of users allowed to view a
bug.  Just hit return if anyone is allowed to view bugs.
      mmanley

List GROUP names of groups allowed to view a
bug.  Just hit return if any group is allowed to view bugs.
      design_team
```

In this example, only *mmanley* and members of the UNIX group *design_team* are allowed to view defects. If these fields are left blank, anyone may view defects.

The last message printed by *adminbug* is just informational.

```
Building project this will take a while ...
Updating  DBMS.  This will take a while...
```

Informing the ClearDDTS network of the new project

At this point, you have completely defined the project to ClearDDTS. Depending on the contents of the *export* file (see Chapter 4), other systems may be informed as well. The speed of your communication link with the other systems will determine how quickly they are notified of your project.

To find out if a remote machine has been informed about the new project's existence, have a user on the remote machine use the *projstat* command. Executing *projstat -bl* lists all projects that the

remote machine is allowed to log bugs against and a short description of the project.

If the remote system does not have the project installed or the remote installation failed due to a mail problem, you can rebroadcast the project to the ClearDDTS network with the *bprj* command (see *Broadcast project parameters (bprj)* on page 5-16). This will broadcast the project information again and allow remote machines to submit bugs against the project.

Close a project (cprj)

Use the *cprj* command to close down a project. If a project is finished and no further bugs will be logged against it, it should be closed. When a project is closed, it is taken off the ClearDDTS network so that no one can log bugs against it. However, all of the defect information associated with the project is preserved so that users can later run comparative metrics, look at bugs, or query the database.

The *cprj* command is also necessary if you want to move the project to a different machine. A project must be closed before moving it to another machine.

```
This command will close and remove the project from the
ClearDDTS network such that no one will be able to log bugs against
the project.

Is this what you want to do? (y/n): y
```

▪ Answer *n* to abort the command. Answer *y* to continue.

```
What is the name of the project to remove?
Project name: compiler
```

▪ Enter the project name. In this example, the *compiler* project is being closed. ClearDDTS will inform the network that the project is closed and users will no longer be able to log bugs against it.

```
Please wait, this will take a while ...
```

Delete a project and project data (dprj)

Use the *dprj* command to close a project and *destroy all data for that project*. In general, you should not use this command because it destroys historical data that may be useful for future projects that need comparative data. For this reason, the command starts with a warning:

```
What is the name of the project to remove?
Project name: compiler
```

- Enter the project name. In the example above, the *compiler* project is being deleted from the system.

```
WARNING WARNING WARNING WARNING WARNING WARNING


This command remove the project from the system
and reclaims the disk space.  All project data will be lost

Is this really what you want to do? (Y/N): y
```

- Answer *n* to abort the command. Answer *y* to continue.

```
If there are many bugs this may take a while...
You must now rebuild the ClearDDTS database with the 'dbms' command.
```

After executing this command, rebuild the database with the *adminbug dbms* command.

Open a closed project (oprj)

Use this command to open a project that was previously closed with *cprj*.

```
Enter project name:   DSD.compiler

Updating  DBMS.  This will take a while...
```

The project is reopened and made available to the ClearDDTS network.

Modify project parameters (mprj)

The *mprj* command allows the project manager or ClearDDTS administrator to modify project parameters. For example, you can use this command to add or delete members from the various

notification lists, add or delete project managers, or change the mail path to subscribing machines.

Note: You cannot modify parameters for a closed project.

The *mprj* command asks the same questions as the *aprj* command above and provides the current answer. If no change is desired, just press RETURN to move to the next prompt. To modify any information, backspace over the current answer and type in the correct answer.

For example, to delete *bob* and add *mike* to a notify list, you would press RETURN until you came to the following question:

```
Enter mail addresses of those to notify when a bug has been resolved:  bob
bill ralph
```

▪ Use the BACKSPACE key to delete *bob bill ralph* and type in *bill ralph mike.* Do this for each prompt you want to change.

The only prompt that is different from the *aprj* command is the prompt for the list of subscribers. When you run *mprj* you see the following prompts:

```
Caution, below is a list of subscribers to this project.
Changes to current entries MAY CAUSE LOSS OF
SUBSCRIPTION FOR REMOTE SYSTEMS. It is generally
a bad idea to modify this list.

The following (if any) is a list of subscribing machines
and paths to those subscribers. You might wish to delete a current
subscriber or to change a path to the subscribing machine but DO NOT ADD
ANYONE TO THIS LIST.

If nothing is listed directly below this line just hit
return.

GSDww larry!moe!ddts
DSDqa ddts@thunder
```

In this example, two machines are subscribing to the project. The two machines are GSDww (whose path is larry!moe!ddts) and DSDqa. If you want to delete subscription for one of these machines, delete the machine name *and* the corresponding path. *Do not add*

machines to the list when using this command. Use the *asub* command to add subscriptions.

Important: If you change the class to which the project belongs, ClearDDTS will run the *patchbug* command to update the class field for defects in that project. You then must run the *adminbug dbms* command to rebuild the database with the updated information.

Broadcast project parameters (bprj)

This command should rarely (if ever) need to be used. It is provided only for the rare condition that a project was added via the *aprj* command or opened via the *oprj* command and the mail message about this event was lost to some system(s) on the ClearDDTS network. In this case, the *bprj* command can be used to rebroadcast the project information to the ClearDDTS network. This command is also useful if you have a project that was not exported due to export restrictions (see Chapter 5) and you change the export rules, and then broadcast the project.

```
Enter project name:  DSD.compiler
```

In this example, the DSD.compiler project information will be rebroadcast to the ClearDDTS network. You can safely broadcast project information at any time.

Save a project (sprj)

Use this command to save a project in preparation for moving it to another machine or for archival purposes. The command saves all of the project bugs and project-related files into the directory *~ddts/projects/<projname>* where *<projname>* is the name of the project. When this command completes, you can use the UNIX commands *tar* or *cpio* to copy the *~ddts/projects/projname* directory and move it to another machine.

When executed, the *sprj* command issues the following prompt:

```
Enter project name:  DSD.compiler
```

- Enter the name of the project to be saved. After you identify the project, ClearDDTS lists the bug IDs that have been logged against the project and are being saved.

Now it is possible to save the project to a tape. Exit from *adminbug* and do the following while still logged in as *ddts*:

```
cd ~ddts/projects
tar cvf /dev/xxx projname
```

The *xxx* is a backup device such as a magnetic tape device, and *projname* is the name of your project, such as GSD.compiler. As *tar* runs, you should see the bug files being saved.

After archiving a project to tape, you can run the *adminbug dprj* command to reclaim disk space, if desired.

Restore a project (rprj)

Use this command to restore a project from a tape archive. Before running this command, you must first restore the project directory that was saved (see *sprj* above) via *tar* or *cpio*.

To restore a project:

1 Enter the following:

```
cd ~ddts/projects
tar xvf /dev/xxx
```

The *xxx* is the tape device. This places all of the project data into the appropriate directory.

2 Run the *rprj* command.

```
Enter current project name:  DSD.compiler
Enter new project name:  SFD.compiler
```

- Enter the name of the project to be restored. In this case, the DSD.compiler project is being restored and renamed with the site ID of the new machine, SFD. (The current project name and the new project name can be the same, if desired.)

The command will notice the new project and install all of the project bugs and project-related files into the local system's ClearDDTS database.

3 Rebuild the ClearDDTS database with the *adminbug dbms* command.

Rename a project (renm)

Use this command to rename a project. In general, a project should be renamed only when absolutely necessary. Other people on the ClearDDTS network are already aware of the project name and renaming the project may cause confusion, especially in remote sites. Here is the dialog:

```
Enter current project name:  compiler

Enter new name for project:  C_compiler
```

ClearDDTS sends a message to the entire ClearDDTS network to update the machines that need to know the new name.

Ask to subscribe to a project (asub)

Use this command to subscribe to a project from a remote machine. Assume that machine A owns a project that you want to subscribe to from machine B. The *asub* command causes all of the bug information for the project on machine A to be shared (replicated) on machine B. You run this command from the machine that wants the subscription (machine B), not the machine that owns the project (machine A).

When the command is executed, a request goes to machine A and is accepted or rejected depending on whether machine A has allowed general subscription. You and the project managers are informed by mail of the success or failure of this command.

If the command fails because machine A is not allowing general subscription, you must verbally negotiate with that ClearDDTS administrator for subscription rights. A limited subscription is available by having the administrator of machine A execute a *mprj*

command and temporarily allow subscription long enough for your *asub* request to be processed. The machine A administrator can then run the *mprj* command again to turn off subscription.

```
Enter the project name you wish to subscribe to: compiler
```

- Enter the project name. If you don't know the project name, abort this operation using your interrupt character (usually ^C or DEL), and use the *lbug* command to display a list of all projects that machine B knows about, along with a short description of each project. If the project you want to subscribe to is not on the list, one of the following is wrong:
  - The project is not being imported by machine B from machine A. Check *~ddts/conf/import* file for machine B.
  - The project is not being exported to your system by machine A. Check the *~ddts/conf/export* file for machine A.
  - Machine B is not connected (with the *conn* command) to the site with the project (machine A).

    If you are connected to the site, you (or the ClearDDTS administrator for machine A) need to change the *import* file, the *export* file, or both for the *asub* command to succeed. See Chapter 4, Managing Remote Access Between Multiple Installations, for more information.

After entering the project name, you are prompted for the person to be notified when certain state transitions have been completed. These prompts are similar to those for the new project (*aprj*) command.

If this project allows the remote site (machine B) to modify defects (determined when the project was created), you are prompted to set up project security on the subscribing site. This consists of establishing who has write and read access to the project. The questions are similar to those for the new project (*aprj*) command. The result is that each site will have its own ~ddts/projects/<project_name>/proj.control file. The project security can be the same for both sites or customized for each. Project security can be modified using the *adminbug msub*

command for subscribed sites. This is only true when remote modification is turned on.

When you answer the last prompt, ClearDDTS sends a request to machine A to configure machine B as a subscriber. A *refreshbug* command is also executed and all project bugs on machine A are copied to machine B. All future bugs submitted against machine A will also be kept up to date on the subscribing machine B.

Delete subscription to a project (dsub)

Use this command to notify a remote machine that you no longer want to subscribe to a particular project.

```
Enter project name that you wish to delete subscription to: compiler
```

▪ Enter the project name. After the command is executed, no more project bugs will arrive from the remote project machine. However, no data on the local machine is destroyed, in case it is needed for metrics purposes. To remove the data from your system, you can use the *dprj* command or the *rmbug* utility after deleting your subscription.

Modify subscription parameters (msub)

Use this command to modify the notification lists for a project. You can also modify security options for subscribed projects where remote modification is turned on.

```
Enter the name of the project to which you are subscribing
compiler
```

▪ Enter the project name.

After entering the project name, you are prompted for the person to be notified when certain state transitions have been completed. These prompts are similar to those for the new project (*aprj*) or

subscription (*asub*) commands. Make your changes to the notification and security lists as appropriate.

List all project parameters (lprj)

This command lists all of the project parameters to the screen. A sample dialog and response is shown below:

```
Enter project name: 386_port

Proj-name: 386_port
Proj-desc: Port of operating system to 386i.
Part-no: 98007-13400
Proj-scope: public
Path-to-owner: SFDww ddts
Templates: templates/default
Proj-status: active

N-notify: mmanley
A-notify:
O-notify:
R-notify:
E-notify: mmanley
V-notify:
D-notify:
P-notify: mmanley

Proj-mgrs: mmanley
Allow-subs: Y
Status:
A-allow: mmanley
A-allow-group:
O-allow:
O-allow-group:
R-allow:
R-allow-group:
E-allow:
E-allow-group:
V-allow:
V-allow-group:
D-allow:
D-allow-group:
P-allow:
P-allow-group:
Subscribers:
```

The data displayed is the information provided when the project was first created.

List project names and descriptions (lbug)

This command lists the names of all of the projects in all classes that users on this system can log bugs against, along with a brief

description of each project. Closed projects are not listed. For example:

```
compiler       C compiler project.
4.3os_port     Port of the 4.3 BSD operating system.
dbms_3.3       Revision 3.3 of the database system.
```

List projects owned on this machine (lown)

This command lists all of the projects in all classes that are owned on this system. A one line description of the project is also listed. These are the projects that were created on this machine with the *aprj* command. For example:

```
compiler       The C compiler project.
4.3os_port     Port of the 4.3 BSD operating system.
```

List projects being subscribed to on this machine (lsub)

This command lists the projects owned on other machines that are being subscribed to on this machine. A one line description of each project is also listed. For example:

```
audit The C2 secure auditing project.
gsd.firmware    All firmware for GSD hardware.
```

6

Reconfiguring a ClearDDTS Network

You need to reconfigure your ClearDDTS network if you do any of the following:

- move a project to a different machine
- physically move a machine to a different location
- move the entire ClearDDTS database to a new machine

Reconfiguring your network does not require any new administrative commands, but relies on a combination of commands you already know. This chapter describes how to perform these configuration changes.

Moving a project

To move a project from one machine to another, you must inform ClearDDTS so that it can inform the network.

Note: You must inform ClearDDTS *before* the move (steps 1 through 8).

To move a project:

1 Log in as the user *ddts*.

2 Run *adminbug*.

3 Close the project with the *cprj* command. Leave the machine running long enough for all outgoing mail to be sent. (Give ClearDDTS time to send mail to other connected machines telling them that the project has been closed. The amount of time required depends on your environment.)

4 Use the *sprj* command to save the project. You should see a list of the defect IDs that are being saved displayed on the terminal.

5 Exit *adminbug*.

6 Save the project to tape by doing the following (logged in as *ddts)*:

7 `cd ~ddts/projects`

8 `tar cvf /dev/xxx <projname>`

Here *xxx* is a backup device such as a magnetic tape drive, and *<projname>* is the name of your project, such as *GSD.compiler*. As *tar* runs, you should see the bug files being saved.

9 Ensure that the tape is good by restoring the tape to some temporary directory.

10 Use *adminbug oprj* to reopen the project.

11 Use *adminbug dprj* to delete the project data from the old system and reclaim the disk space. You must delete the project from ClearDDTS before restoring it on another machine.

12 Use the *adminbug dbms* command on the machine where the project was deleted to rebuild the database.

13 Take the tape to the machine that will be the new home of the project, log in as *ddts*, and do the following:

14 `cd ~ddts/projects`

15 `tar xvf /dev/xxx`

16 On the new home machine, run the *adminbug rprj* command to open the project and install the bugs in the ClearDDTS database.

17 Use the *adminbug dbms* command to rebuild the database on the new machine, adding the new project's bugs.

18 Inform subscribers to the moved project that they need to reissue their subscriptions.

Physically moving the machine to a new location

If you are moving a ClearDDTS machine to a new location, you need to determine whether e-mail addresses (for remote machines and local users) will still be valid after the move.

If e-mail addresses are not valid after moving

To see if e-mail addresses will be valid, issue the *adminbug lsit* command. If this command lists any UUCP style mail addresses or if the e-mail address listed will not be valid after the move, follow the steps below:

1 Log in as *ddts*.

2 Run *adminbug*.

3 Issue *dsub* commands for all subscribed projects.

Note: Running the *dsub* command from your system removes subscriptions you have to projects on other systems, however, it does not remove subscriptions that other systems might have to projects your system. Be sure all subscriptions, to and from your system, are removed.

4 Optionally, save a copy of the `~ddts/conf/submit.sites` file. This file lists all sites connected to your system, but it is removed when you issue the *dcon* command (see the next step). Saving a copy of the file allows you to refer to it later for the list of sites to which you need to reconnect.

5 Issue *dcon* commands against all connected sites.

6 Issue a *dsbl* command to remove your system from the ClearDDTS network.

7 Allow the mail system to transmit messages.

8 Move your machine to the new location.

9 Set up mail at your new location (if necessary).

10 Run *ddtsinstall* to open all closed projects, put you back on the ClearDDTS network, and rebuild the database.

11 Run *adminbug conn* to reconnect your sites (if necessary), referring to the saved copy of the *submit.sites* file.

12 Reissue subscriptions to the projects from which you desubscribed in Step 3.

13 Inform the administrators on all connected sites that they can reissue subscriptions to projects on your site.

If e-mail addresses are still valid

If the *adminbug lsit* command did not list any remote machines or if those e-mail addresses will still be valid after the move, then go ahead and move the system by performing steps 4 - 9 above.

Checking addresses after the move

After the system is installed at the new location, you look at the mail addresses of all the defects on the system. This can be done by issuing the following commands:

```
grepbug  Engineer-mail   >> /tmp/foo
grepbug  Submitter-mail  >> /tmp/foo
grepbug  Other-mail      >> /tmp/foo
```

Note: These commands run very slowly. You should run them in a shell script and come back later to look at the result.

After the commands have completed, look at the result with an editor and see if there are any UUCP addresses or addresses that are no longer valid in the new location. There should be few, if any, illegal addresses. If you find an address that needs repair, use *patchbug* to repair the defect. (See the *patchbug* man page for more information.)

Moving the ClearDDTS database

The procedure for moving a ClearDDTS database is similar to the procedure for physically moving a ClearDDTS machine. Follow these steps:

1 Log in as *ddts* on the current machine.

2 Run *adminbug*.

3 Issue *dsub* commands for all subscribed projects.

Note: Running the *dsub* command from your system removes subscriptions you have to projects on other systems, however, it does not remove subscriptions that other systems might have to projects your system. Be sure all subscriptions, to and from your system, are removed.

4 Optionally, save a copy of the `~ddts/conf/submit.sites` file. This file lists all sites connected to your system, but it is removed when you issue the *dcon* command (see the next step). Saving a copy of the file allows you to refer to it later for the list of sites to which you need to reconnect.

5 Issue *dcon* (disconnect) commands against all connected sites.

6 Issue a *dsbl* (disable) command.

7 Allow the mail system to transmit messages.

8 Enter the following commands:

9 `cd ~ddts`

10 `tar cvf  /dev/xxx .`

The *xxx* is a backup device such as a magnetic tape drive and '.' is the current directory (*~ddts*). As *tar* runs, you should see many bug files and projects being saved.

11 Take the tape to the new machine and go through the standard installation process described in the *ClearDDTS Installation and Licensing Guide*, with the exception that you will run *ddtsinstall* in the current directory (rather than in the NEWCLEARDDTS subdirectory.)

12 Set up mail at your new location (if necessary).

13 Run *adminbug conn* to reconnect your sites (if necessary), referring to the saved copy of the *submit.sites* file.

14 Reissue subscriptions to the projects from which you desubscribed in Step 3.

15 Inform the administrators on all connected sites that they can reissue subscriptions to projects on your site.

16 Remove the system *crontab* entries on the old system. On most UNIX systems, you can do this by logging in as *ddts* and issuing the following command:

```
crontab -r
```

7

Understanding the Master Template File

One of the most important template files in ClearDDTS is the *~ddts/class/<classname>/master.tmpl* file. This file defines the rules for moving from state to state by describing the interactive dialog and data requested to make a state transition. This *master.tmpl* file controls all of the prompting, screen formatting, and terminal processing in ClearDDTS. If you are adding a new state or changing the screen display in any way, you need to modify this file. To learn how to make these customization see Chapter 8, Customizing ClearDDTS.

This chapter covers the following topics:

- Example master.tmpl file
- Understanding the "Begin" field derivation section
- Other field derivations
- Understanding OPERATION and STATE
- How OPERATION and STATE are used
- Most common derivation
- Setting default values
- How webddts pages are generated

Example master.tmpl file

The overall structure of the file is a list of ClearDDTS *field names*, with each field name followed by a group of *derivation* lines. The derivation lines determine the contents of the named field. The example below is an excerpt from the *master.tmpl* file. In this example, *When-found* is a field name and the section of code from *if match $Operation v m* to the second *fi* are the derivation lines for

that field. These derivation lines control the information that is
posted to the database.

```
Begin:       unset Begin
             set Filter-path /usr/bin:/bin:/usr/ucb
             set Oneof-path class/$Class/oneofs
             set Help-path class/$Class/helps
             .
             .
             .
When-found:  if match $OPERATION v m
             or match $STATE$OPERATION Sp Sf
                   "\(8,1)Detected in phase: %-21.21s"
             fi
             if match $OPERATION p m
                   help when-found.H
                   oneof -f phases
                   required
             fi

OS-version:  if match $OPERATION v m
             or match $STATE$OPERATION Sp Sf
                   "\(11,1)Version of OS: %-21.21s"
             fi
             if match $OPERATION p m
                   help os-version.H
             fi

Severity:    if null
                   echo 3
             fi
             if match $OPERATION v m
             or match $STATE$OPERATION Sp Sf
                   "\(12,1)Problem severity: %1.1d"
             fi
             if match $OPERATION p m
                   help severity.H
                   oneof 1 2 3 4 5
                   required
             fi

Submitter-org: if null
                   cat $~/conf/myorg
               fi
               if match $OPERATION v m
                   "\(17,1)Organization: %-22.22s"
               fi
               if match $OPERATION p m
                   help org.H
                   required
               fi
```

Each of these groups is called a *field derivation*, because it
describes how the value of the defect record field is derived.

Understanding the "Begin" field derivation section

The first four lines in the fragment above define information that is used by the rest of the template file for defect field derivation. In the first line:

```
Begin:    unset Begin
```

the field name is *Begin* and the *unset* command says not to include this field in the defect record. This field is not an actual defect field but is used to set up parameters that are used by other field derivations.

The next line:

```
set Filter-path /usr/bin:/bin:/usr/ucb
```

says that user written programs or shell scripts that are not built-in ClearDDTS commands can be found in the *Filter-path* directories listed. This works exactly like the Bourne shell PATH environment variable. Any command not listed as a built-in command in the *template(5)* man page would be searched for and executed if found in one of these directories. The user's current PATH is also searched for the filter commands.

The next line:

```
set Oneof-path class/$Class/oneofs
```

defines the directories where the *oneof* built-in command can find a file with a list of acceptable answers to a field derivation. If a question has a limited set of acceptable answers, then this line defines where to find a file with those answers. Note that this file is Class (*$Class*) specific.

The last line of the *Begin* field derivation:

```
set Help-path class/$Class/helps
```

defines the directories where the field-level help files reside. These files are used when the *help* built-in command is executed (that is, when a user enters a *?* in response to a prompt in *xddts* and *bugs*, or clicks the field-level help icon in *webddts*).

Other field derivations

The first field derivation is a *pseudo-field* that does not become part of the defect record. The next four fields in the example do become part of the defect record. When executed, these field derivations generate or replace the values of four defect data fields named *When-found*, *OS-version*, *Severity,* and *Submitter-org*. See Appendix A, Contents of a Defect Record**,** for the meaning of these and all other defect record fields.

At this point it may help you to look at the *~ddts/class/software/master.tmpl* yourself. As you review the contents of the file, read the *template(5)* man page. Do this before going on to the next section.

Understanding OPERATION and STATE

There are two special preinitialized fields in the *master.tmpl* file. These fields are *OPERATION* and *STATE*. You need to understand how these two fields are used to understand the system.

Every field derivation in the *master.tmpl* is surrounded by *if* statements. These statements permit a single field derivation to be used for multiple purposes. The programs executing the *master.tmpl* do this by preinitializing the *OPERATION* and *STATE* field names before the template file is executed.

There are several programs that use the value of the *OPERATION* field to determine their behavior: *webddts, xddts, bugs, bugmail,* and *dumpbug*.

The *webddts*, *xddts* and *bugs* interfaces are interactive and are used to perform ClearDDTS state transitions. The *OPERATION* values used are:

Value	Meaning	Mode
f	Display empty form before prompting. (*xddts* and *bugs* only)	Output
p	Prompt for state change (move to a new state).	Input

Value	Meaning	Mode
m	Modify fields without changing state.	Input
v	View defect record (just print it, enclosures not printed).	Output

The *bugmail* utility is the program that sends notification mail. The only *OPERATION* letter used by *bugmail* is:

Value	Meaning for bugmail	Mode
n	Process the master.tmpl file for notification mail. See chapter 11, Handling ClearDDTS Mail for more information on notification mail.	Output

The *dumpbug* utility is the program that formats and prints a defect report to the standard output. The *OPERATION* letters used by *dumpbug* are:

Value	Meaning for dumpbug	Mode
l	Show defect record including enclosures.	Output
v	View defect record (enclosures not printed).	Output

The other preinitialized field is the *STATE* field. When you perform state transitions, ClearDDTS sets the *STATE* field to the *next target state* (New, Assigned, Open, etc.) of the defect (not the current defect status). Therefore, the *STATE* field identifies the new state to which the bug is being moved.

How OPERATION and STATE are used

To understand how ClearDDTS executes the *master.tmpl* file, consider the following example. Suppose you want to move a defect from the *Open* state to the *Resolved* state. When you perform a state transition, ClearDDTS executes the *master.tmpl* file three times—each time with a different preinitialized value of *OPERATION* (f, p, and then v) and the same value of *STATE*:

1 First, ClearDDTS executes the template file with an *OPERATION* of *f* and *STATE* of *R*. This operation displays the portion of the form

that must be filled in to move the defect to the *R* Status, and displays all the fields that have previously been supplied data. The values of these Resolution fields are not prompted for yet, and no input is required at this point.

2   Next, ClearDDTS executes the template file with an *OPERATION* of *p* and a *STATE* of *R*. This operation positions the cursor in the Resolution area and prompts for an input value for each Resolution field.

3   After the data requested in step 2 has been supplied, ClearDDTS executes the template one last time to display (view) the final results by executing the template file with an *OPERATION* of *v* and a *STATE* of *R*. This operation displays the defect report in the newly entered Resolved state, with all of the old and new information filled in.

The example above illustrates a state transition. Modifying a defect record is similar except that only two executions are performed with *OPERATION* set to *m* and *v*, and the *STATE* is set to the current *Status* of the defect. The following tables summarize the use of OPERATION and STATE:

Defect State Change Summary

Pass	*OPERATION*	*STATE*	*Mode*	*Purpose*
1	f	future status	Output	Paint form to be filled in.
2	p	future status	Input	Prompt for input.
3	v	future status	Output	Display defect in new state.

Defect Modification Summary

Pass	*OPERATION*	*STATE*	*Mode*	*Purpose*
1	m	current state	Input	Prompt for input.
2	v	current state	Output	Display defect in new state.

Note: You do not need to initialize *STATE* or *OPERATION*. These fields are initialized automatically by *webddts, xddts*, *bugs*, *dumpbug*, and *bugmail*. Any customization in the template file should use the

values of these fields to determine which derivation lines should be executed. Thus, the same template fields display different screens depending on the state and purpose described by *STATE* and *OPERATION*.

Understanding this *STATE* and *OPERATION* preinitialization mechanism is very important. You may want to read this section again before customizing the *master.tmpl* file.

Most common derivation

To begin customizing the *master.tmpl* file, consider the most common field derivation form:

```
When-found:  if match $OPERATION v m
             or match $STATE$OPERATION Sp Sf
                   "\(8,1)Detected in phase: %-21.21s"
             fi
             if match $OPERATION p m
                   help when-found.H
                   oneof -f phases
                   required
             fi
```
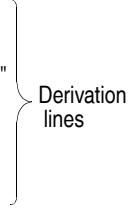
Derivation lines

You can use this simple derivation as a basis for any customized fields that you add. In this derivation, line three is executed if:

▪ the *OPERATION* is view (*v*) or modify (*m*), or

▪ you are prompting for bug submission information (*Sp*), or

▪ you are painting the bug submission form (*Sf*)

This derivation allows the field to be changed when going to the *STATE* specified (*S*) or when modifying all of the defect fields.

A closer look at derivation lines

The code in a field derivation is similar to a shell script and is composed of conditional statements, commands to be executed, and prompt strings. The commands can be ClearDDTS built-in commands, UNIX utilities, or your own custom programs.

The first line of the *When-found* derivation is repeated frequently. The *if* command uses the *match* operation to test whether its first argument (*$OPERATION*) matches any of the remaining arguments.

The next line uses *or* to add more conditions under which the *if* condition is true. In this case, the enclosed lines will also be executed if the *STATE* is *S* and the *OPERATION* is *f* or *p*; that is, when the template is being used for painting an empty form or prompting for a state transition into the *S* state (the initial *Status* when a bug is first submitted).

If the *if* statement is true, the next line that begins with a double quotation mark is executed. The quotation indicates that the line is a prompt string which is to be displayed on the screen. The special sequence *\(8,1)* moves the cursor to row 8, column 1. The characters *"Detected in phase:"* are then displayed. The sequence *%-21.21s* causes the program to do one of two things:

- If the derivation is being executed in output mode (*$OPERATION* of *f* or *v*), the current value is printed and the next line is executed.

- If the derivation is in input mode (*$OPERATION* of *p* or *m*), then the current value is printed and execution pauses at that point and waits for the user to enter a string up to 21 characters in length. The user may backspace over the current value, enter a new value or just press RETURN. The resulting string will be stored as the current value of the *When-found* field.

The specifiers beginning with *%* and sequences such as *\n* are used exactly as they are used in the *printf(3S)* or *scanf(3S)* functions of the C programming language.

Note: Dates in ClearDDTS are stored in yymmdd format. Date input fields should be specified as *%6.6s* not as *%6.6d*, especially for dates past the year 2000. For example, the date January 1, 2000 is stored as 000101, but if it is input as a number instead of a string, the leading zeroes are dropped.

Note that the format string is used for both input and output. For output, the sequence *%-21.21s* specifies that the output value should be left-justified and no more or less than 21 characters (truncated or padded with spaces, as necessary). For input, the sequence *%-21.21s* means accept input of exactly 21 characters. Using fixed width output helps keep the screen organized so that a field is always displayed in the same area of the screen.

The remaining derivation lines of the *When-found* field definition are executed only when the template is being used in input mode (*p*rompting or *m*odifying).

Note that you can rearrange the cursor motions so that the output is in whatever form you want. However, putting new output at the same screen location as the previous output, silently covers up the characters "underneath", so planning the desired layout beforehand can help save a lot of trial and error time.

Note: The *webddts* interface does not use the cursor addressing directives. For more information see "Specific webddts customizations" on page 8-21.

The next line of the template:

```
help when-found.H
```

is for context-sensitive help. It says that if the current field value consists only of the single character *?* (*xddts* and *bugs*) or the field-level help icon is clicked (*webddts*), then the contents of the file *when-found.H* should be displayed. Recall that the help file directories were defined earlier by the line:

```
set Help-path class/$Class/helps
```

So, in this case, the contents of the file *~ddts/class/<classname>/helps/when-found.H* will be displayed. After the user has looked at the help message, this field is executed again from the beginning of the derivation. In *xddts* and *bugs*, if the field value is anything other than a *?*, the *help* derivation line has no effect.

The next line of the *When-found* field derivation:

```
oneof -f phases
```

says that the data entered for the field must exactly match one of the lines in the file *phases* (or have a unique partial match). The location of the *phases* file was defined earlier by the line:

```
set Oneof-path class/$Class/oneofs
```

The file with the choices to be matched is *~ddts/class/<classname>/oneofs/phases*. If a matching line is found in the file, the value is changed to the full contents of that line. If there is a successful match, this field is completely "derived," and ClearDDTS continues on to the next field in the template.

By default, the user is not required to enter any characters other than a carriage return in response to the prompt. The last line, *required,* checks to see that there actually is a value in the *When-found* field. If there isn't, this line "fails," and a message saying that a response is required is printed. The program then starts over at the beginning of the derivation field. If there is something in the input, the program proceeds to the next line.

The derivation of the *OS-version* field is much the same, except that this field can be left empty (because there is no *required* line), and can contain any text (because there is no *oneof* line to require anything special).

The *Severity* field derivation requires a numeric valued input of up to one character in length (*%-1.1d*). Non-digit characters are impossible to enter, so they need not be checked by later derivation lines. Since there is a *required* line, a numeric digit must be entered to satisfy this prompt. This field also has a pre-set default value. If the field has an empty string (*""*) as its current value, then its value is set to 3. This is convenient if most submitted defects are of severity 3.

The *Submitter-org* field is unlike the previous fields in that the prompt string is not executed when painting a form or initially prompting for a state transition, although it is executed when

modifying any defect. This permits the value to be added automatically when submitting (since it seldom changes), but permits it to be changed if the user desires.

When a defect is originally being entered, the value of *Submitter-org* is silently set to the output of the command *cat $~/etc/myorg*. The *$~* is a ClearDDTS metacharacter that expands to the path of the ClearDDTS home directory.

This line could have been coded as */bin/cat `ddtshome`/etc/myorg*. This command would actually first run the *ddtshome* command to locate the home directory of *ddts*, and then run the UNIX *cat* command to retrieve the contents of the *~ddts/etc/myorg* file. You can use this capability to generate field values using UNIX commands, shell scripts or other external programs.

There are also other template metacharacters that you may use. They are:

Meta Character	Value
$Foo	Value of the Foo field (or enclosure title).
$Foo:n	Value of the *n*th word of Foo.
$$	Current value of current field being executed.
$n	Current value of *n*th word of current field being executed.
$:n	Current value of *n*th word of current field being executed.
$!	Length of the current field (or enclosure text) in bytes.
$%	Name of a temporary file containing enclosure field text.
$&	Value of an enclosure field's timestamp.
$~	Home directory of ClearDDTS (*~ddts*)
$@	Current user's login ID.

Setting default values

Many customers want to set up default values for fields. The following example illustrates how a default detection phase is assigned when submitting a defect:

```
When-found: if null
                echo beta test
            fi
            if match $OPERATION v m
            or match $STATE$OPERATION Sp Sf
                "\(8,1)Detected in phase: %21.21s"
            fi
            if match $OPERATION p m
                help when-found.H
                oneof -f phases
                required
            fi
```

In this example, *if null* insures that the *echo* command is only run if there is no current value for the field. The *if null* is very important. This makes sure that you do not overlay any value that has been previously entered and if the field has no current value, it sets one.

Do not be concerned by the *OPERATION* codes of *v* or *m*. This is to make sure that in the viewing mode, the field is displayed and in the modify mode that the value can be modified. Recall that what you are doing here is setting up a default value for the defect detection phase.

The *webddts* interface parses the *master.tmpl* file differently from the other ClearDDTS interfaces. Rational has developed a technical white paper discussing these differences and how they effect making customizations to *webddts*. Included in the white paper is a discussion on using complex default expression in field derivations. See "Making the Move to ClearDDTS 4.x" at *www.rational.com/sitewide/support/whitepapers*, for more information.

Defaults in *xddts*

There are three ways to set up defaults in *xddts*:

- Each user may modify the *.ddtsrc* file in his or her home directory
- You can edit the *~ddts/etc/ddtsrc* file for system-wide defaults.

- You can use environment variables on the fly or in your login files (for example, *.cshrc*).

The precedence is environment variables, the *.ddtsrc* file, and the *ddtsrc* file. Each method is described below.

If you are using *xddts*, you can set up a default in the user's *.ddtsrc* file. Every user has a *.ddtsrc* file that is automatically created by ClearDDTS. This file can be used to set up default selections (for example, the default projects and states to use when you perform a search in *xddts*) and default field values (for example, a default project or phone number). For example, the *.ddtsrc* may look something like this:

```
Class: software
Gui_Prn_PrintAll: 1
Gui_Srch_Class: software
Gui_Srch_States: NAO
Gui_Srch_Sort_Enabled: 1
How-found: random unplanned test
When-found: beta test
Test-name: xddts
Test-system: dart1
OS-version: SunOS 4.1.3
Submitter-phone: (209) 924-9000
Engineer: smith
Problem: design
Recommend-change: design
When-caused: design
Analyze-hours: 1
Est-fix-hours: 1
When-fixed: alpha test
Version: 4.1
Project: Graphix
```

In addition to setting up defaults for the fields listed, you can use this file to define certain characteristics of your graphical environment, such as the format of the index. Although most of these variables are defined automatically when you make selections, you can also edit this file manually or use the *setdsrc* command to set field values as desired. See the man page for *setdsrc* for more information.

While the user's *.ddtsrc* file can set up defaults for individual users, you can also set up system-wide defaults by editing the *~ddts/etc/ddtsrc* file. This file has exactly the same format as above, but sets defaults for all ClearDDTS users.

The last way to set up a default is with UNIX environment variables. For example, suppose that you set up an environment variable called *When-found* and set the value to *alpha test*. In the example above, the *if null* derivation would fail because ClearDDTS implicitly found a value for When-found of *alpha test*. In this case, the value *alpha test* would be presented as a default.

As an example of how variables work, run the following:

```
bugs -c
```

Look at the form presented. Now type CONTROL-C to close the form. Then run:

```
setenv Severity 2
setenv When-found "alpha test"
bugs -c
```

Note how the new form presents the new default values.

Defaults in *webddts*

You can create field defaults in *webddts* from the User Profile page. Setting field defaults allows you to predefine values for specific fields when querying, submitting or modifying defects. There are two ways you can set default values for a field:

- enter a default value: You can enter a default value for any field in the database.

- make the field *sticky*: You can make a field retain the last value you entered and use that as the default for future submissions or queries. The value you enter in effect *sticks* to that field until you change it.

If you enter a default value for a field that is normally system generated, such as Last Modified, that value is only used as the default for queries on that field. When submitting or modifying a defect, ClearDDTS ignores any defaults set for a system defined field.

How webddts pages are generated

The *webddts* interface uses Hypertext Markup Language (HTML) to display ClearDDTS pages in a web browser and to receive input from the user. The fields to display are determined by scripts and utilities that interpret the *master.tmpl* file. This section provides an overview of how these utilities and scripts allow the *webddts* interface to interpret the *master.tmpl* file.

Note: Any changes you want to see in the *webddts* interface should be done in the *master.tmpl* file. There is generally no need to edit the scripts used to generate the HTML or the programs used to interpret the *master.tmpl* file.

Web page generation—the big picture

The utility used to interpret the *master.tmpl* file is the *wtform* program. The *wtform* program receives input from *webddts* CGI scripts (action, state, class information), and uses that information to process the *master.tmpl* file. The field attributes for relevant fields are generated and passed back to the *webddts* CGI scripts which generate the HTML necessary to display the requested page.
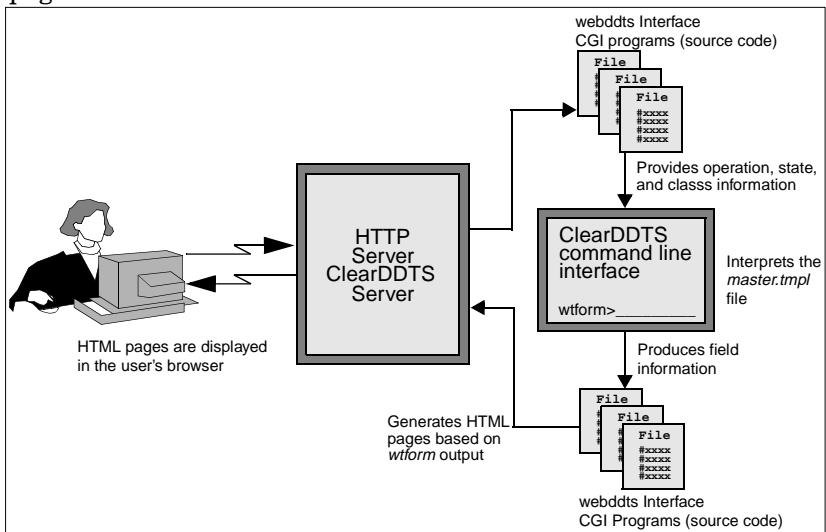
A CGI script is an executable program that conforms to the Common Gateway Interface standard for interfacing external applications with information servers. In simple terms, CGI programs crete a gateway through which information can pass from an external application, such as ClearDDTS, to a Web server where it can be displayed to a client or Web browser. In reverse, information can be sent from the Web server to the external application.

Updating the database

The process for updating the database is similar to generating a web page. When a user submits or modifies a defect, the *master.tmpl* file is interpreted and template files are created for the database updates.

The utility used to interpret the *master.tmpl* file for database updates is the *wttmpl* program. The *wttmpl* program receives input from *webddts* CGI scripts (action and state information) and user input, uses that information to process the *master.tmpl* file, creates a template file (containing the updated fields, values and validation logic), and passes the template to the *batchbug* program. The *batchbug* program uses the template information to update the database in the *allbugs* directory.

Restrictions—what is not interpreted

The web page generation cannot accommodate unusual customizations you might make to the field derivations in the *master.tmpl* file. This section describes the type of derivation that works well with the *webddts* interface, and describes some of the non-standard derivations that will not work.

Note: Most of the non-standard derivations will work with the *xddts* and *bugs* interface. If you have already customized your *master.tmpl* file for use with the other interfaces, refer to "webddts specific customizations" on page 7-18 to learn how to preserve these customizations and create web specific customizations.

Standard field derivation

A field derivation is usually divided into three parts:

- the expression to compute the default value
- the "prompt" string (where the actual user input is performed)
- the expression for the validation

Each of these three portions often have similar constructs. For instance, the default value is often a static variable set using the "echo" filter. The validation often contains a static oneof (choice list) or "required" filter. The prompt string is usually very structured with a location, label, and `%s` for the input/output.

Non-standard field derivations

The *master.tmpl* file becomes difficult to interpret for HTML when a customization deviates from the standard derivation form. In particular, the following cannot be interpreted:

- conditional prompts or *goto* statements

  Conditional prompts based on data from a source other than the *master.tmpl* file cannot be handled for HTML. For example, assume you want the Severity prompt to appear if a certain project is entered, otherwise you want the Priority prompt to appear. Since the form must be generated before any input is done (and no project is selected) no default value can be found. *webddts* would simply display the first prompt found by *wtform*, in this case Severity.

  As with other conditionals, conditional *goto* statements used with any options other than OPERATION or STATE cannot be supported if the condition depends on data external to the *master.tmpl* file.

- the *interact* command

  In the *xddts* interface the *interact* command launches an xterminal window running an application external to ClearDDTS (usually a bourne shell script or C program executable). However, on the web there is no way to start an

external application in a window on the local machine. Therefore, the *interact* command is not supported in the *webddts* interface.

▪ multiple input prompts

In the *xddts* interface, the user can be prompted multiple times for the same field to provide different pieces of data. The result is a single field updated in the database. Web page generation and validation make this impossible, therefore, the *webddts* interface will only provide the first prompt for such a field.

webddts specific customizations

To provide values that cannot be provided through non-standard field derivations, and to preserve some of those field derivations for use by *xddts*, you can use *webddts* specific code in the *master.tmpl* file. Code contained within specially formatted sections of code are ignored by other ClearDDTS interfaces while allowing you to provide different or extra data for *webddts*.

The *webddts* specific code is indicated by using an if www statement. With this format you can indicate what field derivation should be used if the interface is *webddts*, otherwise use the existing field derivation. For example, to create an appropriate field label for the Version field, we have used the following code:

```
Version:
                if www
                    "\(1,40)Version: %-8.8s"
                fi
                if match $OPERATION v m
                    "\(1,30)$Software, version %-8.8s"
                elif match $STATE$OPERATION Sp Sf
                    "\(1,40)Version: %-8.8s"
                fi
                if match $OPERATION p m  # "if input" would work, too
                    help version.H
#                   oneof -f version
                    required
                fi
```

8

Customizing ClearDDTS

There are many ways to customize ClearDDTS. For example, you may want to add or delete defect states, modify a screen, add additional pages of information to the display or change the state transitions. This chapter discusses these types of customization and others. If you are the ClearDDTS administrator, read this chapter carefully and refer to it whenever you are making changes to your system. The following topics are covered:

- Before making changes
- Locating files to customize
- Adding new fields
- Adding defect states
- Further template customization
- Specific webddts customizations
- Specific xddts customizations
- Debugging a custom template file

Note: This chapter focuses on changes to the user interface. For information on the *master.tmpl* file see Chapter 7, Understanding the Master Template File. For information about customizing reports, see Chapter 9, Creating Custom ClearDDTS Reports. For information about modifying the database, see Chapter 13, Managing and Customizing the ClearDDTS Database.

Before making changes

At the lowest level, ClearDDTS is a finite state machine where you can define the defect states and the rules for moving defects from state to state. Because you can define the states and the state transition rules, you can tailor ClearDDTS to suit your own unique needs and methodology.

Before you begin customizing ClearDDTS, however, you should consider the existing system and how simple or difficult it will be to accommodate your changes. You should also work with development engineers, project managers, and quality assurance groups to determine what bug report states should exist, and what data should be collected during each state transition. Be sure to involve your entire user community and show them what ClearDDTS provides and then ask them for input.

Locating files to customize

When you customize ClearDDTS, your changes are reflected in the web-based interface *webddts*, the character-based program *bugs*, and the graphical user interface *xddts*.

Note: For information on how web pages are generated and specific web customizations you can make see "How webddts pages are generated" in Chapter 7, Understanding the Master Template File.

To get started, look at the *~ddts/class/software* directory. When you use the *ls* command on this directory, you should see something like this:

```
README                 helps/              states
add.encl               link_semantics/     submit.encl
admin.help/            master.tmpl          summary.print/
admin.tmpl/            notify.tmpl          user.encl
batchbug/              oneofs/             user.hist
bugmail_ignore_fields  proj.prompt         user.index/
clone.prompt          report_conf          verify.encl
description           resolve.encl         web_conf
editencl.tmpl         statenames          www-helps/
```

This directory contains all of the files you may want to customize for the *software* class. Similar directories are available for each class containing the files you can customize for that class. (The only other files we recommend you customize are the *awk* report scripts located in the *~ddts/bin* directory.)

The *~ddts/class/software* directory contains the following files and subdirectories:

Name	Description
add.encl	This template is used to ask the user for an enclosure name (see *user.encl* and *editencl.tmpl* below).
admin.help/*	This directory contains help files that are used by *adminbug* when creating or modifying project parameters. It contains *adminbug* help for class-specific states.
admin.tmpl/*	This directory contains template files used by *adminbug* when creating or modifying project parameters. These files control the state-specific information (such as who gets mail when a defect enters this state) that ClearDDTS requests when you create a project. This information is then saved with the rest of the project-specific parameters.
batchbug/*	This directory contains template files that *batchbug* may use to do programmatic state transitions. You may find them useful as examples of *batchbug* processing.
bugmail_ignore_ fields	The file contains a list of fields to be suppressed when sending e-mail notification. For more information on e-mail notification, see Chapter 11, Handling ClearDDTS Mail.
clone.prompt	This template file is used for cloning defects. View the file for information on defining what fields to prompt for when clonging a defect.
description	One line description of this class.
editencl.tmpl	This template file may be used to restrict a user from editing an enclosure.
helps/*	Context sensitive help files used by the *master.tmpl* file for this class. This type of field level help is available for the *xddts* and *bugs* interfaces only. The *webddts* interface has separate HTML help.
link_semantics/*	This directory contains example implementations of a link semantic. The code in these directories allows you to traverse defect links and perform database actions on the linked records.

Name	Description
master.tmpl	Master template file controls state transitions, interaction, modification, and formatting. This file defines the rules for how bugs move from state to state, how bugs are printed to the screen, and how they are formatted for e-mail. Most of your customizations are done here.
notify.tmpl	The notification template is used to control the format and content of notification mail through a variety of configuration parameters. For more information on e-mail notification, see Chapter 11, Handling ClearDDTS Mail.
oneofs/*	The set of *oneof* files for this class. These files define the list of valid responses for particular fields.
proj.prompt	This template file defines the first screen displayed and is used to prompt for the project and class a user wants to submit a defect against. (*xddts* and *bugs* only)
report_conf	This file is used for *xddts* and *webddts* management reports. It defines the reports and menu that is displayed.
resolve.encl	This file can be used to require users to enter the editor when a defect is moved to the resolved state. It contains one line which says "Please describe your resolution to this problem:".
statenames	This file defines state letters, state names, and the attributes of each state. It also defines the "normal" progression of defect states for the class.
states	This file defines the legal state transitions and how a defect report may move from one state to the next.
submit.encl	Example of a file that could be added to an enclosure via *editfile* command. See the *Related-file* field in the *master.tmpl* file. The file can be used if you want to force the user into the editor when submitting a defect.
summary.print/*	Template files, one file per state, used for printing three-line summary reports in *ddts*.
user.encl	Template file used for printing enclosures.
user.hist	Template file used by *bugs* for printing history enclosures.
user.index/*	Template files, one per state, that define the format of the index lines.

Name	Description
`web_conf`	This file configures various aspects of the webddts interface including: - prompting for a project before submitting a defect - running reports in serial or parallel mode - available gif graph display sizes - x-axis and y-axis labels and tick marks - acceptable file types for attachments - ability to turn auto wrapping on or off for enclosures - width of the enclosure edit window - viewing expanded enclosure contents - number of enclosure icons displayed across a page - ability to make toolbar icons active or inactive
`www-helps`	This directory is used to store field-level help files for the *webddts* interface. When defining a field in the *master.tpml*, you can use the same help file for both the xddts and webddts interfaces, or create specific webddts help files. See the default software class *master.tmpl* for examples.

As you may have noticed, most of the files are ClearDDTS template files. The format and meaning of these files are described in later sections.

Adding new fields

Adding a new field to a particular state transition is as easy as adding a new field derivation to the *master.tmpl* file. When you add a new field derivation, only new bugs will have the new data incorporated into the flat defect file located in *~dmts/allbugs/**.

Note: If you are using the web interface and have defined groups of fields, be sure to add any new fields to the appropriate group. See *Specific webddts customizations* on page 8-21 for information on grouping fields.

While adding a new field derivation inserts the new data into the flat file in the allbugs directory, making it available for display, it does not insert it into the ClearDDTS SQL database. If you want to use the new field in defect metrics, sorting, database searching, querying, or index lines, you need to modify the database to include it.

Modifying the database involves editing the database schema and configuration files and running *adminbug dbms*. See Chapter 13, Managing and Customizing the ClearDDTS Database, for more information.

Note: You can also delete fields from the *master.tmpl* file or the database. However, care should be taken to avoid deleting special fields that ClearDDTS uses internally. See Appendix A for a complete listing of required fields.

Adding defect states

ClearDDTS comes with several default states and legal state transitions (see Chapter 2 in the *ClearDDTS User's Guide* for a review of these states and state transitions). This set of states should handle most users' needs. Before adding a new state, you may want to consider simply adding an attribute to an existing state. However, if you feel that a new state is necessary, you can add the new state to ClearDDTS by making some simple modifications to a few text files.

These modifications are summarized below:

- Add the new state to the *~ddts/class/<classname>/statenames* file. This file defines the names and attributes of states. See "Editing the state names file (statenames)" on page 8-7.

- Add new state transitions to the *~ddts/class/<classname>/states* file. See "Editing the state transitions file (states)" on page 8-9.

- Modify the *~ddts/class/<classname>/master.tmpl* file for prompting, modifying, displaying, and mailing notifications about defects in the new state. Because this file defines the rules for moving from state to state, the modifications you make here are more complex than in the other files.See "Editing the master template file (master.tmpl)" on page 8-11.

- Modify the template files used for prompting by *adminbug*. These templates are in the directory *~ddts/class/<classname>/admin.tmpl*. See "Editing administrative template files" on page 8-13.

- Add an index template file for the new state to the directory *~ddts/class/<classname>/user.index*. (*xddts* only) See "Modifying the information in a query index" on page 8-15.

- Add a file to the *~ddts/class/<classname>/summary.print* directory that defines how to print a three line summary of a defect in the new state. (*xddts* only) See "Editing the three-line summary template file" on page 8-17.

- Modify the *awk* scripts in *~ddts/bin* (for example, *dawk01, dawk04, dawk08* and *dawk06.sh*), so that Management Reports can report on defects in the new state. See "Changing the reporting system for new states" on page 8-17.

- Modify/create files in the ~ddts/www/cache directory so that cached *webddts* pages are reset and display new customizations. See "Maintaining the cache directory" on page 8-25.

Most of these modifications are very simple and can be completed in just a few hours. The steps are described in greater detail in the following sections.

Editing the state names file (statenames)

The *~ddts/class/<classname>/statenames* file is used to put labels on states. As shipped, this file looks like:

```
S    s  Submit     Submitted
N    u  New        New
A    a  Assign     Assigned
O    a  Open       Opened
R    r  Resolve    Resolved
V    r  Verify     Verified
D    x  Duplicate  Duplicate
P    o  Postpone   Postponed
F    s  Forward    Forwarded
```

This file defines the states that exist for this class, and each line consists of four fields as follows:

- The first field is a single uppercase character field that defines the state letter that is kept in the defect. This class has states S, N, A, O, R, V, D, P, and F. You can define up to twenty-three states per class. Each state must have a unique letter. The states S, N, and F are reserved for Submitted, New, and Forwarded.

Our example shows the state letter matching the first letter of the ASCII field name, but you can use any available letter.

- The second field is an attribute character defined as:

s	Pseudo-state. The defect generally goes through this state but doesn't stay in this state (for example, a temporary state).
u	This is an unresolved state and no engineer is assigned for repair.
a	This is an unresolved state but an engineer is assigned to fix the defect.
r	This is a resolved defect.
x	This is a dead end state (not part of the main set of states) and should not be counted in metrics (for example, a duplicate).
o	This is not part of the main set of states, but should be included in metrics.

- The third field is an ASCII name for the state in present tense.
- The fourth field is an ASCII name for the state in the past tense.

The ASCII names given in the third and fourth fields are used by *webddts*, *xddts, graphbug,* and *tallybug* to allow state selections based on more descriptive names (rather than on single letters).

Placing a New State in the File

ClearDDTS has the notion of a *mainline* set of states. For example, in the following figure, the mainline states are S, N, A, O, R, and V:

Mainline States
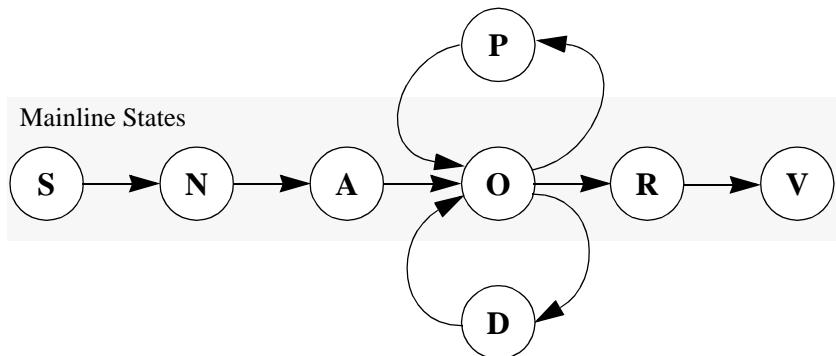
Some ClearDDTS graphs and metrics present defects as they move from state to state. For example, one graph shows a plot of

New defects, on top of that are the Assigned defects, on top of that are the Open defects, and so forth. The order used in the graph is determined by the order the states are defined in the *statenames* file. The *xddts* interface uses this order to define the natural progression (lifecycle) of a defect. Therefore the order of states in the *statenames* file is significant.

If you are adding a new state, you need to determine if it is a mainline state and where the state belongs in the overall lifecycle. Then add the new state line at the appropriate place in the file.

Editing the state transitions file (states)

Legal ClearDDTS state transitions are determined by the contents of the *~ddts/class/<classname>/states* file. This file consists of ordered triples of capital letters, one triple per line. Each letter represents a different state. Each triple *A B C* represents one possible transition: "you can get from state *A* to state *B* by first going to state *C*." Taken together, the list of triples represents the state transition diagram for the given class of defects. The order of lines in the *states* file is significant, because it provides the next logical default value for the **States** field in the *webddts* interface. There can also be comments on any line (portions of lines beginning with #). Such comments are ignored.

Here are some of the lines from the *class/software/states* file as shipped:

```
N A A
N O A
N R A
N V A
A A A
A O O
A R O
A V O
O A A
O O O
O R R
O V R
R R R
```

The first line above means that if a bug is in state *N* (New), it is possible to get to state *A* (Assigned) simply by entering state *A*.

The second line means that to get from state *N* (New) to state *O* (Opened), the bug must first enter state *A* (Assigned). The fourth line means that you must first Assign bugs that are New as the first step toward the Verified state. Note that you must go through state *R* (Resolve) to enter state *V*.

ClearDDTS uses the *states* file to determine what to do whenever you attempt a state transition. For instance, if a defect is in the *N* (New) state, and you change to the *R* (Resolve) state, the defect first moves into state *A* because of the third line shown in the example above (*N R A*). ClearDDTS asks for all of the information associated with state *A*, and then considers how to get to the desired state (*R*) from the state it is in now *(A)*.

The seventh line in the example above says that to get from state *A* to state *R*, the defect must first go through the *O* state. ClearDDTS then asks for the information associated with the *O* state and again considers how to get to the desired state *(R)* from the current state *(O)*. In this case, the line *O R R* tells ClearDDTS that the defect can be moved to the *R* state and no more processing is necessary.

If there is no triple that lists the current state in the first position, you can't get out of that state. Similarly, if there is no triple that lists a particular state in the second position, bugs can't get to that state from the current state.

The triples can be in any order. However, if there are contradictory state transition rules in the file, such as:

```
N P O
N P P
```

Then only the rule that appears last is used.

As an example, let's add a state called *K* (perhaps for Killed) to be inserted between Resolved and Verified. Let's assume that entering this state means that a program source has been modified, and the new source code has been checked into whatever

place the "official sources" are kept. We'd add the following lines to
the *states* file:

```
N K A
A K O
O K R
R K K
V K K
P K O
D K O
K O O
K R R
K V V
K P O
K K K
.
.
.
N F F
A F F
O F F
R F F
V F F
```

These lines define the following state transitions:

- Only defects in state *R* (Resolved) and *V* (Verified) can enter
  state *K* directly.

- You can get directly back to state *R* from state *K*, or back to state
  O from state *K*.

- To get from state *O* to state *K*, though, you must first enter state
  R.

- The *F* state is a pseudo state that is used to forward bugs from
  one project to another. You can't forward (*F*) defects in state *K* to
  another project (none of the lines with K in the first column has
  F in the second column).

Once ClearDDTS knows that a new state exists, you need to tell it
the transition rules (interactive dialog) for entering and leaving
that state. The next section explains how to do that.

Editing the master template file (master.tmpl)

The *~ddts/class/<classname>/master.tmpl* file defines the rules for
moving from state to state by describing the interactive dialog and
data requested to make a state transition. This *master.tmpl* file

controls all of the prompting, screen formatting, and terminal processing in ClearDDTS. If you are adding a new state or changing the terminal dialog in any way, you need to modify this file. For a description and example of the master.tmpl file, field derivations, and the OPERATION and STATE fields, see Chapter 7, Understanding the Master Template File.

Note: The format and interpretation of template files is described in detail in the *template(5)* man page. Although the template files are discussed in this chapter, you should read and understand the man page after you read this section. Pay particular attention to the syntax of the "built-in" commands.

Modifying the master.tmpl for New States

If you have read and understood Chapter 7, Understanding the Master Template File, you are ready to customize this file. If you decide to add a new state to the *master.tmpl* file, it is important to ensure that the *if match* statements are correct for your new state. The easiest way to do this is to find another state that treats each field similarly, and duplicate all of the references to that state in *if* conditions with the new state. For example, if you're adding state *K*, and it's similar to state *P*, you should change lines throughout the *master.tmpl* file that look like this:

```
if match $STATE$OPERATION Pm Pv Op Of
```

to read:

```
if match $STATE$OPERATION Pm Pv Km Kv Op Of
```

You must also add code at an appropriate place to change the value of the Status field to the new state. The syntax is:

```
set Status K
```

If you do not add the code to change *Status* into the new state, you have created a *pseudo-state*. Pseudo-states prompt for information to be added or changed in the current defect record without actually changing the state of the defect.

Another thing to consider when adding states to the *master.tmpl* file is how to handle the *OPERATION* codes of *p* and *f*. These two codes cause a *goto* to be executed in the template file. The code fragment is:

```
        .
        .
        .
# The if statement below does a "go-to X-fields" where X is
# the future state of this defect. It does this ONLY for painting
# the form ($OPERATION == f) of new questions to be filled out
# and for prompting ($OPEATION == p) for the new information
# required to enter this state.

# Other $OPERATION codes (m, v, n, and l) execute each and every
# field derivation below.

    if match $OPERATION p f# prompt or form
    and not equal "X$STATE" X
        goto $STATE"-fields"
    fi
    .
    .
    .
```

If your new state is *K*, you need to create a field called *K-fields* and insert the appropriate template code at this point in the template file.

Editing administrative template files

If you add a new state, there are four simple *adminbug* template files that must be edited so that the dialog in *adminbug* will ask appropriate questions regarding your new state. The files are in *~ddts/class/<classname>/admin.tmpl* and are:

```
aprj2.tmpl
aprj3.tmpl
mprj2.tmpl
mprj3.tmpl
```

The *aprj2.tmpl* and *mprj2.tmpl* files contain fields called *O-notify*, *R-notify*, and so forth. These fields list the users to whom mail is sent when a defect enters that state. For example:

```
O-notify:
        "\nEnter mail address of those to notify when a bug is\n"
        "opened by the assigned engineer:\n"
        "%s\n"
        help aprj06.hlp
```

```
R-notify:
        "\nEnter mail address of those to notify when a bug has\n"
        "been resolved:\n"
        "%s\n"
        help aprj07.hlp
```

If your new state is *K*, add a field called *K-notify* with the same
derivation lines as you see for the other **- notify* fields. You may
also want to add a context-sensitive help file (similar to *aprj07.hlp*)
in the *class/<classname>/admin.help* directory.

In the *aprj3.tmpl* and *mprj3.tmpl* files, you see derivations for fields
called *O-allow, O-allow-group, R-allow, R-allow-group*. For example:

```
O-allow:
      "\nList LOGIN names of users allowed to OPEN (O state) a\n"
      "bug. Just hit return if anyone is allowed to OPEN bugs.\n"
      "%s"
      help aprj12.hlp
      if not null
          goodusers
      fi
O-allow-group:
      "\nList GROUP names of groups allowed to OPEN (O state) a\n"
      "bug. Just hit return if any group is allowed to OPEN bugs.\n"
      "%s"
      help aprj13.hlp
      if not null
          goodgroups
      fi
```

These fields list the login names of the users (*O-allow*) and the
group names of the groups (*O-allow-group*) that are allowed to
make the specified state transition. For example, if your new state
is *K*, add two new fields called *K-allow* and *K-allow-group* with the
same derivation lines as you see for the other **-allow* and
*-allow-group fields. You can also add a context sensitive help file
(similar to *aprj13.hlp*) in the *class/<classname>/admin.help* directory.

As you can see, what you are doing here is adding field derivations
(similar to existing ones) regarding the mail notifications and
permissions for your new state. The derivations for state *P* fields
are good examples to copy and modify.

After you make these modifications, any new projects created with
the *adminbug aprj* command will ask the appropriate questions
concerning who can make state transitions to the new state and
who should be sent mail when a defect report enters that new

state. However, for projects previously created, anyone will be able to modify them and no mail notifications will be sent, since the new fields were not recorded for those projects. After you have modified the administration template files, you can use *adminbug mprj* to modify this project information and set it up, if desired.

Note: You should test each of your modified template files with the *tmpltest* utility to ensure that a new field does what you want before actually using it. For *adminbug* templates, use the *-m* option, because those templates do not operate in screen mode. For more information, see *Specific xddts customizations* on page 8-26.

Modifying the information in a query index

When you submit a request to ClearDDTS to query the database, the results of that query are displayed in a query index. You can determine what information is displayed in the index.

Index display in webddts

The *webddts* interface displays results in the Query Results index. Selecting fields to display in the index is user controlled (no administator work required). To select which defect fields are displayed, use the *webddts* Query Builder and click the *Show* field.

The index contains the selected fields in the order you see them in the Query Builder.

Index display in xddts and bugs

The template files located in *~ddts/class/<classname>/user.index* are used to specify the format of the index lines in *bugs* and *xddts*. There is one template file for each state defined in the system. If you add a new state to the system you must add a new file to this directory.

A template file for the resolved state is shown below:

```
Status:           "%-1.1s "
Severity:         "Sv%1d "
Identifier:       "%-10.10s "
Software:         "%-8.8s "
Headline:         "%-35.35s "
Resolver-id:      " %-8.8s"
Enclosure-count:  if equals $Enclosure-count 0
                  then
                  else
                       " +%-2d"
                  fi
```

Note that this template file just creates one line of 72 characters. (There are no newlines (*\n*) in the format strings.) This line displays various fields of the defect (*Status*, *Severity*, *Headline*, etc.) and is included in the ClearDDTS index to provide a summary of the defects under examination. As shipped, the line looks like this:

```
17 N Sv3  QTKqa00633  foo    foo is broken                +2
```

If you create a new *K* state, you must add a new file called *K* to this *user.index* directory. The contents of the new file should be similar (or identical) to the above.

Note that any field mentioned in this template file must also be defined in the ClearDDTS database. If you modify the ClearDDTS database, you must rebuild it with the *adminbug dbms* command. See Chapter 13, Managing and Customizing the ClearDDTS Database.

Editing the three-line summary template file

ClearDDTS prints defects in a variety of formats. One format is a three-line summary. The *~ddts/class/<classname>/summary.print* directory has a set of template files, one per state, that define the three line summary format. The template files look like:

```
Status:          "10t=%s, "
Severity:        "Sv=%s, "
Identifier:      "Bug id = %s, "
Version:         "Vers = %s, "
Software:        "Defect in %s\n"
Headline:        "Desc: %s\n"
Engineer:        "Assig Engr = %s ,"
Submitted-on:    "Found: %s, "
```

These files can be modified to suit your needs. If you add a new state to the system, make sure that you add a file to this directory with the new state letter as the name and containing the code needed to print defects in that state.

Changing the reporting system for new states

If you add a new state, you need to edit four files to report on defects that enter the new state. In *~ddts/bin*, edit the *awk* scripts *dawk01*, *dawk04*, *dawk08*, and the shell script *dawk06.sh*.

As an example, all of these scripts include code for adding a new *E* state, and that code has been commented out. In most cases, you only need to uncomment the code and change *E* to the new state letter that you have defined. See the scripts and Chapter 9, Creating Custom ClearDDTS Reports, for more information.

Further template customization

Besides adding new fields and states, ClearDDTS templates allow you to do many other types of customization. This section provides examples of some of the more common modifications.

Creating field dependencies

You can make the list of valid responses for one field dependent on the value of another field. For example, maybe you would like to

link together the *Project* and the *Software* fields so that if a user selects the *compiler* Project, only *C*, *fortran*, and *pascal* are acceptable answers for the *Software* field. For example:

```
Begin:      unset Begin
            set Filter-path /usr/bin:/bin:/usr/ucb
            set Oneof-path class/$Class/oneofs
            set Help-path class/$Class/helps
            .
            .
            .
Software:   if match $STATE$OPERATION Sp Sf
                    "\(1,20)Software: %-20.20s"
            fi
            if match $OPERATION p m
                    if equals $Project compiler
                            oneof C fortran pascal
                    elif equals $Project admin
                            oneof -f admin
                            help admin.H
                    else
                            oneof -f others
                            help others.H
                    fi
                    required
            fi
```

If the value of the *Project* field is *compiler*, the template will only accept *Software* field values of *C*, *fortran*, or *pascal*. In addition, since no help file is specified, the default help built into the *oneof* filter command (that is, listing the acceptable answers) is used. If the *Project* entered was *admin*, the list of acceptable answers is found in the file *~ddts/class/<classname>/oneofs/admin*. The path to find files for the *oneof* filter command is described by the special field value *Oneof-path*, which was set in the *Begin* field derivation. So, the path *~ddts/class/<classname>/oneofs/admin* is derived from the lines:

```
set Oneof-path class/$Class/oneofs
oneof -f admin
```

In the case of the *admin* projects, the context-sensitive help file for the *Software* field is located in *~ddts/class/<classname>/helps/admin.H*. The location of this file was determined similarly by the lines:

```
set Help-path class/$Class/helps
help admin.H
```

If the value of Project is not *compiler* or *admin*, the acceptable values for the *Software* field are defined in

~ddts/class/<classname>/oneofs/others and the context-sensitive help is located in *~ddts/class/<classname>/helps/others.H*.

This mechanism is simple to use and quite flexible. There are also comments in the template file to help you understand how it works and how it is used by various ClearDDTS programs.

For more complicated validation or expansion of input values, you can use the UNIX utilities such as *grep*, *cut*, and *sed*. See *template(5)* for more information on using filter commands.

Prompting for and requiring enclosures

You can prompt users to add and edit an enclosure when submitting a new record. To do this add the following derivation lines in the *master.tmpl* file:

```
Related-file: if equal $STATE$OPERATION Sp
              if uniquetitle -f "Related-file" "Problem"
                  editfile -r -i $~/class/$Class/submit.encl "Problem"
                  inc Enclosure-count
                  log Enclosure "$$" added by $Submitter-id
              else
                  editfile -r "Problem"
                  log Enclosure "$$" modified by $Submitter-id
              fi
           fi
```

This derivation already exists in the default software class *master.tmpl.* The title for the enclosure is *Problem*. The *-i* option causes ClearDDTS to use the contents of *~ddts/class/<classname>/submit.encl* as the initial text in a *Problem* enclosure file every time the user submits a new record. The *-r* option requires that the enclosure contain text (making it a required field), and when used with *-i*, that text must be different from the initial text. Without the *-r* option, the user is still prompted for the enclosure, but adding text is optional (the enclosure is not required). The rest of the derivation increments the *Enclosure-count* field which contains the number of enclosures in the defect, and includes the name of the submitter in the History enclosure.

In additon, you can use the *-R* option to make an enclosure required. This option is similar to -r (the enclosure cannot be empty), but has the additional requirement that the contents of the enclosure before editing cannot match the contents after editing. For example, this could be used to force additions to the Resolution enclosure if a defect is re-opened and then resolved a second time.

You can use a similar mechanism to prompt users to edit existing enclosures or add enclosures at different state changes. See the comments in the *master.tmpl* file for more examples.

Customizing enclosures, prompts, and e-mail

In addition to the modifications already discussed, ClearDDTS provides some other template files that you may want to customize. The following table summarizes these files and the kinds of modifications you can make. For more information, see the files themselves.

Template File	Modifications
clone.prompt	The clone prompt template file *clone.prompt* is used to control the dialog used when cloning a defect.
notify.tmpl	The notification template is used to control the format and content of notification mail through a variety of configuration parameters. For more information on e-mail notification, see Chapter 11, Handling ClearDDTS Mail.
resolve.encl	The file can be used to prompt for an enclosure upon resolving a defect. This is an *xddts* only feature.
submit.encl	The file can be used to prompt for an enclosure upon submitting a defect. This is an *xddts* only feature.
editencl.tmpl	The *editencl.tmpl* template file is executed before editing an enclosure. It is provided so that you can restrict who may edit enclosures. This is an *xddts* only feature.

Creating custom filter commands

You can create and use your own filter commands. These commands could be shell scripts or C programs that read a value from standard input, write the value to standard output, write any explanations of why the value is bad to standard error, and exit with a status of zero (good) or non-zero (bad).

These commands can be used for field validation or for initializing system-generated default values. For example, a filter command is used to initialize the Assigned on field with the current date.

By ensuring that the filter command is at a location in the *Filter-path* (given in the *Begin* field in the *master.tmpl* file), you can use your new filter command in your customized template. An example of using a shell script for a user-defined filter command appears in Appendix C, Sample Filter Command Script.

Specific webddts customizations

If you plan on using the *webddts* interface there are several changes to the user-customizable portions of ClearDDTS, specifically to the *master.tmpl* and *web_conf* files, that will enhance the web page display.

Label and type modification via the "www" filter

Some fields do not usually have an appropriate field label in view mode (for example, the Headline field) making the View Defect page hard to understand.

To solve this issue we use a filter called "www". Used with the "if" command, you can effectively control the output of the HTML pages. We have used this in several fields in our default *master.tmpl* file, specifically in the Headline, Submitted-on, Software, and Resolved-on fields. You can refer to the *master.tmpl* file in *~ddts/class/software* for exact syntax.

A more complete description of the interaction of the HTML generation tools (*wtform* and *wttmpl*) and your *master.tmpl* files is

available in a white paper available from Rational Technical Support.

Web layout using field grouping

In the *bugs* and *xddts* interfaces the layout of the form is controlled by cursor addressing in the prompt strings. In HTML the layout is in the order of the *master.tmpl* file. This can be modified by grouping fields into related sections.

The *webddts* defects are displayed in a grouped table format, similar to the way the submission and modification forms are displayed, except that the values are displayed as static text and not input items. This grouping is configured through the use of the command "set" with the first argument "group_fields".

Group fields are comma separated field specifiers. Each position can have an optional modifier after the field name delimited by a colon (:), as well as multiple modifiers separated by white space (spaces or tabs). Each position must have at least a modifier or a field name. The format looks like:

```
set group_field [<field>][:<modifiers>][, [<field>][:<modifiers>][,
[<field>][:<modifiers>] [...] ] ]
```

Modifiers can be broken down into two categories:

▪ table modifiers.

Table modifier	Result
numcols=##	Specifies the number of fields wide (label-value pairs) the table appears.
tablewidth=###	Specifies the minimum width of the table in relation to your browser. This number can be specified as a percentage by putting quotation marks around the percent string (tablewidth="75%").
	To make a table always be as wide as the browser window enter tablewidth="100%".

- field modifiers

Field Modifier	Result
nowrap	Prevents text within both the label cell and the value cell from wrapping.
nowrap_label	Prevents text within the label from wrapping.
nowrap_value	Prevents text within the value cell from wrapping.
colspan=##	The number you specify is the number of other logical "fields" you want the value cell to occupy.
null	Causes an empty label cell and value cell to be inserted in the corresponding position. The actual field named, if any, is not displayed. Note: Do not specify a field name when using the null modifier. An incorrect field name will cause the modifier to be ignored.
startcol=##	Makes the specified field begin in the specified column. For example, you can make enclosures always start in the first column by specifying startcol=1.
mailto	Wraps the contents of the cell within a mailto html tag: <A href="mailto: [field contents]"> </A> Used with address fields, this allows a convenient way to contact the people associated with the current defect.

Table modifiers always need to appear before the first field in the group_fields list. You cannot specify a field with table modifiers.

The following are correct and incorrect examples of group fields:

Correct:

```
set group_fields    :numcols=4, Field1, Field2
```

Incorrect: Field1 would not be displayed:

```
set group_fields    Field1:tablewidth="60%" numcols=2, Field2
```

Correct:

```
set group_fields   :tablewidth="60%" numcols=2, Field1, Field2
```

Incorrect: All table modifiers must be in the first position:

```
set group_fields   :tablewidth="60%", :numcols=2, Field1, Field2
```

Incorrect: Spaces separate modifiers, so there is no valid modifier
on this line:

```
set group_fields  Field1: colspan = 5, field2
```

Example

The following example is from the default *master.tmpl* file shipped
with the software class. It shows how the table is created to
display the Defect Information group when viewing a defect:

```
Def-group:
                if www
                        "DEFECT INFORMATION"
                        set group_fields :numcols=2 tablewidth="75%",
Project, Identifier, Software, Version, Headline : colspan=2 startcol=1,
Showstopper, Enhancement, Status, STATE, Last-mod, Enclosure-count,
Problem_encl : colspan=2 startcol=1
                fi
                unset Def-group
```

The table appears on the view defect page as:

Web display options via the web_conf file

Edit the *~ddts/class/<class>/web_conf* file to control various
aspects of the *webddts* interface display including:

project-pre-prompt	If you have fields dependent on the Project field, prompting for Project before presenting the rest of the Submit page reduces the JavaScript required (helping to avoid JavaScript errors). To prompt for the project field before displaying the Submit page, set the value of project-pre-prompt to "Y".

report_run_mode	Determines if reports are run in "serial" or "parallel" mode. Serial is the default. Depending on your machine, parallel may not provide the best result since it consumes machine resources.
report_gifsize	Determines the sizes of gifs produced for graphs, and the default value. It also determines the labels presented in the Size field in the webddts Generate Reports page. See the examples in the web_conf file for more information.
report_scale_labels	Sets an autoscaling factor for x-axis and y-axis labels. The higher the number, the fewer the labels.
report_scale_ticks	Sets an autoscaling factor for x-axis and y-axis ticks. The higher the number, the fewer the ticks.
attachment-ignore-ext	Attachments normally require an extension so the browser knows what to do with them. You can override this behavior by specifying a regular expression to match for exclusions. For example, to attach core file, do the following: `attachment-ignore-ext: core.*`
auto-encl-wrap	Determines whether enclosure text should automatically wrap. If set to Y, enclosure wrapping is allowed, and is saved in the database. If set to N, enclosure text will not automatically wrap. Defaults to enclosure wrapping even if this file or line are missing. The wrapping occurs at the width of the enclosure window.
encl-width	Sets the width of the enclosure edit window. Note that if the width is set to anything less than 40 or greater than 132, the window defaults to a width of 72 even if this file or line are missing.
enclosure-icon-wrap	Sets the number of enclosure icons you want to appear in a single row across the View Defect page.
expand-enclosures	Determines whether to display the expanded contents of enclosures below the defect information on the View Defect page. The default is "N". Users can also set this option on the User Profile page in the web interface.
toolbar-mode	The toolbar can be "active", meaning it contains JavaScript to present moving images, or it can be "inactive", where no JavaScript runs, and the images are static. The default is active. Users can also set this option on the User Profile page in the web interface.

Maintaining the cache directory

ClearDDTS uses a caching system to store initial versions of the *webddts* pages. The cached pages are stored in the

`~ddts/www/cache` directory. The ClearDDTS cache is completely separate from the browser cache. It is only used for ClearDDTS pages so that they do not need to be generated each time you access them.

Cache files are cleared automatically when changes are made to the *master.tmpl* file. However, manual clearing may be necessary after certain changes and customizations are made. If you do not see your changes appear, there are files you can create in the cache directory that perform cleanup as necessary. These files include:

NOCACHE	Cache files are still created, but are continually reset.
CLEAN	Cache files older than the CLEAN file are reset
CLEAN_<class>	Cache files specific to the named class are reset if they are older than the CLEAN.<class> file.

Users can also clean their own cache through the *webddts* interface by clicking the **Clean Cache** button on the User Profile page.

Specific xddts customizations

This section contains features unique to the *xddts* interface.

Adding new pages

To support multiple pages in a defect report in *xddts*, ClearDDTS uses four special commands and one variable.

Note: The *webddts* pages use scroll bars for viewing information that requires more than one page to display. Multiple pages are not necessary.

Using the PAGE variable and return statement

The PAGE variable is used to distinguish between the various pages associated with a defect report. This variable is used in the *master.tmpl* file in much the same way as *STATE* and *OPERATION*. The template file runs with the value of PAGE set to some value, and is used in conditional statements to display different pages.

The *xddts*, *bugs*, *bugmail*, and *dumpbug* utilities use the PAGE variable to display multiple pages. This requires a "handshake" between these programs and the execution of the template file so that these applications know which page should be displayed next. This handshake is accomplished through the use of the template file *return* statement.

To force the display of a different page, the last statement executed in the *master.tmpl* file must be a *return* and it must return the name of the next page to be displayed. This name can be any string up to thirty characters. For backwards compatibility, the null string is the first (base) page, and returning null means there are no more pages left to view. This return value is stored internally as RETURNVAL, so you **must not** use this variable as a field variable name.

Modifying the master template to display other pages

When executing the other pages, you should manually clear the screen, paint the form, prompt for or modify the additional fields, then go back to the first page. To do this, you use:

- the *call* filter which recursively executes the *master.tmpl* file,
- *pushscreen* and *popscreen* which save and restore the screen respectively, and
- \f which clears the screen.

The *call* filter is used with variable/value pairs that re-execute the current template with the new variables. This allows you to repaint the form for the next page, prompt for the new fields on the next page, and then return to the previous page. The variables are only set locally and return to their original value when the *call*

filter is finished. Note that all fields are global and can be changed from any page.

You can use multiple pages in a variety of ways, but most likely they will fall into three categories:

- Sequenced: Pages displayed/modified one after another.
- Sectioned: Every displayed/modified page has data in common.
- Forms-within-forms: Various pages displayed/prompted on demand.

As an example, assume that you want to add an extra 10 fields to the Verified state in the *software* class. Since the *software* class screen is already quite full, you are going to prompt for and display all of the Verified fields on a second page. To do this, you need to modify the *software* class *master.tmpl* file as follows:

1 To make sure that the V-fields are displayed on the second page, you must place a statement at the beginning of the *master.tmpl* (**before** the code that does the *goto $STATE"-fields"*).

```
# Add these three lines

     if equals "$PAGE" VERIFY
          goto V-fields
     fi

# BEFORE these four lines

     if match $OPERATION p f      # prompt or form
          and not equal "X$STATE" X
          goto $STATE"-fields"
     fi
```

In this example, the second page is named VERIFY. You can name a page anything you wish; however, you should make it descriptive. Note the double quotes around $PAGE. This is required because the first page is named with the null string. The *equals* statement will not work correctly without these quotes.

2 Modify the code located at the *V-fields* field derivation. This derivation is a preamble to the actual execution of the various Verify state fields. Make sure that the code will only be executed

when the value of PAGE is VERIFY. There are two situations, output mode and input mode, that you must address:

- If the template file is executing in output mode (OPERATION *f* or *v*), you must check that the page is correct.
- If the template file is executing in input mode (OPERATION *p* or *m*), you must explicitly cause the second page to be executed.

The code to implement is shown below:

```
V-fields: unset V-fields
        if not equals $STATE V
             goto F-fields
        fi
        if not equals "$PAGE" VERIFY
        and input
             pushscreen
             "\f"
             call -o OPERATION f PAGE VERIFY
             call -i PAGE VERIFY
             popscreen -r
        fi
        if not equals "$PAGE" VERIFY
             goto Last-mod
        elif equals $OPERATION v
             "\f"
        fi
```

The *pushscreen* command saves the current screen in memory. The *\f* then clears the entire screen. The first *call* command re-executes the *master.tmpl* file with PAGE set to VERIFY and OPERATION set to *f*. This paints the verify form on the screen. The second *call* command again causes the entire *master.tmpl* file to be re-executed with PAGE set to VERIFY and OPERATION set to *p*.

At this point the user is prompted for all the verify information. When the last field value is entered, execution returns to the *popscreen* command. The *popscreen* command restores the screen saved by *pushscreen*.

3 Now you simply need to define the page order for printing. This is for programs like *dumpbug* that need to print out all defect pages and **then** print the enclosures.

To set up printing order, insert the following code **after** the *Last-mod*
field derivation and **before** the *Do-enclosures* derivation:

```
#  This is existing code

Last-mod: if match $OPERATION p m
                today
          fi

# Here is the new code

Pages:    unset Pages
          if null "$PAGE"
                  return VERIFY
          fi
```

If we are executing the first page, then we return the next page,
VERIFY. If we are executing the last page, in this case also
VERIFY, then we fall through to the enclosure code so that
dumpbug works correctly.

Implementing other customizations

The example above illustrates only one possibility. Other
customizations can be done with the same code and additional
conditional statements. Placing these three blocks of code in
various places within the *master.tmpl* file will also produce varying
effects. For example, placing the first block farther down in the
master.tmpl file will produce a sectioned form where some common
data would be displayed on every screen (see *company* class for an
example). This will make all pages print some common fields.

You could also make the second page dependent on *Verify-check.* For
example:

```
if equals $Verify-check Y,
   pushscreen
   call -o OPERATION f PAGE VERIFY
   call -i PAGE VERIFY
   popscreen -r
fi
```

This would give you forms-within-forms. An almost unlimited
number of combinations can be implemented.

Debugging a custom template file

Once you make changes to the template file, you need to debug these changes without affecting all of the projects on the system. There are two ways to do this:

- Using a special (dummy) class
- Using the *tmpltest* utility

Setting up a dummy class

One way to debug a custom template file is by creating a dummy class with the *adminbug clas* command. When you create the new class, ClearDDTS makes a copy of the *master.tmpl* file in the new *class/<classname>* directory. You then create a ClearDDTS project that is a member of this class with the *adminbug aprj* command. After you have created the new class and project, you can modify the class *master.tmpl* file and submit test defects against the new project to test your changes.

Testing a template file

Another way to debug template files is with *tmpltest*. This program allows you to test a template with a set of specified *STATE* and *OPERATION* values without creating a dummy project or using the *bugs* program.

For example, recall the *foo.template* and *foo.script* files mentioned earlier that demonstrated the *interact* filter command. The *tmpltest* utility can be used to test run these scripts. To debug these scripts, you could issue the following commands:

```
chmod +x foo.script          # make foo.script executable
tmpltest -Xfpv foo.template
```

The options cause *tmpltest* to execute the template file *foo.template* exactly the same way *bugs* and *xddts* execute it. That is, once with *STATE* equal to *X* and *OPERATION* equal to *f* (form), again with *STATE* equal to *X* and *OPERATION* equal to *p* (prompt), and finally with *STATE* as *X* and *OPERATION* equal to *v* (view).

As another example suppose you want to test a change to a locally modified version of the *master.tmpl* file. The following invocation would step through all of the executions of the template file just like *bugs* would for a newly submitted defect that you immediately decided to resolve. It would also go through all of the steps in modifying a defect for a particular state.

```
tmpltest -ofpvm -SAORV template.file
```

This would test all executions for *xddts* and *bugs*. See the *tmpltest* man page for a complete explanation of how *tmpltest* works.

9

Creating Custom ClearDDTS Reports

This chapter explains how the ClearDDTS report system is constructed and how it can be customized. Read this chapter if you want to modify the reports supplied with ClearDDTS or create new reports.

As shipped, ClearDDTS includes all of the source programs for the management reports so that these source programs can be modified to suit your needs. If you want to report additional information or produce custom reports, working examples are available to use as a starting point.

The following topics are covered:

- Understanding how reports work
- Creating Reports
- Integrating a report into the xddts and webddts interfaces

Understanding how reports work

Each ClearDDTS interface expects reports and graphs to be produced in a particular format. For a report program to work, it must accept ClearDDTS parameters and produce output in the correct format. This section details those parameters and formats.

The report_conf file

One of the most important files in the ClearDDTS report mechanism is the *~ddts/class/<classname>/report_conf* file. This file associates a report title with a report program and specific attributes. The ClearDDTS interfaces look here to see what reports are available in the formats appropriate for that interface.

To integrate a report into the *xddts* and *webddts* interfaces it must appear in the *report_conf* file. A copy of this file is shown below:

```
#                                report_conf
#
# This file drives the ClearDDTS reporting mechanism - note that the '#'
# character indicates a comment and all text from that character
# to the end of the line will be ignored.
#
# a line ending with a '\' character will be considered to continue
# on the next line. Each Report configuration line consists of three
# fields separated by the ':' character.  Each line looks like
#
#   attributes : Report Label Text : report command string
#
#                   R E P O R T   A T T R I B U T E S
#
#   -ascii  - the report's output is in ASCII format
#   -ps     - the report's output is in PostScript format
#   -troff  - the report's output is in Troff format
#   -notty  - the report should not be sent to a tty device.
#   -gif    - the report's output is in gif when run from the web
#   -html   - the report's output is in html when run from the web
#
#                   R E P O R T   L A B E L   T E X T
#
# The report label text is displayed in the list of available reports.
#
#               R E P O R T   C O M M A N D   S T R I N G
#
# The report command string is essentially a command similar to a string
# that would be passed to the system (3) call.  There are several
# variables that are understood and interpreted by the reporting
mechanism.
# They are as follows.
#
#    $start    - start of date range for the reporting period
#    $end      - end of date  range for the reporting period
#    $projects - list of projects to be passed to report script
#    $states   - states argument to be passed to report script
#    $color    - Tells report script whether or not to generate
#                color output.
#
#
-ascii            : Table of Problems by Project by State.:\
                    dawk01.sh $start $end $states $projects # an appended
comment
-ascii            : Table of Problems by Project by Severity.:\
                    dawk02.sh $start $end $states $projects
-ascii            : Table of Problems by Assigned Engineer by Severity.:\
                    dawk03.sh $start $end $states $projects
-ascii            : Table of Problems by Assigned Engineer by State.:\
                    dawk04.sh $start $end $states $projects
-ascii            : Table of Problems by Submitter by Severity.:\
                    dawk05.sh $start $end $states $projects
-ascii            : Table of Problem Arrival and Repair Rates.:\
                    dawk06a.sh  -b $start $end $states $projects
-ascii            : Three Line Summary of all Problems.:\
                    dawk07.sh $start $end $states $projects
```

```
-ascii            : General Problem Statistics.:\
                    dawk08.sh $start $end $states $projects
-ascii            : Report of Problems Submitted Over the Last Week.:\
                    wsubmit.sh $states $projects
-ascii            : Report of Problems Resolved Over the Last Week.:\
                    wresolve.sh $states $projects
-ascii            : Full Text Listing of Queried Problems.:\
                    wdump.sh
-ps -notty -gif   : Graph of Phase Metrics.:\
                    dgawk01.sh $color -b $start $end $states $projects
-troff            : Phase Containment Effectiveness Report.:\
                    dgawk02.sh -b $start $end $states $projects
-ps -notty -gif   : Phase Containment Effectiveness Bar Graph. :\
                    dgawk11.sh $color -b 1 $start $end $states $projects
-ps -notty -gif   : Total Faults Sourced Per Phase. :\
                    dgawk11.sh $color -b 2 $start $end $states $projects
-ps -notty -gif   : Graph of Problems by Resolution Type.:\
                    dgawk03.sh $color -b $start $end $states $projects
-ps -notty -gif   : Graph of Submitters, Assigned Engineers, and
Resolvers.:\
                    dgawk05.sh $color -b $start $end $states $projects
-ps -notty -gif   : Graphs - Problems by Severity, Problems by State,
Problem Arrival Rate.:\
                    dawk06.sh $color -b $start $end $states $projects
-ps -notty -gif   : Graph of Problems by Severity.:\
                    dgawk07.sh $color -b $start $end $states $projects
-ps -notty -gif   : Graph - Repair Time & Diff Between Est & Actual Fix
Time.:\
                    dgawk08.sh $color -b $start $end $states $projects
-ps -notty -gif   : Graph of Overdue Problems. :\
                    dgawk09.sh $color -b $start $end $states $projects
-troff -notty     : One Line Meeting Summary of Selected Problems.:\
                    dgawk10.sh -b $start $end $states $projects
```

The active lines in the file are tuples separated by colons. The first parameter is an attribute that describes whether the output is ascii, postscript, gif, html, or troff output. The second parameter is used by *webddts* and *xddts* to produce the list of reports to select from. The last parameter is a string that is used to invoke the report.

There is a separate *report_conf* file for every class.

Report scripts

The default ClearDDTS reports consist of Bourne shell scripts that use *awk* to process the data. These files are located in *~ddts/bin*. This directory contains many pairs of files named dawk*nn* and dawk*nn*.sh; for example, dawk09 and dawk09.sh. There are also pairs of dgawk*nn* and dgawk*nn*.sh scripts.

Each dawk*nn*.sh or dgawk*nn*.sh file is a shell script that runs the corresponding *dawk/dgawk* report script. Most of these shell scripts are similar except for the name of the *awk* script that they invoke.

How each interface uses these scripts to produce reports is detailed below.

xddts

To run a report, *xddts* executes the following process:

1 *xddts* accesses the *report_conf* file to obtain the list of reports to display on the Management Reports screen. It supports reports that are defined with the following attributes: *-ps*, *-ascii*, and *-troff*.

2 When a user selects a report or graph, *xddts* runs the shell script associated with that title using the values of the associated variables. For example, the Graph of Problems by Severity calls the *dgawk07.sh* script as defined in the *report_conf* file:

```
dgawk07.sh $color -b $start $end $states $projects
```

3 The *dgawk07.sh* script expands the variables and runs an associated report script, *dgawk07*, to produce the output.

4 In the case of a graph, an additional step is necessary to pass the information to the *tallybug* program, which in turn uses *graphbug* to produce a PostScript graph.
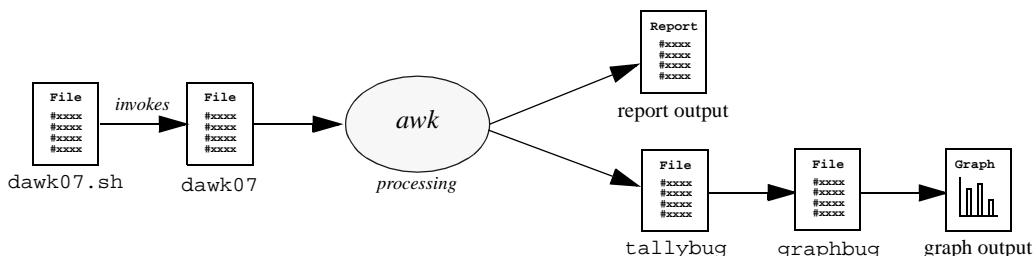
webddts

To run a report, *webddts* executes the following process:

1 *webddts* accesses the *report_conf* file to obtain the list of reports to display on the Generate Reports page. It supports reports that are defined with the following attributes: *-ascii*, *-html*, *-gif*, and *-ps*.

2 When a user selects a report or graph, the web reporting engine builds and runs a temporary shell script which:

- expands a set of web-specific environment variables including:

```
PATH # PATH set for architecture
DDTSHOME # ddts home directory
DDTSCLASS # class in which the query is run
DDTS_WEBPROG # webddts program, "/ddts/ddts_main"
DDTS_REMOTE_USER # userid
DDTS_REPORT_TYPE # -gif|-ascii|-ps...
DDTS_THCOLOR # table header color, for html reports
DDTS_BGCOLOR # page background color, for html reports
DDTS_TEXTCOLOR # text color, for html reports
DDTS_REPORT_FIELDS # the "show" fields from the query
DDTS_REPORT_STARTDATE # earliest date from the query (for tallybug)
DDTS_REPORT_ENDDATE # latest date from the query (for tallybug)
DDTS_REPORT_STATES # list of states from the query (for tallybug)
DDTS_REPORT_SEVERITY # list of severities from the query (for
  tallybug)
DDTS_GRAPH_FORMAT # "gif" or empty
DDTS_GRAPH_BASE # directory name
DDTS_GRAPH_WIDTH # pixels, used by gifbug
DDTS_GRAPH_HEIGHT # pixels, used by gifbug
DDTS_GRAPH_GRID # true or false, used by gifbug
DDTS_GRAPH_TRANSPARENT # true or false, used by gifbug
```

- calls the associated shell script (for example, dgawk07.sh)

3 The shell script does the following:

- ignores the variables ($start, $end, $states, $projects) normally associated with the script in the *report_conf* file and processes the *findbug* query string produced by the *webddts* Query
- runs *awk* (for reports) or *tallybug* (for graphs)

4 In the case of a graph, an additional step is necessary to pass the information from the *tallybug* program to either *gifbug* (to produce

the graph in .gif format) or *graphbug* (to produce the graph in .ps format).
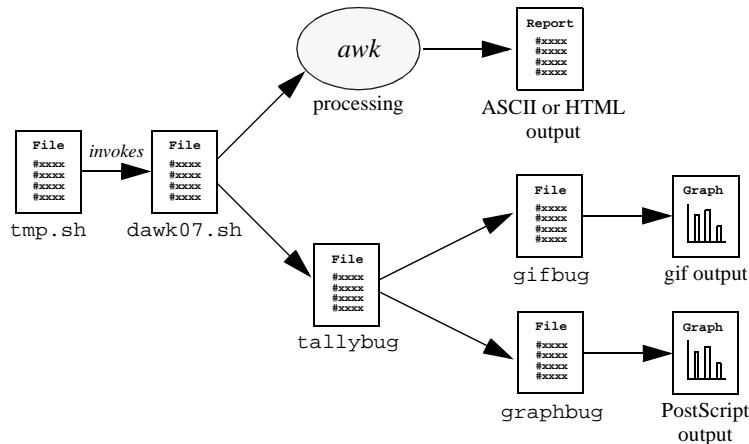
Creating Reports

You can write report scripts in any format that will accept ClearDDTS input variables and produce the expected output. As a start, you can edit the existing shell and *awk* scripts. For example, assume you want to write a ClearDDTS report to produce ASCII (.txt output) and call it report #22. To do this:

1 Go to the *~ddts/bin* directory and copy the file *dawk02.sh* to *dawk22.sh*:

```
cd ~ddts/bin
cp dawk02.sh dawk22.sh
```

2 Change the *dawk22.sh* shell script to invoke your *awk* script called *dawk22*.

3 Create/modify the *dawk22* script to produce the desired report. To do this, you need to know how to use *awk* and what defect records look like. Defect records are normal UNIX text files with lines of the form (see Appendix A for a complete description of the fields):

Keyword: value

A portion of a defect record file is shown below. Note that these files always begin with the "Start: *bug-ID*" field and end with the "End: *bug-ID*" field.

```
Start: CMMaa000135
Project: DDT.bugs
Engineer: mike
Submitted-on: 880606
Headline: The clinit(3) routine dumps core in diagnostic mode
Severity: 2
Status: N
        .
        .
        .
End: CMMaa000135
```

The defect records were designed to make writing *awk* scripts easy. For example, look at *dawk07*, a portion of which is shown below:

```
/^Severity:/ {
              severity = $2
              next
             }

/^Status:/   {
              status = $2
              next
             }

/^Headline/  {
              for(i = 2; i <= NF; ++i)
              headline = headline $i " "
              next
             }
             .
             .
             .
             {
              printf "\nBug Number = %s\n",xstart. . .
              printf "St=%s, Sv=%s, %-65.65s\nModule: %s, . . .
             }
```

This script produces three-line bug summaries. If you already know how to write *awk* scripts, writing scripts for the ClearDDTS defect record format is very easy. If you are not familiar with *awk* scripts, there are numerous sample scripts to start with.

GIF reports

For *webddts* to produce graphs in gif format the report must pay attention to the DDTS_GRAPH_FORMAT environment variable. If it is set to "gif", the report must call the program *gifbug*. The *gifbug* program operates much the same as *graphbug* (it parses tallybug output), but it produces gif instead of PostScript output.

Since multiple gifs can be generated, the gifs need to be created in a certain way so that *webddts* knows how and where to view them:

```
if [ "$DDTS_GRAPH_FORMAT" = "gif" ]; then
  gifbug < tmp/dg5tmp1.$$ >$home/www/$DDTS_GRAPH_BASE.01.gif
2>$home/www/$DDTS_GRAPH_BASE.01.err.txt
  gifbug < tmp/dg5tmp2.$$ >$home/www/$DDTS_GRAPH_BASE.02.gif
2>$home/www/$DDTS_GRAPH_BASE.02.err.txt
  gifbug < tmp/dg5tmp3.$$ >$home/www/$DDTS_GRAPH_BASE.03.gif
2>$home/www/$DDTS_GRAPH_BASE.03.err.txt
  gifbug < tmp/dg5tmp4.$$ >$home/www/$DDTS_GRAPH_BASE.04.gif
2>$home/www/$DDTS_GRAPH_BASE.04.err.txt
  echo "$Title1" > $home/www/$DDTS_GRAPH_BASE.00.desc
  echo "$Title2" >> $home/www/$DDTS_GRAPH_BASE.00.desc
  echo "$Title3" >> $home/www/$DDTS_GRAPH_BASE.00.desc
  echo "$Title4" >> $home/www/$DDTS_GRAPH_BASE.00.desc
  rm -f tmp/dg5tmp1.$$ tmp/dg5tmp2.$$ tmp/dg5tmp3.$$ tmp/dg5tmp4.$$
  exit 0
else
  graphbug $colorflg < tmp/dg5tmp1.$$ > $home/tmp/ps.dg5awk1.$$
  graphbug $colorflg < tmp/dg5tmp2.$$ > $home/tmp/ps.dg5awk2.$$
  graphbug $colorflg < tmp/dg5tmp3.$$ > $home/tmp/ps.dg5awk3.$$
  graphbug $colorflg < tmp/dg5tmp4.$$ > $home/tmp/ps.dg5awk4.$$
fi
```

This code snippet shows that *gifbug* is called in much the same way as *graphbug*, but the output filename has changed.

In the case of multiple gifs, each gif should be directed into its own sequentially numbered file. Furthermore, the description file contains one title per line. W*ebddts* associates the first line of this file with "01.gif", the second line with "02.gif" and so on. When the report is complete, *webddts* lists them something like:

```
_Page 01_ Problems by state
_Page 02_ Problems by severity
_Page 03_ Problems by engineer
_Page 04_ Problems by phase
```

This clarifies what each gif will display when you click on the link. Be sure that the description file is numbered "00" so that it sorts before the other gif files, otherwise it will not get picked up.

You can customize the gif display sizes available in the *webddts* interface and set the default value (640x480 as shipped). To do this edit the *~ddts/class/<class>/web_conf* file. You can also customize the number of x-axis and y-axis labels and ticks through the *web_conf* file.

HTML reports

For html reports, the report itself must generate the appropriate html tags so that the web browser can interpret it properly. Three color environment variables are set that can be used in HTML reports:

```
DDTS_THCOLOR   # table header color, for html reports
DDTS_BGCOLOR   # page background color, for html reports
DDTS_TEXTCOLOR # text color, for html reports
```

Including these environment variables keeps the report visually continuous with the rest of the web interface.

You can also provide links to defects in the report by building URLs such as:

```
<a href=$DDTS_WEBPROG?REMOTE_USER=$DDTS_REMOTE_USER&NextForm=DumpBug
&bug_id=$bug_id>
```

where $bug_id is a defect identifier that you have calculated. This URL displays only the defect record, not the full frames mode of *webddts*. To get the full frames mode, specify id=$bug_id instead of bug_id=$bug_id.

If there is a whole list of defects, you can display them in a new window by specifying a target in the URL.

Integrating a report into the xddts and webddts interfaces

To integrate a new report into the *xddts* and *webddts* interfaces, you need to look at the *~ddts/class/<class>/report_conf* file. This is where the interfaces look for the list of reports to display.

Note that the reports being invoked in the default report_conf file are the *dawknn.sh* shell scripts. Reports you add can also be awk scripts or any other type you have created that will produce the expected output.

To add your *dawk22.sh* report from the previous example to the *report_conf* file, you would add something like the following:

```
-ascii:   This is the foo report:   dawk22.sh $states $projects
```

10 Customizing Link Actions

This chapter describes how to define the actions that you want applied to linked records. The following topics are covered:

- What is defect linking?
- Configuring links
- Defining link actions

For information on creating links between records see the *ClearDDTS User's Guide*.

What is defect linking?

ClearDDTS allows you to link defect records. This is useful if your installation needs to collect defects into specified groups. For example, some installations have the notion of a Change Proposal (CP) and a Problem Report (PR). Defects and enhancements are logged as PRs and a collection of PRs is put together and defined as a Change Proposal (CP). This can be pictured as follows:

Change Proposals ▶ CP   CP

PR   PR   PR   PR

Problem Reports (Individual Defect and Enhancement Records)
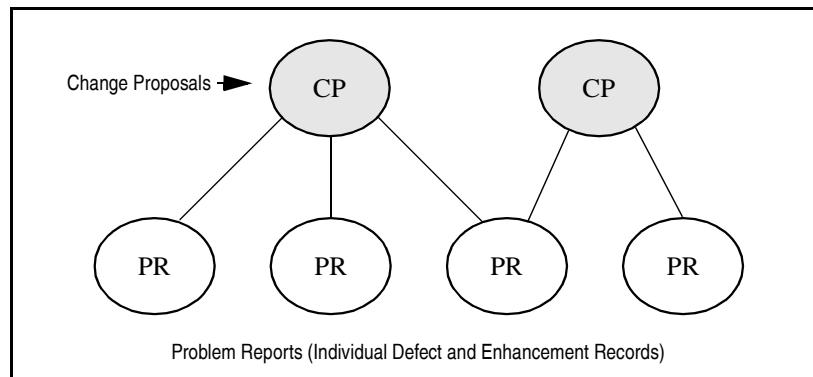
In this example, the Change Proposals (CPs) can be seen as the *parent* records and the individual Problem Reports (PRs) are the *children* linked to one or more CP.

Once you have established how records are linked, you can define rules for what actions or operations are performed based on those links. For example, one of these CPs might be discussed at a Change Control Board meeting and approved. When the CP is moved to the Approved state, you may then want all associated PR also moved to the Approved state. Therefore, defect linking involves configuring links and defining link actions.

Configuring links

Every defect has two fields for entering *parents* and *children*. Using the CP and PR model, the CP can be identified as a parent and the PR as children. The defect IDs of the children are put into the *children* field of a parent defect and the defect IDs of the parents are put into the *parents* field of the children defects. Children of the same parents are considered *siblings*.

Defect linking is also very flexible, and children can have more than one parent or more than one generation (for example, *grandchildren*). The number of parent and children links is also configurable via the number of bytes set aside in the *parents* and *children* fields in the *~ddts/dbms/ddts/schema_file* file (11 bytes per link). By default, the system supports six parent and six child links (or 66 bytes for each).

Defining link actions

When you create links, there is usually an operation or action that you want associated with the link. We call this action a *link semantic*. For example, an Approved CP might cause all the associated PRs to be put into the Approved state, or you might define a link semantic to change the status of the CP every time all PRs have been moved to the next logical state (for example, if all PRs are Resolved, the CP is automatically Resolved).

The ClearDDTS administrator can configure the semantic involved with linking for every database transaction. Every time a database transaction occurs, a trigger is called to see if a link semantic action should be executed. This trigger is a Bourne shell script called *ddtsbackend*. This script has the ability to look at (and modify) any fields in the record (CP/PR) just modified or look at (and modify) any fields in the parents or children of the just modified CP/PR. This design allows the link semantic to be customized by class or by *any other field in the CP/PR*. Thus the number and type of link semantic processing is not restricted.

Below is a sample *ddtsbackend* script that does link processing. The semantic action implemented is to look at the Class and if the Class is equal to "Change.CP" and the Status field is equal to "G" (Approved), then all the child defects are moved to the "G" (Approved) state.

```sh
#!/bin/sh

#
#       C L E A R D D T S   D B M S   U P D A T E   T R I G G E R
#
# This script is used as a backend trigGer for ClearDDTS database updates
# Whenever bugs.in(1) does a DBMS update it marks the bugid that was
# just processed in the ~ddts/spool/dbackend directory.  The ddtsd(1)
# daemon will notice this and dbackend(1) will invoke this shell script.
# This script is invoked with the concurrency control that the ddtsd(1)
# daemon provides.  With this mechanism in place only 1 ddtsbackend(1)
# script will run at a time.

# This script is invoked within the DDTs home directory as the real
# user 'ddts'.  The script needs to do backend processing for all files
# located in the ~ddts/spool/dbackend directory.  This script MUST remove
# the file when it is finished processing it otherwise dbackend(1) will
# assume a core dump and try twice more to execute this script before
# manually removing the file and complaining (via E-mail) to the DDTs
# administrator.

# This does not need to be a script, it could be a C program.


# ##### DEBUG ##### DEBUG ##### DEBUG ##### DEBUG ##### DEBUG ##### DEBUG
# set -x
# cd `ddtshome`
# ##### DEBUG ##### DEBUG ##### DEBUG ##### DEBUG ##### DEBUG ##### DEBUG
xx=`today -l`
echo "ddtsbackend $xx Daemon Started - $$" >> $home/spool/LOG

x=`ls spool/dbackend`
for bugid in $x
do
```

```
      if test X"$bugid" = "X"
      then
            continue       # empty record?
      fi
      bdir=`echo $bugid | cut -c9,10`    # find RELATIVE path to bug
      bugfile="allbugs/$bdir/$bugid"     # bugfile has path

#  Do Link processing here

      set -- `bugval -i $bugid Class Status Identifier Children`
      if test "$1" = "Change.CP"
      then
            Class=$1
            shift
            Status=$1
            if test "$Status" = "G"
            then
                  shift
                  Identifier="$1"
                  shift

#  Paranoid Check.

                  if test "x$bugid" != "x$Identifier"
                  then
                        xx=`today -l`
                        echo "ddtsbackend $xx Daemon started - $$" >>
$home/spool/LOG
                        echo "ddtsbackend $xx INTERNAl ERROR can't sync on
$Identifier on behalf of $bugid" >> $home/spool/LOG
                        continue
                  fi

#  Get the version (may be more than one word so do it separately)

                  CP_release=`bugval -i $bugid Version`
                  CP_approver=`bugval -i $bugid Approver-id`
                  xx=`today -l`
                echo "ddtsbackend $xx Daemon started - $$" >> $home/spool/LOG
                  for i in $*
                  do
                        batchbug -l 100 -i $i -t
$home/class/$Class/link_semantics/G.tmpl CP_ID "$bugid" CP_approver
"$CP_approver" CP_release "$CP_release" &
                        xx=`today -l`
                        echo "ddtsbackend $xx Ran bugval(1) on $i on behalf of
$bugid" >> $home/spool/LOG
                  done
            fi
      else
            continue
      fi
      rm -f spool/dbackend/$bugid
done
xx=`today -l`
echo "ddtsbackend $xx Daemon Terminated normally - $$" >> $home/spool/LOG
```

As you can see this is a simple shell script. The *bugval* utility extracts fields from the just processed defect (Parent CP) and *batchbug* updates the linked defects (child PRs.)

Note that *batchbug* calls a simple template to do part of the processing. Here is that template file:

```
Go-on:       today

Approver-id:  log $Status -> G by $CP_approver
             log Approved via approval of $CP_ID by $CP_approver
             log Approved for Release $CP_release
             set Status G
             echo $CP_approver
```

The template file is used to write audit trail information into the History enclosure of all of the Children PRs. The template is also used to set the Children PR Status field to "G" (Approved).

The *software* class contains a directory called *link_semantics* that has example code for creating link semantic actions.

11

Handling ClearDDTS Mail

Throughout the defect life cycle, ClearDDTS uses UNIX electronic mail to notify users about bug submissions and state transitions. In addition, mail can be sent between separate ClearDDTS installations for remote submission and subscription. This chapter discusses how ClearDDTS uses mail and how that mail can be customized for your environment. The following topics are covered:

- Why use electronic mail?
- How ClearDDTS handles mail
- Determining who receives mail
- Customizing notification mail
- Sending mail to ClearDDTS

Why use electronic mail?

There are a number of benefits to using electronic mail for data transfer. For example, mail is usually the first network utility that is set up on a UNIX system, and one can safely assume that system administrators will keep it working. By piggy-backing on top of mail, ClearDDTS allows system administrators to easily manage the environment without having to administer yet another communications facility.

In addition, because electronic mail is so pervasive in the UNIX community, ClearDDTS can be connected between any two systems on the Internet or any two systems where a telephone exists. Thus connectivity can be sophisticated or simple and ClearDDTS will use whatever system is in place.

How ClearDDTS handles mail

Incoming mail for ClearDDTS usually contains copies of raw defect reports and automatic administration commands. A daemon reads this mail, and determines what should be done with the mail message.

The daemon forwards any items it cannot understand to the person who has been designated the ClearDDTS administrator. For example, mail about the company picnic sent to ClearDDTS would be forwarded to the ClearDDTS administrator. Usually such mail is from human users; these typically are questions about ClearDDTS, but occasionally they are "bounced" notification messages resulting from incorrect user mail addresses.

After reading and processing each mail message, ClearDDTS records the transaction, and deletes the message.

Note: The ClearDDTS administrator should not modify or delete ClearDDTS mail. ClearDDTS manages its own mail box with no help from human users.

Types of ClearDDTS mail

ClearDDTS defines several types of mail messages:

- Formatted bug mail sent to users. For example, this is the mail sent to an engineer when he or she is assigned a defect.
- Raw (ASCII) bug mail sent from one ClearDDTS site to another. This occurs when an engineer on one machine submits a defect against a project that resides on a remote machine.
- ClearDDTS administration commands. This type of mail has to do with connecting ClearDDTS systems together and with automatically making project information available over the ClearDDTS network. For example, when two systems are connected to share project information, the *adminbug conn* command generates mail from one site to another to "bind" the systems together. After that, any time a project is created on one

system it is automatically available for defect submission on the remote system.

- Formatted mail from users sent to ClearDDTS to submit a defect or append to an existing defect. For more information on this type of mail see, *Sending mail to ClearDDTS* on page 11-14.

Looking at an example

To see how ClearDDTS handles mail, assume that we have two ClearDDTS machines called *SFOqa* and *LDNcc*. These machines can be any place (say California and London). On each machine is one project. On *SFOqa* we have project XX and on machine *LDNcc* we have project ZZ.

ClearDDTS uses the *~ddts/conf/aliases* file to determine the mail addresses and aliases for the local host. Since formatted bug mail is only generated locally, the ClearDDTS administrator never has to worry about remote aliases.

Note: Global aliases are recognized as long as the mailer program specified in *adminbug inst* uses them. However, ClearDDTS does not explicitly use those aliases.

If a state transition occurs on *SFOqa* and a user on *LDNcc* must be notified, then ClearDDTS on *SFOqa* will send a raw bug transaction to *LDNcc*, and ClearDDTS on *LDNcc* will then send local formatted mail to the user on *LDNcc*.

Similarly, if an engineer on machine *SFOqa* submits a defect against a project ZZ that resides on the remote machine *LDNcc,* the raw bug (ASCII) data that comprises the transaction is sent from *SFOqa* to *LDNcc* where it is incorporated in *LDNcc's* database.

An acknowledgment is then sent back from *LDNcc* to *SFOqa* and *SFOqa's* database is updated to reflect the fact that the bug was received. This round trip handshake guarantees that no defect will ever be lost in the mail system.

Determining who receives mail

Formatted mail is generated for every state transition in four ways. The first three ways rely on fields defined within the defect. If the defect report has any of the following fields defined, then mail is sent to the address specified in that field:

- `Submitter-mail`: Mail address of the defect submitter. This mail is sent from the submitting ClearDDTS system if:

  the first 5 characters of the defect Identifier matches this site ID (i.e., submitted on this site)

  AND $Submitter-id is not null,

  AND $Submitter-mail is not null,

  AND $Submitter-id is not equal to $Updated-by.

- `Engineer-mail`: Mail address of the assigned engineer. This mail is sent from the machine that owns the project the defect was submitted against if:

  the $Engineer field is not null,

  AND $Engineer-mail is not null,

  AND $Engineer is not equal to $Updated-by.

  If the value of $Engineer has changed, mail is sent to both the old and new values of $Engineer.

- `Other-mail`: This field is left for user-defined mail needs. For example, you can configure ClearDDTS to send mail to a division manager if a defect is Severity 1 or affects a particular project. This mail is sent only from the machine owning the defect.

Note: By default, notification mail is not sent to the person performing the update transaction (the value of $Updated-by). To force e-mail to be sent to the updater, add the following line to the ~ddts/etc/ddtsrc file:

```
Allow-updater-mail: Y
```

Below is an example of how the *Submitter-mail* and *Engineer-mail* fields are set up in the *master.tmpl* field. If these fields are empty, no e-mail is generated.

```
#######################################################
Submitter-id:    if equals $STATE$OPERATION Sp
                 or null
                     whoami
                 fi

                 if equals $OPERATION v
                     "!%-.8s"
                 fi

Submitter-mail: if match $STATE$OPERATION Sp Sm Nm
                     if equals $Notify-submitter Y
                         get_email_address "$Submitter-id"
                     else
                         echo " "
                     fi
                 fi
                 .
                 .
                 .

Engineer:        if match $OPERATION v m
                 or match $STATE$OPERATION Ap Af
                     "\n\(6,41)LABORATORY INFORMATION"
                     "\(7,42)Assigned engineer: %-8.8s"
                 fi

                 if match $OPERATION p m
                     required
                     help user.H
                     goodusers
                 fi

Engineer-mail:  if match $OPERATION p m
                     get_email_address "$Engineer"
                 fi
#######################################################
```

Notification list

The fourth way formatted mail can be generated is from a *notification list*. A notification list is specified when a project is created. Whenever a bug enters a particular state, or is modified while in that state, users on that notification list receive a

formatted copy of the updated defect. The mail notification list is
defined in the *~ddts/projects/<project>/proj.notify* file . For example:

```
N-notify: chris dave
O-notify: chris
A-notify: chris
R-notify: chris dave
D-notify: chris
F-notify: chris
P-notify: chris
V-notify: chris
```

In this example, the user *chris* is notified of every new state
transition. The user *dave* is only notified of bugs entering or
modified in the New and Resolved states. The ClearDDTS
administrator and the project manager can modify this list with
the *adminbug mprj* command.

Customizing notification mail

You can control the format and content of notification mail by
customizing a notification template (*notify.tmpl*). You can create a
class-specific *notify.tmpl* or create user-specific templates. To
determine mail configuration parameters, ClearDDTS uses the
first template file it finds. It searches in the following order:

```
~ddts/www/user_prefs/<userid>.notify.tmpl
~ddts/class/$class/notify.tmpl
~ddts/class/$class/mail.subject (format prior to release 4.5)
```

Note: Using a notification template is optional. If you do not use a
notification template, ClearDDTS uses the default *mail.subject*
file only. The *mail.subject* file allows you to customize the subject
string of e-mail, but not the content. There are extensive
comments in the *mail.subject* file that describe how to do this.

Mail Domain

ClearDDTS uses the e-mail address to search for a user-specific
notification template. Since some users enter their domain name
(*chris@bigcorp.com* instead of just *chris*) in the notification lists
and mail fields, ClearDDTS must be able to strip off the domain
name prior to looking for the user-specific notification template. To

ensure that ClearDDTS can find the appropriate user-specific template, enter the following field in the *~ddts/etc/ddtsrc* file:

```
Notify-domain: @<your_domain>.com
```

With this field set, ClearDDTS can find the preferences file for the user *chris* (*chris.notify.tmpl*) even if the e-mail address found is *chris@bigcorp.com*. If the domain found does not match the domain in the *ddtsrc* file, ClearDDTS uses the entire e-mail address when looking for the notification template.

Debugging Tool

The options you can set for notification mail provide a great deal of flexibility. While configuring and testing your settings, you can have more verbose output for notification mail generated in the *~ddts/spool/LOG* file. To do this add the following field to the *~ddts/etc/ddtsrc* file:

```
Bugmail-verbose-log: Y
```

The additional output can include information on the notification template used, user e-mail address, message-template used, and other options.

Notification Options

The notification template file (either class-specific or user-specific) uses the following fields to control the e-mail parameters:

- *Message-template*: Defines the path to the template file to use for message content.
- *Subject*: Determines the subject line of the e-mail.
- *Suppress-mail*: Suppresses mail if no unsuppressed changes occurred.
- *Bugmail-ignore-fields-file*: Defines the path to the file containing fields to be suppressed.
- *Display-added enclosures*: Determines whether newly added enclosures appear in the change log section of the e-mail.

- *Show-enclosures-on-submit*: Determines whether enclosures are included in the change log section of the e-mail for new defects.
- *Bugmail-diff-command*: Defines the path to the script used to evaluate the differences between enclosures.

Each of these is described in more detail in the following sections. For an example, look at the *~ddts/class/software/notify.tmpl.* (If you are upgrading from a previous release this file can be found in *~ddts/NEWCLEARDDTS/class/software/notify.tmpl*.) It is helpful to follow the example while reading the details about each option.

Special variables

Notification mail sets certain variables that can be used to derive the fields appearing in the mail (see Message-template below). Defects are looked at in both their old version (before editing) and new version. The following variables are defined before running the notify.tmpl file in the context of the new version of the defect:

Variable	Description
$OLDSTATUS	The previous state letter of the defect record.
$TRANSITION	State transition (new state letter) for which the mail is being generated.
$TRANSITION-TYPE	Analysis of changes to the defect: "new" - no old bug "transition" - State/Class/Project changed "assign" - engineer changed "modify" - something changed
$RECIPIENT	The e-mail address to which we are sending mail.

In addition, all the old fields are also available to the notification template and have been renamed to have "--" added to the beginning of the name. For example, the old headline can be accessed using the name "$--Headline" while the new headline is accessed with "$Headline".

Message-template

The *Message-template* field defines the template file used to determine which defect fields to include in the notification e-mail. By default, this field contains the path to the *master.tmpl*. You can create your own message template (or multiple templates) to include only the fields you are interested in. If the *Message-template* field points to a non-existent file, no e-mail is sent. For example:

```
Message-template:
    if equals $TRANSITION-TYPE modify
        echo $~/class/$Class/modify.tmpl
    elif equals $TRANSITION-TYPE assign
        echo $~/class/$Class/engr.tmpl
    elif not equals "$Project" "$--Project"
    and not equals $TRANSITION-TYPE new
        echo $~/etc/forward.tmpl
    elif equals $TRANSITION O
        echo ""
    elif equals "$Showstopper" Y
        echo $~/class/$Class/showst.tmpl
    else
        echo $~/class/$Class/master.tmpl
    fi
```

In this example, the $TRANSITION-TYPE and $TRANSITION variables are used to determine which message template to use for formatting the mail. It is defined as follows:

- If the transition type is modify (some field has changed) then the modify.tmpl is used.

- If the transition type is assign then the engr.tmpl is used.

- If the Project has changed (comparing the old and new values of the Project field), then we assume the defect has been forwarded and the forward.tmpl is used. Notice that the forward.tmpl is not located in the class directory. Templates can be located outside the class directory (for example, when they are used by more than one class).

- If the state transition is to the open state no mail is sent (no message template is defined).

- If the Showstopper field is set to yes, the showst.tml is used.

- If none of the other conditions are met the master.tmpl is used as the message-template.

Change_history filter

The *change_history* filter determines what changes have been made to a defect. In order to have a list of changes (the field change log section) appear in notification mail you must include the *change_history* filter in a field of the message template. How you do this depends on the message template you are using.

In the default *master.tmpl* (which is also the default message template), OPERATION n is used by ClearDDTS to process notification mail. In the default *master.tmpl* file, the Begin derivation tests for OPERATION n and sets Putmail as follows:

```
Begin:        unset Begin
              .
              .
              .
              if equals $OPERATION n    # mail notification?
                  set OPERATION v        # yes, pretend viewing,
                  set Do-enclosures Y    # but also show enclosures,
                  set Putmail Y          # and process mail.
              elif equals $OPERATION l  # dumpbug w/enclosures?
                  set OPERATION v        # yes, pretend viewing,
                  set Do-enclosures Y    # but also show enclosures,
                  set Putmail N          # ...with no mail processing
               else
                  set Do-enclosures N    # don't show enclosures
               fi
```

When OPERATION is n, ClearDDTS changes OPERATION to v with the addition of showing enclosures (Do-enclosures) and processing mail (Putmail).

Because Putmail is set in the Begin derivation you must add the *change_history* filter to the Putmail field derivation in the default *master.tmpl* (it is the last derivation in the default *master.tmpl*):

```
Putmail:    unset Putmail
            if equals "Y"
                  change_history "Changes to this defect include:"
            fi
```

To set up a change log when using a different message template, you must include a field derivation containing *change_history* in that template. For example:

```
Changes:
        unset Change_history
        "\n"
        change_history "Changes to this defect include:"
        "\n"
```

If you choose to include the change log you can also choose where in the e-mail message it should appear. To do this, move the entire derivation containing the *change_history* filter to a new location in the message template. For example, in the *master.tmpl* file you can put it between the Begin and Identifier field derivations to make the change log appear at the beginning of the notification e-mail.

The *change_history* filter can accept up to two optional arguments:

`-f`	Only include the change log for fields and not enclosures.
`-e`	Only include the change log for enclosures and not fields.
`"title"`	Uses this text as a title for the change log section of the e-mail. The title must be enclosed in quotes. If a title is not supplied, ClearDDTS uses the following default: +++++ Field change log (added to mail by bugmail) +++++

Defect access with webddts

Notification mail can contain a URL to access the referenced defect using the *webddts* interface. If you have web integration with your e-mail tool, this provides a way to quickly launch *webddts* and view the defect. A sample derivation containing a URL might be:

```
Putmail:    unset Putmail
            change_history "Changes to this defect include:"
        "\nThe URL for the defect is:\n"
        "http://web.server/ddts/ddts_main?id=$Identifier\n\n"
```

Subject

This field defines the subject line of the notification e-mail. Typically the subject line includes the defect Identifier and a message appropriate to the transaction. For example:

```
Subject:     echo "$Project defect $Identifier"

             if equals $TRANSITION S
                 echo "$$ was submitted"
             elif equals $OLDSTATUS$TRANSITION SN
                 echo "$$ arrived"
                 break
              elif equals $OLDSTATUS $TRANSITION
                 echo "$$ was updated"
             else
                 echo "$$ was moved to state $TRANSITION"
             fi
```

Suppress-mail and Bugmail-ignore-fields file

You can choose to suppress notification of certain changes to a defect. For example, suppose you do not want to be notified if the only change to a defect is an update to the Timestamp. To suppress notification changes about the Timestamp field you would:

1 Make sure the *change_history* filter is being run in the message template (see above).

2 If you are using a class-specific notification template, create the file *~ddts/class/$class/bugmail_ignore_fields*.

If you are using a user-specific notification template, create the file *~ddts/www/user_prefs/<userid>_ignore_fields*.

This is a dfile that contains a list of the fields to be suppressed.

3 Add the field name followed by a colon to the file. For example:

```
Timestamp:
```

4 In your notification template, define the path to the appropriate *bugmail_ignore_fields* file. For example:

```
Bugmail-ignore-fields-file:
        echo ~ddts/www/user_prefs/<userid>_ignore_fields
```

5 In your notification template, set the *Suppress-mail* field to *Y*.

```
Suppress-mail:
        echo Y
```

When the *Suppress-mail* field is set to *Y*, changed fields are compared to those listed in the *bugmail_ignore_fields* file. Notification mail is not sent if the only changes to a defect are to suppressed fields.

Note: You can also suppress enclosure changes by adding certain enclosure fields to the *bugmail_ignore_fields* file as in the following examples:

```
Related-file::::Problem
History::::
```

Display-added-enclosures

Use the *Display-added-enclosures* field to determine if the *change_history* filter should display newly added enclosures (to existing defects) in the change log section of notification mail. If this field is set to *Y*, the entire enclosure appears in the e-mail. If this field is set to *N*, only the line "Enclosure <title> has been added" appears in the e-mail. Note that this option does not control enclosures for a newly submitted defect (see below).

Show-enclosures-on-submit

Use the *Show-enclosures-on-submit* field to determine if the *change_history* filter should display enclosures for newly submitted defects in the change log section of the notification mail. If this field is set to *Y*, enclosures in the new defect appear in the e-mail. If this field is set to *N*, only the line "New record, no historical data" appears in the e-mail. This option does not control e-mail for existing defects, and is not affected by the *Display-added-enclosures* field.

Bugmail-diff-command

If you are running the *change_history* filter in your message template, changes to enclosures can also be compared. The script used to do the comparison is defined in the *Bugmail-diff-command*

field. By default, this field is set to *~ddts/bin/bugmail_diff*. You can edit this file or create your own file. If you create your own, remember to set the *bugmail-diff-command* field to the path for your new script.

If the *bugmail_diff* script finds differences to report, the changed text is printed in the changes section of the notification mail under the heading "Enclosure Changed: <title> ".

Mail for changed sites

If your site reaches 100,000 defects and you change the site identifier (see the Rational Software web site for an FAQ on how to do this), ClearDDTS needs to know about the old site identifier. To ensure that notification mail can reach users for defects with an old site identifier, add the old site identifier to the *~ddts/etc/ddtsrc* file as follows:

```
Old-Machids: BUGno XXXxx
```

The notification mail process compares the site identifier of a defect to the current site and to values set in this field. If it finds a match, it acts like it owns the defect and is able to send notification mail.

Sending mail to ClearDDTS

You can easily mail a defect to ClearDDTS. To do this, use the e-mail format described in Appendix D. You may also want to refer to the *ddtsmailbug* shell script for an example. This script, located in the *~ddts/bin* directory, includes a customization description.

You can also send mail to ClearDDTS that is appended to a defect as an enclosure. If you send mail (or reply to notification mail) that contains a defect ID for the current site in the subject of the message, that message is appended to the defect in an enclosure. If a defect ID is not found in the subject, a "Confusing mail" message is sent to the ClearDDTS administrator.

ClearDDTS also searches the subject for "enclosure=", and uses the word following as the enclosure title. If ClearDDTS cannot determine the name of the enclosure, it defaults to "mail_log".

When the message is appended to the defect, you will see text similar to the following in the enclosure:

```
batchbug 981116 164457 Appended via email by "chris@bigcorp.com"

<body of e-mail message>

batchbug 981116 164457 End of message
```

ClearDDTS checks the sender and prevents "root", "mailer-daemon", "daemon", "postmaster", and names starting with "ddts" from being appended. For more information on appended mail to defects and how to customize this feature see the man page for *ddtsappend*(1).

12

Managing ClearDDTS Security

ClearDDTS supports several types of security. This chapter discusses these security mechanisms and how they are administered. Topics covered include:

- HTTP (web) security which controls access to ClearDDTS via the web
- Write Access Control which determines who is allowed to make changes to defect records.
- Read Access Control which determines who may view defects.
- Field-level security (*xddts* specific)

HTTP (web) security

HTTP security is used to specify which users can access ClearDDTS via the *webddts* interface, and also provides a mechanism for identifying users who access ClearDDTS. Project and defect-level access control depend on the $REMOTE_USER variable to determine the identity of a user. The only way to ensure that the $REMOTE_USER variable contains valid information regarding the user's identity is to set up HTTP username/password access control.

Identifying the user

Identifying a user accessing ClearDDTS data is important in enforcing security. When ClearDDTS is accessed in a UNIX environment, users are identified by their UNIX login name. Identifying users through the web is a more complicated process.

On the web, the *webddts* programs and scripts are run by the user who owns the HTTP server installation (usually the user ID *http*).

For ClearDDTS security to be effective, the true identity of the user must be established. To establish a user's identity, HTTP security does the following:

- When a user connects to *webddts*, the web server sees the security controls (detailed later in this chapter) and prompts for a valid user ID and password.
- If the user ID and password are valid, the server allows access and sets a special variable called $REMOTE_USER to the value of the user ID entered.

What happens when HTTP security is not implemented

If you do not set up HTTP access control for the *webddts* pages, any defect and project-level control you define can be overridden.

When HTTP username/password access control is set up, any user attempting to access ClearDDTS must enter a username and password. If the user enters a valid username and password, the $REMOTE_USER variable is set to the value specified in the username field.

When HTTP security is not set up, *webddts* sets the value of the $REMOTE_USER variable by asking the user to enter a *Login* name. The user is not required to enter a password and can enter any character string in the Login field. Whatever string the user enters is used to set the $REMOTE_USER variable. Because any character string is valid:

- Virtually any user with a Web browser and an Internet connection can access ClearDDTS.
- An unauthorized user can enter the username of an authorized user and override any project or defect-level security defined.

Controlling access to web pages

Using your HTTP server security mechanisms, you can require a user to enter a valid name and password to access a particular directory and the documents stored in that directory. By implementing this form of security, you can control which users have access to the directory containing the *webddts* pages.

To restrict directory access by username and password, UNIX-based HTTP servers use password and group files similar to UNIX password and group files. To give you an example, the Apache HTTP server procedures for setting up username/password-based directory security are described here. To determine the exact procedures for your HTTP server, see your server documentation.

Note: For pointers to World Wide Web resources that provide information on setting up HTTP server security, see Appendix H, Information Resources on the Web.

Using the Apache HTTP server, there are two ways to add username and password protection to the directory containing the *webddts* pages:

- Editing the `access.conf` central access configuration file
- Adding the file `.htaccess` to the directory

Although both procedures are described here, editing the `access.conf` central access configuration file is considered a better method for three reasons:

- If you want to set up security for multiple directories, it is easier to manage one central configuration file than it is to manage multiple `.htaccess` files spread throughout your directory system.
- An `.htaccess` file, unlike the `access.conf` file, can be fetched as a Uniform Resource Locator (URL) by a user accessing your site through a Web browser.
- There is a greater possibility that an `.htaccess` file, as opposed to an `access.conf` file, will be modified or overwritten.

Editing the access.conf central access configuration file

To set up access control for the *webddts* directory by editing
`access.conf`:

1 Create the password file `.htpasswd` and add an authorized user
to the file.

Use the Apache `htpasswd` program, located in the `support`
directory, to create the `.htpasswd` file. You may need to compile
the program first (see the Apache documentation).

Run the following command the first time you invoke the
`htpasswd` program:

```
htpasswd -c [path_to_passwordfile]/.htpasswd [username]
```

The `-c` flag in this command indicates that you are creating a new
password file. Replace `[path_to_passwordfile]` with the path
to where the `.htpasswd` file is to be stored. Replace `[username]`
with the name of an authorized ClearDDTS web user to be added
to the file.

When you execute the above command, the `htpasswd` program
prompts you to enter a password for the user specified in the
`[username]` variable. Once you enter the password, the
`htpasswd` program stores the username and password in the
`.htpasswd` file.

2 Add additional users to the `.htpasswd` file.

Run the `htpasswd` command for each web user who needs access
to ClearDDTS. Since you already created the `.htpasswd` file in
the previous step, leave the `-c` flag out of the command:

```
htpasswd [path_to_passwordfile]/.htpasswd [username]
```

3 Edit the `access.conf` file to secure the *webddts* directory.

To ensure that only users listed in the `.htpasswd` file can access the *webddts* directory, add the following language to the `access.conf` file:

```
<Directory full_path_to_protected_directory>
      AuthName         [any_name]
      AuthType         Basic
      AuthUserFile     [path_to_passwordfile]/.htpasswd
      require valid-user
</Directory>
```

Replace `[full_path_to_protected_directory]` with the path to the directory where the *webddts* interface is installed.

Replace `[any_name]` with descriptive text about the application whose security you are setting up in this section of code. This text appears as part of the username/password prompt displayed by the HTTP server when a user accesses *webddts*. For example, since you are defining which users have access to the *webddts* directory, you can replace `[any_name]` with the text *DDTSusers*.

Replace `[path_to_passwordfile]` with the path to the directory where the `.htpasswd` file is stored (see steps 1 and 2).

4 Reinitialize the HTTP server.

Before your HTTP server can detect and use the changes made in steps 1–3, it must reread the HTTP configuration files. To signal the HTTP server to reread the configuration files, issue the command `kill -1 [PID]`, substituting `[PID]` with the server daemon *process ID* owned by root. This command does not stop and restart the server, but it does signal the server to reread the configuration files and to stop and restart any subordinate processes initiated by the HTTP server daemon.

Adding the file .htaccess to the directory

The alternative method for adding username and password protection is adding the .htacess file to the directory. To set up

access control for the *webddts* directory by adding the file
`.htaccess`:

1 Create the password file `.htpasswd` and add an authorized user
to the file.

See step 1 in the above section "Editing the access.conf central
access configuration file."

2 Add additional users to the `.htpasswd` file.

See step 2 in the above section "Editing the access.conf central
access configuration file."

3 Create a file called `.htaccess` in the directory where the *webddts*
interface is installed.

Include the following information in the `.htaccess` file:

```
AuthName          [any_name]
AuthType          Basic
AuthUserFile      [path_to_passwordfile]/.htpasswd

<Limit>
require valid-user
</Limit>
```

Replace [any_name] with descriptive text about the application
whose security you are setting up in this section of code. This text
appears as part of the username/password prompt displayed by
the HTTP server when a user accesses *webddts*. For example,
when defining which users have access to the *webddts* directory,
you can replace [any_name] with the text *DDTSusers*.

Replace [path_to_passwordfile] with the path to the
directory where the .htpasswd file is stored.

Controlling access to data across the network

ClearDDTS does not use any encryption algorithm to control
communications between Web browsers and the HTTP server
where the *webddts* interface is installed. To secure data across
your network, you must use an HTTP server that supports data
encryption.

Monitoring access to webddts pages

To monitor user access to specific *webddts* pages, view the access information in the HTTP access log files. Because the names and locations of the HTTP access log files are different for different servers, check your HTTP server documentation for information about these files. Also, see your server documentation for instructions on how to modify the contents of these files.

Write access control

Write Access Control by project can be accomplished using the *adminbug* utility. The most important way to control who can change records in ClearDDTS is on a per-project basis. When projects are created with the *adminbug aprj* command, you are asked to identify who is allowed to change each state that a project may go through. You do this by specifying user IDs (UNIX or ClearDDTS aliases), variables from a defect, variables from the `proj.control` file, or groups (UNIX and/or ClearDDTS defined). The following example shows some of the questions asked:

```
List LOGIN names of users allowed to ASSIGN (A state) a
bug. Just hit return if anyone is allowed to ASSIGN bugs.
mmanley

List GROUP names of groups allowed to ASSIGN (A state) a bug. Just hit
return if any group is allowed to ASSIGN bugs.
proj_managers

List LOGIN names of users allowed to RESOLVE (R state) a
bug. Just hit return if anyone is allowed to RESOLVE bugs.
$Engineer $A-allow

List GROUP names of groups allowed to RESOLVE (R state) a
bug. Just hit return if any group is allowed to RESOLVE bugs.
$A-allow-group
```

In this example, the user *mmanley* and users in the group *proj_managers* are allowed to assign defects for repair. They are the only users allowed to move a bug into the Assigned (A) state and the only users allowed to modify a defect once in that state. The group can either be defined in `~ddts/conf/groups` or be UNIX system defined.

Similarly, only the engineer assigned to fix a defect (the value of the variable $Engineer), users who are allowed to assign defects (the value of $A-allow), and groups allowed to assign defects (the value of $A-allow-group) may move a defect to the Resolved (R) state or modify defects in that state.

When choosing variables to use for defining access control, follow these rules:

- Any field from a defect can be specified as long as it expands into a list of users (for example, $Engineer and $Submitter-id). The definition of the field cannot include a nested definition. For example, the value of $My-field cannot expand to "Sub-field1 Sub-field2" even if those fields would further expand to a list of users.

- Any field from the proj.control file (A-allow, O-allow, and so on) can be specified. The proj.control variable definitions can be nested. In the earlier example, R-allow is defined as $ENGINEER $A-allow. This expands to the name of the assigned engineer and the user *mmanley*.

- The proj.control variable definitions for <State>-allow-group work the same way as those for <State>-allow fields, except the variables need to expand into group lists.

- The two types of proj.control file fields, <State>-allow and <State>-allow-group, cannot be used to define each other. For example, the proj.control fields for <State>-allow cannot be defined using <State>-allow-group fields because these expand to a list of groups not a list of users. In our earlier example, you would not define the list of users allowed to resolve a defect as:

$Engineer $A-allow $A-allow-group

This would expand to the assigned engineer, *mmanley*, and *proj-managers*. Since *proj-managers* is a group not a user, this value would not be valid.

- When defining groups in `~ddts/conf/groups`, the definition of the group must be a list of users. A group cannot be defined in terms of another group. For example:

proj-managers: mmanly chris

To make changes to any of these parameters, the ClearDDTS administrator or project manager can run the *adminbug mprj* command.

Read access control

ClearDDTS supports Read Access Control on a per-project, or per-defect basis.

Per-project read access control (adminbug)

To enable read access control on a per-project basis, you need to add the following line to the *~ddts/etc/ddtsrc* file:

```
Proj-read: Y
```

Once you enable this feature, the ClearDDTS *adminbug aprj* and *mprj* commands allow you to establish read access control with the following prompts:

```
List LOGIN names of users allowed to view a bug. Just hit return if anyone
is allowed to view bugs.
mmanley

List GROUP names of groups allowed to view a bug. Just hit return if any
group is allowed to view bugs.
design_team
```

In this example, the only users allowed to view defects are *mmanley* and members of the UNIX group *design_team*. If these fields are left blank anyone may view defects. The ClearDDTS administrator or the project manager can run the *adminbug mprj* command to change these parameters at any time.

Note: Per-project read access control is not supported for subscribed projects unless the remote modification option is used. This is because the remote modification option allows you to have a local proj.control file for the subscribed project.

Per-defect read access control

To enable read access control on a per-defect basis, you need to add the following line to the *~ddts/etc/ddtsrc* file:

```
Bug-read: Y
```

Once you enable this feature, ClearDDTS assigns a *security token* to each user and marks each submitted defect with a security token in the Security-token field. The content of the Security-token field is set by looking up the user login name in the file *~ddts/etc/security*. A sample security file is shown below:

```
xerox: xe
kodak: ko
rico:  ibm
tracy: ko xe
OTHER: xxx
```

In this example, if the user *tracy* submits a defect, the value of the Security-token field is set to "*ko xe*."

Read access to the defect is granted if any security token in the defect Security-token field matches one of a requesting user's security tokens. In the example above, the defect submitted by user *tracy* would be viewable by users *xerox* and *kodak* but not by *rico*.

Note: Enabling per-project read access control disables per-defect read access control for subscribed projects unless the remote modification option is used.

One way to use this mechanism is with competing vendors. Suppose that you want to allow individuals from the Kodak and Xerox corporations to submit defects against the same project but you do not want Kodak to see Xerox defects and likewise Xerox should not be allowed to view Kodak defects. This can be accomplished by setting a different security token for all Kodak logins (or perhaps just one *kodak* login) and for all Xerox logins. In this example only the user *tracy* would be able to see both Kodak and Xerox defects.

Using the OTHER security token

The *OTHER* login name in the security file is special. If a user submitting or viewing a defect is not in the security file list, then the security token from the *OTHER* field is used. If *OTHER* is not in the *~ddts/etc/security* file, then no security token is added to the defect and anyone may view the defect.

The example above shows an actual situation where all members of the design and QA teams needed access to all defects but Kodak and Xerox employees were limited to viewing only those defects that members of their company had submitted.

Access was granted to the design team through Per-Project Read Access Control and denied to Xerox and Kodak employees through Per-defect Read Access Control. If the *OTHER* field was set to "ko xe" or was missing altogether, then Kodak and Xerox would also be granted read access to defects submitted by the project team and the QA group. If the *OTHER* field was set (for example to *xxx*), Kodak and Xerox could only view the defects that they submitted (or defects marked viewable by the *View* field described below).

If read access is denied, a message appears saying that some defects are not available because of access control.

Making defects viewable

There are times when a defect should be viewable by everyone regardless of who submitted the defect. Marking a defect as viewable regardless of the security token may be done in the *master.tmpl* file. If the value of the *View* field is set to *Y,* then the defect is marked as viewable by everyone. (For example, this would allow Xerox to view a defect submitted by Kodak.)

See Figure 12-1 for an overview of ClearDDTS read access control.

Figure 12-1: Summary of Read Access Control Logic

Project or Bug Read Access Control Implemented

N

Can View Bug Report

Y

User logged in as *ddts*?

Y

N

Am I the Defect Submitter?

Y

N

What's in the View FIeld?

Y

N or Blank

Project Read Access Control Setup?

Y

Is Login ID OK?

Y

adminbug aprj/mprj

N

Is Group OK?

Y

adminbug aprj/mprj

N

N

Bug Read Access Control Setup?

Y

Does User Security Token Match Bug Security?

Y

N

N

Permission Denied
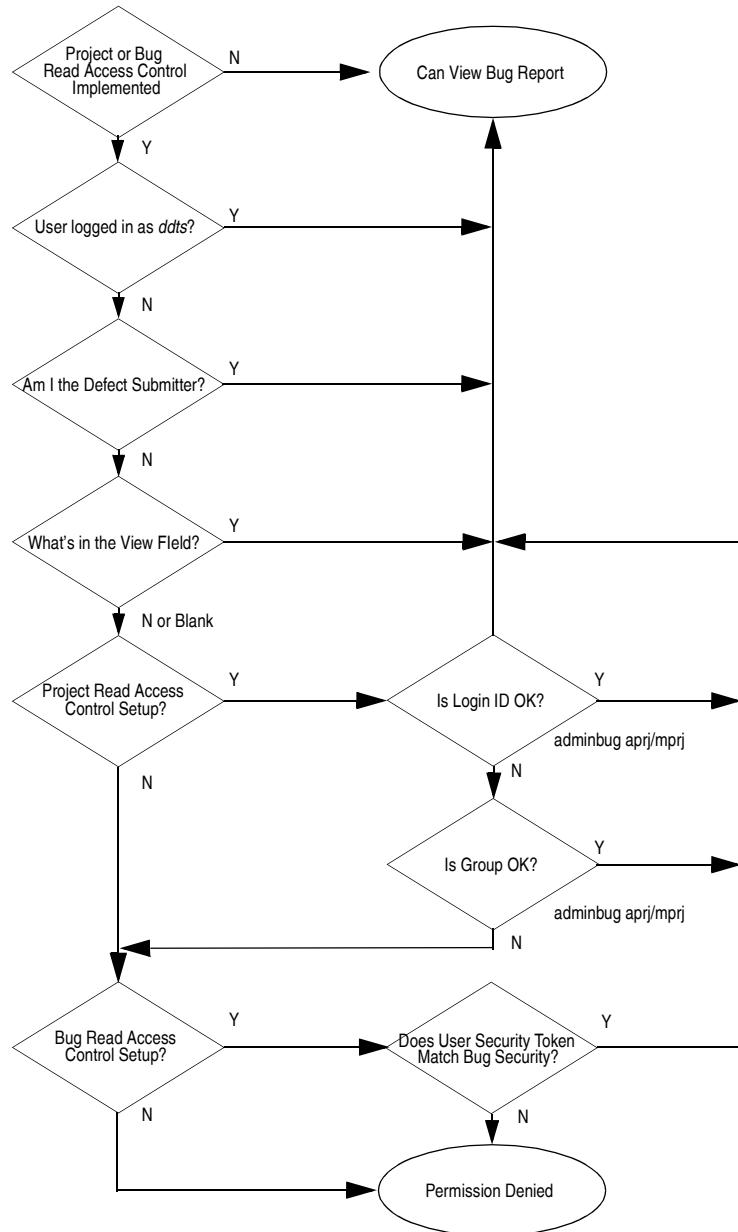
xddts specific security

Controlling field access by customizing the master.tmpl file

You can control the access to each field in a defect record by customizing the *master.tmpl* file. For example, the standard Problem field in the *master.tmpl* file is derived as follows:

```
Problem: if match $OPERATION v m
        or match $STATE$OPERATION Op Of
            "\(8,42)\kProblem type: %-14.14s"
        fi
        if match $OPERATION p m
            oneof -f problems
            required
        fi
```

To implement write access control on this field, you might change this field derivation to:

```
Foo:    if match $OPERATION p m
            unset Foo
            whoami
            if not oneof -f oklist
                goto Bar
            fi
        fi

Problem: if match $OPERATION v m
        or match $STATE$OPERATION Op Of
            "\(8,42)\kProblem Type: %-14.14s"
        fi
        if match $OPERATION p m
            oneof -f problems
            required
        fi

Bar:    .
        .
        .
```

In this case, only the users listed in the *oklist* file are allowed to modify the value of the field. Anyone is allowed to read the value of the field.

Field read access control

You can also provide read access control on each field in a defect record by customizing the *master.tmpl* file. For example, the standard Problem field in the *master.tmpl* file is derived as follows:

```
Problem: if match $OPERATION v m
         or match $STATE$OPERATION Op Of
             "\(8,42)\kProblem type: %-14.14s"
         fi
         if match $OPERATION p m
             oneof -f problems
             required
         fi
```

To implement Read Access Control on this field, you might change this field derivation to:

```
Foo:     if match $OPERATION v
         and not oktoview
             goto Bar
         fi
                .
                .
                .
Problem: if match $OPERATION v m
         or match $STATE$OPERATION Op Of
             "\(8,42)\kProblem type: %-14.14s"
         fi

         if match $OPERATION p m
             oneof -f problems
             required
         fi

Bar:     . . .
```

In this case, a user-defined program *oktoview* is run to see if this user has permission to view certain fields. If so, the Problem field is displayed on the screen. If not, the Problem field is not displayed.

13

Managing and Customizing the ClearDDTS Database

This chapter describes the ClearDDTS database, how to perform database maintenance, and how to modify the database schema. Topics covered include:

▪ How information is posted to the database

▪ Backing up and restoring the database

▪ Reviewing the database schema

▪ Modifying the database

How information is posted to the database

In ClearDDTS, your defect records are stored in flat files in the `allbugs` directory. These files represent the "real" data and are used to post "copies" of the data to the SQL database. The SQL database acts as a cache to provide more efficient querying and performance.

When a user updates or generates a new defect record, the ClearDDTS daemon process, *ddtsd*, checks the *~dts/spool* directory and launches the *bugs.in* program to post the information to the database. The *bugs.in* program posts all defect information into a flat file in the *allbugs* directory, and only posts fields defined in the *schema_file* and mapped by the *database.cfg* file into the

ClearDDTS SQL database. The following figure provides a simplified view of the key components.
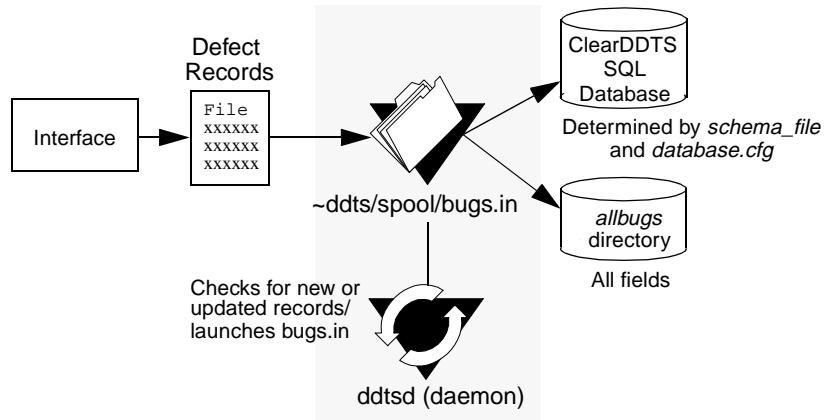
Backing up and restoring the database

As with any database, it is extremely important that you develop a backup strategy and perform regular database backups. To back up the database, simply back up the *allbugs* directory using a command such as *tar*. For example to back up the database to device *xxx*, run the following commands as user *root* or *ddts*:

```
cd ~ddts
tar cvf /dev/xxx ~ddts/allbugs
```

You can then restore the database using the same method. For example:

```
cd ~ddts
tar xvf /dev/xxx
```

After restoring the *allbugs* directory, you need to rebuild the SQL database with the *adminbug dbms* command.

Note: You should also occasionally back up the entire ClearDDTS home directory much as you back up normal user directories (for example, using *tar*). Complete backups help to protect your customized files as well as the actual data in the database.

Reviewing the database schema

As shipped, the ClearDDTS database consists of the following tables:

```
defects
enclosures
change_history
```

You can modify the ClearDDTS database by adding or deleting fields in these tables. For the most up-to-date information, you can view the contents of the schema file directly in *~ddts/dbms/ddts/schema_file*.

Note: If you are using an external database, see Appendix G, Using an Oracle Database, for information on tables.

Modifying the database

The ClearDDTS administrator (*ddts*) can make changes to existing database tables. For example, you may want to add a new field or change the definition of an existing field.

Note: The three default ClearDDTS tables are required and should not be deleted. You can create new tables for your own use (and access them directly through ddtssql), but they will not be recognized by ClearDDTS functions.

In general, any change to the database involves the following steps:

1 Log on as *ddts* on the machine where the ClearDDTS server is running.

2 Edit the *~ddts/dbms/ddts/schema_file* to reflect the change you want to make. This file contains the table and index definitions, including field types and lengths. See Appendix F, Database Reference, for information about ClearDDTS tables and indexes.

3 Edit the *~ddts/dbms/ddts/database.cfg* file to reflect the change you want to make. This file maps the field and table names from their format in the allbugs files to SQL database columns and tables.

4 Run *adminbug dbms* to rebuild the SQL database.

5 Edit the master.tmpl file to use the changes you have made. See "Editing the master template file (master.tmpl)" on page 8-11.

Editing the schema file

Edit the *~ddts/dbms/ddts/schema_file* to reflect the change you want to make. This file contains the table and index definitions. Entries in this file use the format:

```
begin table <tablename>
      fieldname size
      fieldname size
end table
```

For example:

```
begin table change_history
      identifier 10
      change_date datetime
      engineer 16
      text 0
end table
```

Data types

ClearDDTS has three types of database fields (three data types), two for general use, and one for system use only:

Customer use

- *character*: For character fields, you can set the field size by specifying an integer greater than 0 after the field name.

- *datetime*: For fields that hold date information, specify *datetime* as the field size after the field name. Note that time is ignored in the SQL database.

System use only

- *variable*: Variable fields are character fields with no limit on their size. ClearDDTS uses variable fields for enclosures in the enclosures and change_history tables only. For variable fields, the field size is set to zero (0) after the field name. Each table can have only one variable length field. For performance

reasons, variable fields must *always* be listed at the end of the table definition. Variable fields are for internal ClearDDTS use only.

Indexes

Indexes for a table immediately follow the table definition. For indexes, the format is:

```
begin index <indexname> dup|nodup
      key A|D
end index
```

After the index name, you specify whether the index requires a unique key (*nodup*) or allows duplicates (*dup*). On the next line(s) you specify the table columns that comprise this index and whether this is an ascending (*A*) or descending (*D*) index. For example:

```
begin index bugid nodup
      identifier D
end index
```

You may create additional indexes, or add additional keys to an existing index; however, you should never have more than 8 columns in an index or more than 16 indexes for any table.

For every update to the database, indexes are also updated. Adding indexes can make database queries for the indexed fields faster, however, having many indexes will make writing changes to the database slower.

Note: As shipped, the ClearDDTS database consists of the following tables: *defects*, *enclosures*, and *change_history*. For the field names, data types, and sizes associated with these tables, see Appendix F, Database Reference.

Editing the database configuration file

The *~ddts/dbms/ddts/database.cfg* file acts as a mapping mechanism. It identifies where the information entered in fields should be placed in the database (table and column name).

The format of entries in this file is as follows:

```
FieldName: table.column_name [table.column_name . . .]
```

For example, if you are adding a field called Sub-Project-Manager to the defect record and you have added a new column for *sub_proj_mgr* to the *defects* table in your schema file, you would locate the appropriate field position in the database configuration file and enter:

```
Sub-Project-Manager: defects.sub_proj_mgr
```

The field name and the name you enter in the database configuration file need to be the same. As the example shows, the only exceptions are for capitalization and the use of underscores or hyphens.

Rebuilding the database

After you have finished making your changes, run the *adminbug dbms* command to rebuild the database.

Remember that when you make changes to the database you may also need to make changes to the *master.tmpl* file (for example to incorporate a new field). See Chapter 8, Customizing ClearDDTS.

Warning: Binary data cannot be sent to the SQL database. For this reason, the *adminbug dbms* command checks for binary data. If any is found in a normal field an error is issued and the defect is not loaded into the SQL database. Any binary found in an enclosure is converted to spaces before being placed in the SQL database.

Note that the *allbugs* file is never changed in this manner. Only the copy in the SQL database is modified, and only for enclosures. The History file is not a user editable enclosure and is treated as a normal field in this regard.

14 Using the ClearDDTS SQL Interface

This chapter provides an introduction to the ClearDDTS command line query program, *ddtssql*, and provides examples of how your can use SQL with ClearDDTS. It also describes the limits and unique features of the ClearDDTS database server. Topics covered include:

- Learning SQL (how to create queries)
- ClearDDTS and standard SQL
- Recommended reading

Learning SQL

The ClearDDTS database is a fairly small database with only a few standard *tables*. Each table consists of a number of different *columns* and each column holds specific information about your defect records. The following example provides a simplified illustration of a table (not all columns are included):

Table Name ⟶ defects

Columns ⟶

identifier	headline	engineer
QTKqa01001	Defect record 1	roberts
QTLqa01002	Defect record 2	smith
QTKqa01003	Defect record 3	jones
QTKqa01004	Defect record 4	roberts

Rows of data

To retrieve information from the database, you construct a *query* that requests the information you want to see or work with. Each query helps you perform an action, such as select records to view, insert new records, update existing records with new information, or delete records.

In ClearDDTS, queries are written using a subset of the standard Structured Query Language (SQL). SQL is very simple to learn but is also very powerful. This section introduces some of the most common SQL statements; however, with SQL you can create far more complex queries than those described here.

Note: Although they may be shown in uppercase in this manual, SQL commands are not case-sensitive; only actual data is case-sensitive.

Starting the SQL command line interface

To start the SQL command line interface to the ClearDDTS database, *ddtssql*, enter:

```
ddtssql
```

You should now see the line prompt:

```
1>
```

You can now enter your query, pressing RETURN to move to a new line. To execute the query, enter *go* (on a new line) or a semi-colon (;) at the end of the query or on a new line. For example:

```
1> SELECT engineer
2> FROM defects
3> go

1> SELECT engineer from defects;
```

The result of your query displays and the first line prompt (1>) returns so you can enter a new query. To repeat a query, enter two exclamation points (!!).

To see what tables are available, use the *help* command and press RETURN. For example:

```
1> help

table name
-------------------
defects
enclosures
change_history
```

To see the details about a specific table, enter *help* and the name of the table, then press RETURN. For example, to see the details about the *enclosures* table, do the following:

```
1> help enclosures
fieldname          size  type
-------------------------------------
identifier          10    char
name                32    char
operation           16    char
op_date             11    datetime
engineer            16    char
text                 0    long
```

Writing Queries

The most common activity you will perform is retrieving information from the database. For example, to see complete information about all of the defects in the system, you could use the following command:

```
SELECT * FROM defects
```

This command will display (*select*) everything (all columns, indicated by the *) from the table, *defects*. However, in many cases this is far more information than you are interested in seeing. To restrict the output to certain records, you can indicate a condition. For example, if you only want to see a summary of the defects associated with the project *QTMS*, you could enter:

```
SELECT identifier, headline FROM defects
WHERE project = 'QTMS' ;
```

This command displays the *identifier* and *headline* columns from the *defects* table for the defects linked to the project *QTMS* (the column *project* has the value *QTMS*). Note that the strings you search for are always enclosed in single quotation marks. The order you list the columns in is the order in which they are displayed. You can continue combining conditions or restricting the output until you get exactly the result you want. For example:

```
SELECT identifier, headline, severity
FROM defects
WHERE status = 'A'
AND severity < 3
```

Note: Be sure to always use a comma to separate column names in the SELECT statement. If you forget the comma, your query may produce unexpected results.

In addition, it is possible to control the order in which the results of a query are displayed. For example, to organize the results of a query so that it is displayed alphabetically by engineer's name:

```
SELECT identifier, severity, engineer FROM defects
WHERE status = 'A'
ORDER BY engineer
```

Using dates with the Oracle database

When using an Oracle database, if the query from *ddtssql* is for dates, the date format used depends on the date range being searched. For a date range contained within the current century, use the YYMMDD format. For dates outside the current century, use the RRMMDD format where RR stands for years in any other century. For example, to see defects with an estimated fix date greater than January 1, 2000 (and today is in 1999 or earlier):

```
SELECT to_char (est_fix_date, 'rrmmdd')
FROM defects
WHERE est_fix_date > to_date ('000101','rrmmdd')
;
```

Note: For more information on Oracle format models see your Oracle documentation.

Formatting query output

The format for a simple query for list of the engineers assigned to each defect in the *defects* table would look as follows:

```
1> SELECT identifier, engineer
2> FROM defects
3> go

identifier    engineer
------------  ---------
QTKqa00001    fred
QTKqa00002    fred
QTKqa00032    jones
QTKqa00034    jones
QTKqa00062    fred
(5 rows selected)
```

Notice the extra space between the ID and the engineer fields. By default, *ddtssql* pads each field with spaces so that it is the width defined in the database schema (*schema_file*). The *format* command can be used to override this behavior. The *format* command allows you to specify a *printf*-style format to control how the output is formatted. See the *printf(3)* man page for a complete description.

For example, to execute the previous query with the first column left justified with a maximum width of 10 characters and with the second column allowed to extend as wide as needed:

```
1> format %-10.10s %s
2> go
identifier engineer
---------- ---------
QTKqa00001 fred
QTKqa00002 fred
QTKqa00032 jones
QTKqa00034 jones
QTKqa00062 fred

(5 rows selected)
```

Note that the formatting instructions are applied to all subsequent queries and that only the columns that have a format will be displayed in the output. For example, if you add the *severity* column to the above query, the query would still produce the same results.

Note: You can use either *vi* or *emacs* to edit queries in *ddtssql*.

Using SQL in a Script

Another way you can use *ddtssql* is within a shell script to retrieve information from your database. The following is a simple example of how you could use *ddtssql* in a shell script to return the assigned owner of a defect record.

```
#!/bin/sh
# Lookup the owner of <defect id>.
#
# Usage: owner <defect id>
#
defectid=$1
engineer=`ddtssql -f - -noheader <<_E_O_F
      format %s
      select
          engineer
      from
          defects
      where
          identifier = '$defectid'
      ;
_E_O_F`
echo "$engineer"
```

To quit *ddtssql*, enter *quit* or *exit* at the line prompt. Now that you are somewhat familiar with the *ddtssql* interface and some of its features, you are ready to practice writing queries.

Retrieving Information from Multiple Tables

So far we have looked at queries that only collect information from one table. Getting information from multiple tables is very similar, except you have to identify which table each column belongs to (for example, in the WHERE clause). To do this, you add the table name before the column name like this:

```
defects.engineer
```

Table name   Column

For example:

```
SELECT defects.identifier,defects.project,
       enclosures.name
FROM defects, enclosures
WHERE defects.identifier = enclosures.identifier
WHERE defects.severity = 1
```

This query displays the defect id, associated project, and related file for all defects with a severity level of 1.

Now that you have a general understanding of how to use the SELECT statement to retrieve information from your database, you are ready to begin writing more advanced queries.

ClearDDTS and standard SQL

Every implementation of SQL offers slightly different capabilities. This section describes the unique features of the ClearDDTS database and summarizes the supported and unsupported SQL commands available.

Date conversion

By default, dates are displayed in the YYMMDD format in ClearDDTS. However, the database allows you to control the format of a date field using the *date_convert* function. This function takes the field name and formatting instructions in ANSI C *strftime()* format as follows:

```
date_convert(field, format)
```

For example, to have the submitted-on date in month/day/year (such as 11/21/97) format, you would use:

```
date_convert(submitted_on, '%m/%d/%y')
```

For a complete list of formatting descriptors, see the UNIX man page for *strftime*.

Aggregate comparisons

One of the unique features of the ClearDDTS database is the ability to perform aggregate comparisons. This capability allows you to easily extrapolate information based on the data in your database in a variety of ways. As a simple example, assume you want to see the number of assigned and resolved defects for engineer *jones*. In standard SQL, this would require two separate queries, but by using an aggregate comparison, you could have a query like this:

```
SELECT assigned=COUNT(status = 'A'),
       resolved=COUNT(status = 'R')
FROM defects
WHERE engineer = 'jones'
```

This query would return something like this:

```
assigned  resolved
--------  --------
30        18
```

By combining aggregate comparisons, you can perform multiple tasks and calculations using a single SELECT statement. For example, the following query illustrates how you can calculate the percentage of resolved defects per engineer:

```
FORMAT %-10.10s %-10.10s %-8.8s %-4.4s %-8.8s %-5.5s %-4.4s
SELECT project, engineer,
       Assigned = COUNT(status='A'),
       Open = COUNT(status='O'),
       Resolved = COUNT(status='R'),
       Total = COUNT(status),
       Pct = Resolved * 100/Total
FROM defects
WHERE project = 'X-Graph'
GROUP BY project, engineer
ORDER BY Total desc ;
```

project	engineer	Assigned	Open	Resolved	Total	Pct
X-Graph	saxon	70	3	183	274	66.7
X-Graph	mcannon	5	0	174	205	84.8
X-Graph	joeb	2	0	151	162	93.2
X-Graph	connell	30	2	67	143	46.8
X-Graph	NULL	0	0	0	91	0
X-Graph	rico	0	0	44	81	54.3
X-Graph	stuart	17	2	37	59	62.7
X-Graph	rex	8	0	2	10	20
X-Graph	patd	0	0	0	8	0
X-Graph	nwong	6	0	0	6	0
X-Graph	mjm	0	0	5	5	100
X-Graph	mgr	0	0	2	4	50
X-Graph	chris	0	0	1	2	50
X-Graph	djg	0	0	1	1	100

```
(14 rows selected)
```

Table and column aliases

ClearDDTS allows you to use aliases for table and column names in your queries. If you are selecting numerous fields or writing complex queries, this feature can simplify the task of entering the query.

To create an alias (or *correlation name)* for a table, enter the alias
after the table name in the FROM clause. You can then use the
alias when selecting columns. For example:

```
SELECT status,
       last_mod,
       name
FROM defects D, enclosures E
WHERE D.identifier = E.identifier
AND D.last_mod > 950101
```

In this query, *D* is used as an alias for the *defects* table and *E* is
used as an alias for the *enclosures* table. You can also use a table
alias when specifying a column before actually defining the table
alias. For example:

```
SELECT D.identifier,
       D.last_mod,
       E.name
FROM defects D, enclosures E
WHERE D.identifier = E.identifier
AND D.last_mod > 950101
```

By default, ClearDDTS supplies column headings for your query
results using the column names entered in the query. However,
you can change these column headings to make your results more
readable by using column aliases. For example

```
SELECT DefectID = identifier,     /* column alias */
       os_version Platform,       /* column alias */
       when_found Phase_detected, /* column alias */
       status, severity
FROM defects
WHERE project = 'Demo'
AND submitted_on > 970101 ;
```

This query would provide a result similar to the following:

```
DefectID     Platform     Phase_detected   st se
------------ -----------  ---------------- -- --
QTKqa04315   SunOS4.1.3   beta test        A  1
QTKqa05033   HP UX 9.05   post-release     A  2
QTKqa04318   Sun 4.1.3    post-release     A  3
QTKqa04263   Sun 4.1.3    post-release     A  3
QTKqa04299   Sun 4.1.2    post-release     A  3
QTKqa04342   AIX 1.4      beta test        A  2
QTKqa04343   AIX 3.2.5    alpha test       A  3
QTKqa04246   Sol 2.x      beta test        A  3
QTKqa04248   HP-UX 9.0    beta test        A  3
QTKqa04268                integration      A  3
QTKqa04331   ALL          post-release     N  2
QTKqa04330   AIX 3.2.5    functional test  N  3
QTKqa04344   AIX          beta test        N  3
```

```
QTKqa04313   4.1.2       functional test  R  1
QTKqa04307   4.1.3       post-release     R  1
QTKqa04308   4.1.3       post-release     R  1
QTKqa04317               alpha test       R  1
QTKqa04327   Solaris     post-release     R  2

(18 rows selected)
```

As shown in this example, ClearDDTS supports both the
<alias> = *<column>* syntax and the *<column>* *<alias>* syntax.

Supported SQL statements

The following table lists the standard SQL statements supported
by the ClearDDTS database.

Warning: The INSERT, UPDATE, and DELETE statements are
supported, but they should **never** be used directly. ClearDDTS
user interfaces use these statements in conjunction with code to
update the data in the *allbugs* directory and the SQL database in
such a way as to keep the information synchronized. If you use
these statements to modify the data in the SQL database directly,
your changes will be overwritten by the data in the *allbugs* file the
next time anyone updates the defect using a standard ClearDDTS
interface (such as *webddts* or *xddts*).

Statement	Notes
Select	Provides all of the standard functionality associated with the SELECT statement plus the ability to perform aggregate comparisons.
Insert	Can only be used with one table at a time (*joins* are not allowed). See the warning above.
Update	Can only be used with one table at a time (*joins* are not allowed). See the warning above.
Delete	Can only be used with one table at a time (*joins* are not allowed). See the warning above.
Help	Displays schema information. Entering *help* displays table names, types, and number of fields in each table. To see field information for any table, enter *help <tablename>*.
Order By	Can only be used with fields included in the SELECT clause.
Group By	Can only be used with fields included in the SELECT clause.
Having	Can only be used in conjunction with GROUP BY.

Statement	Notes
Between	Allows you to specify a range of values.
Like, Not Like	Allows you to qualify your search using the wildcard characters % (for a string of character) and _ (for a single character). The value you are searching for needs to be enclosed in single quotation marks.
Null, Not Null	Null is used to indicate an empty field (the absence of any value).
And	Allows you to combine conditions in the WHERE clause and return the records that satisfy both (all) conditions.
Or	Allows you to combine conditions in the WHERE clause and return the records that satisfy either (at least one) condition.

Unsupported SQL statements

The following list summarizes the types of SQL statements that are *not supported* by the ClearDDTS database.

- ALTER statements (such as ALTER TABLE)

- CREATE statements (such as CREATE TABLE)

- DROP statements (such as DROP TABLE)

- Transaction processing statements (such as BEGIN TRANSACTION, COMMIT, ROLLBACK, etc.)

- Security statements (such as GRANT, REVOKE, ROLE); however, security is provided by only allowing the ClearDDTS administrator to manipulate (INSERT, UPDATE, DELETE) data.

- Cross-database queries and database owners.

- Subqueries (nested queries).

- Virtual tables (called Views).

- The logical operator NOT can only be used with Like (NOT LIKE) and Null (NOT NULL). The operators ANY, ALL, and EXISTS are not supported.

- Triggers and stored procedures cannot be used within queries.

Recommended reading

To learn more about SQL and how to write complex queries, you may want to consider the following resources:

- Bowman, Judith S., Sandra L. Emerson, and Marcy Darnovsky. *The Practical SQL Handbook*. Reading: Addison-Wesley, 1993.

- Trimble, J. Harvey Jr., and David Chappell. *A Visual Introduction to SQL*. New York: John Wiley & Sons, 1989.

- Date, C. J., *Database: A Primer*. Reading: Addison-Wesley, 1990.

15

Creating a Change Management System

In many organizations, there is a need for an integrated approach to Configuration Management (CM) and Defect Tracking (DT). This integrated approach is necessary for effective change control and is often called *Change Management*.

This chapter describes the general problems, requirements, and levels of integration necessary to control and track changes, and the specific ways that ClearDDTS can help you implement a robust Change Management system.

Understanding Release/Configuration Problems

Before considering the solution, it is best to understand why you may want to integrate configuration management and defect tracking. For the most part, every Release Engineering Manager would like to be able to answer the following questions:

- When an engineer checks in a file, on behalf of which defect (Engineering Change Order, ECO) was this change made?

- What other files were changed on behalf of the defect or ECO?

- For this release/configuration, what actual defects were repaired?

- For this release/configuration, what enhancements were made?

- What defects remain in this configuration/release?

- For any release/configuration, how can I produce an automatic set of release notes, including a table of contents, that summarizes what defects were repaired and enhancements made?

- Was the defect approved for repair when the file was checked out?

- Was the defect approved for repair when the file was checked in?
- Was the defect approved when the changed file was made part of the new release?
- How do I merge management-defined processes for CM and DT that include approval steps?
- How do I merge management-defined access control policies for CM and DT?
- Both CM and DT have access control and process policies. How do I add my own unique needs to this process?

None of the questions above can be answered by a standalone CM or DT tracking system. To address this problem, ClearDDTS is designed to work with your current CM system to provide an integrated change management solution.

Providing an Integrated Solution

Answering the questions posed above requires a robust CM/DT integration. The resulting software, called the Change Management Control System (CMCS), is similar to what hardware engineers and manufacturing departments have used for years to control changes that go into manufacturing (the Engineering Change Order, or ECO, system).

In manufacturing industries, Engineering Change Orders are used to control and track hardware changes that need to move from engineering into production. The ECO process includes approvals, schematic drawing or blue print check out/in, process controls, defined release dates, and predictability in the hardware release process.

In the software industry, a similar process can be used where the DT system records change orders and the CM system provides a repository and defined configurations where the software (hardware drawings and blue prints) can be checked in and out.

This software ECO system requires a CM and DT system to be integrated at three distinct levels:

- Version Control Level
- Configuration Level
- Process Control Level

Version Control Integration

Effective version control integration can help answer the following questions:

- When an engineer checks in a file, on behalf of which defect (Engineering Change Order, ECO) was this change made?
- What other files were changed on behalf of the defect or ECO?

Version control allows you to make changes to a file (or versioned object) and still be able to recreate any prior version of the file. There are several freely available version control systems in the UNIX environment. The most common version control system is the Source Code Control System, SCCS. Other freely available, often used systems include the Revision Control System, RCS, and the Concurrent Version System, CVS.

ClearDDTS provides version control integration with RCS and SCCS by keeping pointers to objects that these systems manage. The version control system is configured to remember a defect identifier at check-in time and ClearDDTS remembers all the files that were checked in and out on behalf of a particular defect. This is illustrated in Figure 15-1.

Figure 15-1: Version Integration
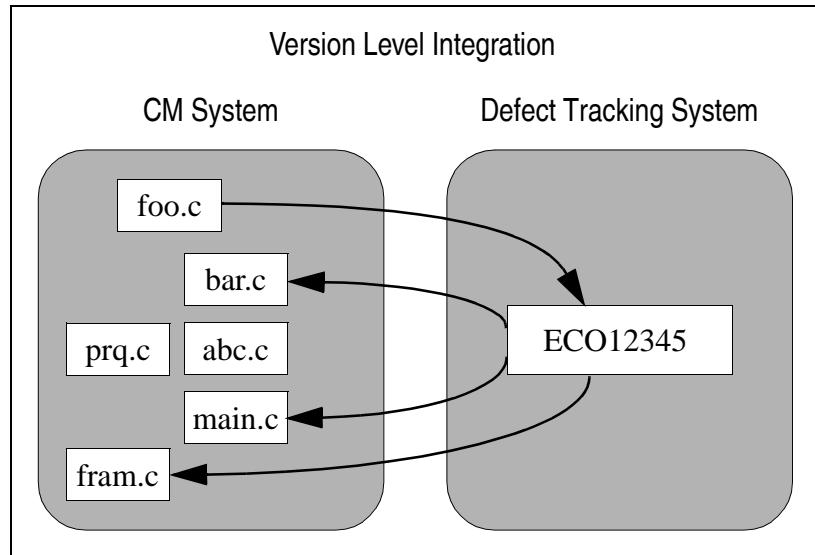
In this figure, file *foo.c* was checked out on behalf of defect identifier ECO12345; files *main.c, bar.c,* and *fram.c* were checked in on behalf of defect identifier ECO12345. The version control system saves the defect id in the version database and the defect tracking system saves all the file names checked in/out in the defect record called ECO12345.

If a user wants to back out a change to *foo.c,* he can follow the CM link through the defect tracking system and discover that the change must also be backed out of *main.c , bar.c*, and *fram.c*. That is, a user may look at a versioned file and then go look at the associated defect record. That record will tell the user who changed the file, who approved the change, why it was changed, when it was changed *and* the other three files that were changed to repair the defect (to implement the ECO).

This integration between version control and ClearDDTS can also make reading check-in summaries considerably more meaningful. For example, instead of cryptic comments like "fixed bug" or "fixed

the pointer problem," ClearDDTS can provide the one line bug description at file check-in time.

Note: Although version control systems can be very useful, they have little or no understanding of the collection of files that constitutes a configuration or product release. This is the primary difference between a version control system and a configuration management system.

Configuration Integration

Effective configuration integration attempts to answer the following questions:

- For this release/configuration, what actual defects were repaired?
- For this release/configuration, what enhancements were made?
- What defects remain in this configuration/release?
- For any release/configuration, how can I produce an automatic set of release notes, including a table of contents, that summarizes what defects were repaired, what enhancements were made, and what defects remain?

Unlike a version control system which knows about individual files, but not about the collections of files that constitute a configuration or a release, integrating at the configuration level requires that the CM database be able to produce the list of defect IDs associated with a CM query about a "configuration" or a "release." The CM system already knows all the files within a release and can print out a report of these files. This facility simply needs to be extended to report the defects IDs associated with those files. In general, this is a trivial extension to the CM query facilities, and some CM systems such as ClearCase already have this capability.

Once your CM query facility can report defect IDs (as opposed to file names), those defects can be passed to the defect tracking system and you can easily answer questions posed above. This is shown graphically in Figure 15-2.

Figure 15-2: Configuration Integration
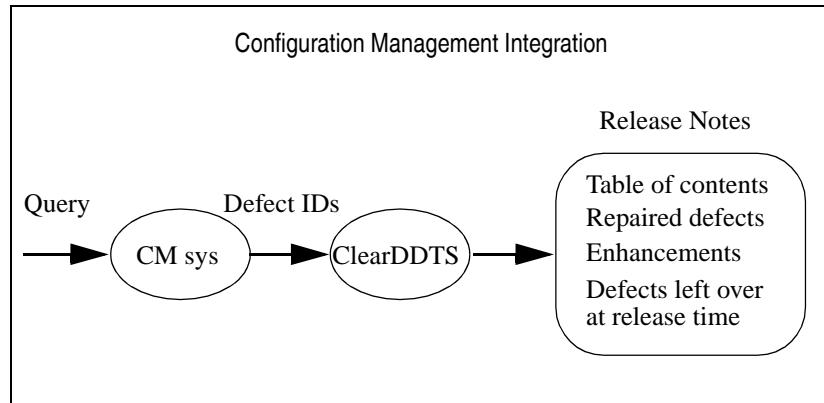
ClearDDTS provides this level of integration and for many CM systems can automatically produce release notes. Given any query that produces a list of defect IDs, ClearDDTS can easily sort the list into enhancement requests, actual defects, remaining unresolved defects, and a summary table of contents. This is done with the *cm.relnotes.sh(1)* utility.

Process Integration

Process integration requires merging the access control and process control policies of:

- Configuration management system
- Defect tracking system
- Release engineering manager (the user)

This type of integration is often the most difficult because access and process control policies are often a point of contention between CM, DT, and Release Engineering (User). The CM and DT vendor both want to control this process and thereby add value. The user also has unique requirements that need to be met. However, effective process integration will answer the remaining questions posed in the first part of this chapter.

The solution to this problem is an architected exit from the CM system at check in/out time. This exit must be allowed to return a flag that prevents the actual check in/out and issues an error message.

Generally this is not technically difficult. Many CM vendors already supply an architected "trigger" that can be executed before or after a check in/out. Providing such an exit at check in/out time allows the DT system to query the defect tracking database and find out if the defect is in the correct state.

This process is illustrated in Figure 15-3. In this example, the top portion shows a typical CM maintenance process. The user is assigned a defect for repair and wishes to check out a file, go through the normal edit-compile-debug process, and check in the repaired file. A merged CM/ClearDDTS system is shown in the lower portion of the figure.

Figure 15-3: Process Integration

Typical Process

Integrated Process
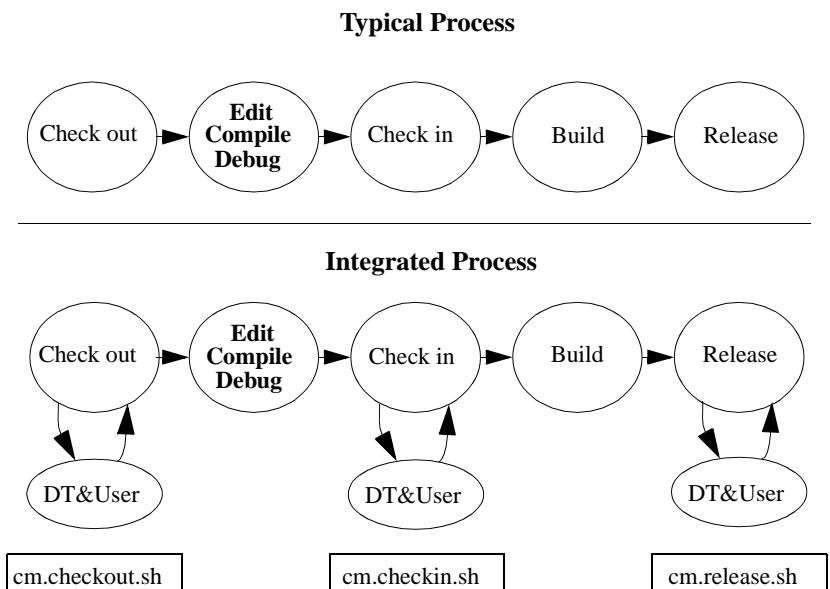
A merged CM/ClearDDTS system provides several exits from the CM system:

- At check out time (using *cm.checkout.sh(1)*), the exit calls ClearDDTS to make sure that the defect is approved for repair and that the engineer checking out the file is the engineer assigned to repair the defect. This exit also allows the user to add his/her process.
- At check in time (using *cm.checkin.sh(1)*), another exit is called to make sure the defect has been moved to the "Resolved" state and that the defect is approved for inclusion in the next release.
- An exit may also be configured at release time (using *cm.release.sh(1)*), for example, to make sure that all defects associated with the release are in the "Resolved," "Verified," or "Integrated" state.

Other examples abound. There may be a requirement that a software change be approved by a Change Control Board. Typically such approvals are recorded in a DT (ECO) system or user database. Again, an architected CM exit allows the Change Control Board, DT system, or user to automate and merge their own unique process requirements. The real issue here is that by passing control to all three parties (CM, DT, and the user), the access and process control policies are merged into a cohesive Change Control process.

ClearDDTS provides special exit utilities for the checkin, checkout, and release build processes. These scripts need to be tailored for your particular needs and are discussed briefly in this chapter. See the *cm.checkout.sh(1)*, *cm.checkin.sh(1)*, and *cm.release.sh(1)*, man pages for more information.

Setting Up ClearDDTS for Change Management

The proposed integrated Change Management system (using ClearDDTS with CM systems) described in the previous section can provide the Release Engineering team with the same benefits of control and predictability that the hardware ECO process has provided manufacturing and hardware engineering for many years.

To implement this system, you need to understand how to set up ClearDDTS to work with your Configuration Management (CM) system and how to customize the system for your particular needs.

One advantage of the ClearDDTS change management control system is that it is implemented using template files and Bourne shell scripts that can be easily customized to meet your unique needs. Before looking at these templates and shell scripts, let's walk through an example that illustrates the overall integration (see Figure 15-4).

Figure 15-4: Overview of the Integrated Change Management Control System
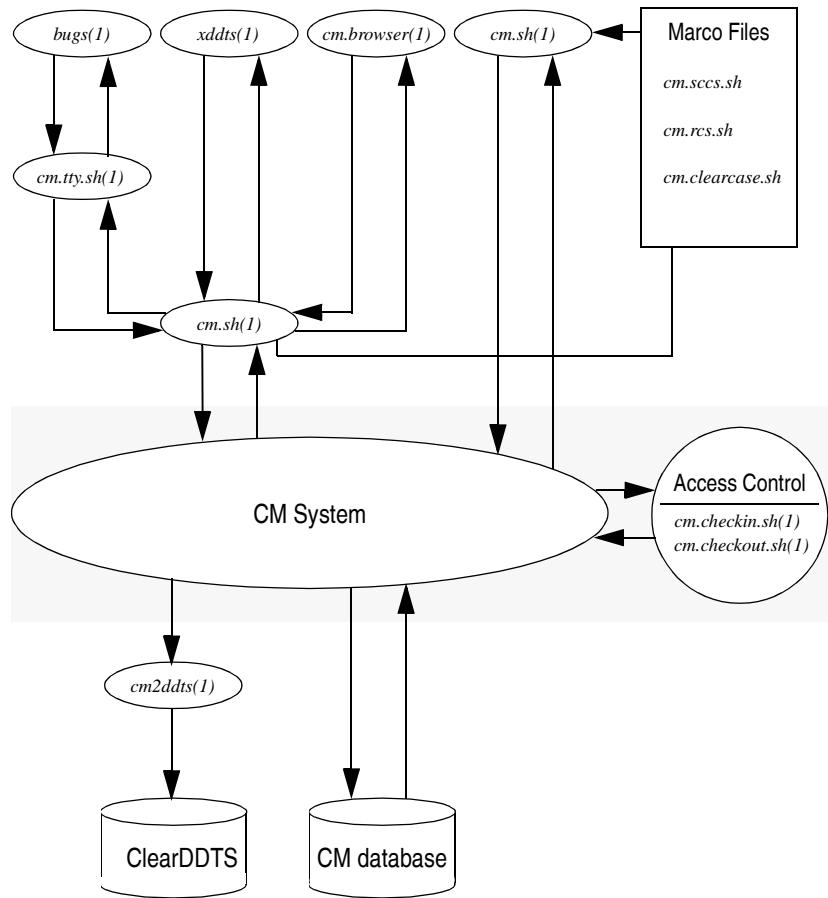
In ClearDDTS, there are four ways to call your CM system: through *xddts(1)*, *bugs(1)*, *cm.browser(1)*, or *cm.sh(1)*. For example, *xddts(1)* has a drop down menu called **CM**. From this menu, the user can specify a CM operation such as checkin or checkout. When an operation is selected, a file selection browser pops up and allows the user to select a file or files to perform the action upon. When both the operation and the files have been specified, *xddts* calls *cm.sh* to perform the CM action on those files.

The *cm.sh* utility determines the CM system to use by looking at a bugid that is passed into the utility. Based on the bugid, *cm.sh* finds the associated ClearDDTS project and the CM system to use for this project (CMsys field of *proj.control* file).

Once the CM system is determined, *cm.sh* uses a macro file to load in the commands appropriate for that CM system to perform the desired action. Current macro files appear in the form *cm.xxx.sh(1)*; for example, *cm.rcs.sh(1)*. You should look at these macro files now. They contain lines similar to the following:

```
CKIN='cd $path; cmsetuser $dirname sccs delget -y"$bugid by
$user, \ $Headline" $basefile'
```

This line creates a macro called CKIN that describes how to checkin a file. Once the macro is loaded, the appropriate command line is executed to check in or check out the specified files. If the CM action is successful, the CM system must then update its own internal database to reflect the CM operation and call the *cm2ddts(1)* using a post check in/check out trigger. The cm2ddts utility then passes control to some lower level programs to update the ClearDDTS database.

Providing Access Control

The integration between ClearDDTS and your CM system allows you to merge the access control policies of the CM system, the defect tracker, and the user using an access control trigger from the CM system. This is shown in the right side of Figure 15-4. Before every checkin, a utility called *cm.checkin.sh(1)* is called and before every checkout a utility called *cm.checkout.sh(1)* is called. If these utilities return a '0' exit code, the checkin or checkout is allowed to occur. A non '0' exit code causes the checkin or checkout to abort.

This ability to impose an external user-defined (or Change Management defined) access control policy is extremely powerful. For example, at checkout time, *cm.checkout.sh(1)* can make sure that the defect has been approved for repair and that the engineer

checking out the file is the engineer assigned to repair the defect. At checkin time, *cm.checkin.sh(1)* can make sure the defect has been moved to the Resolved state and that the defect is approved for inclusion in the next release.

You can completely customize the access control and authorization policies to suit your environment by editing these scripts.

Installing CMCS

To install the ClearDDTS Change Management Control system, all you need to do is define the CM system to use for every project. If you have already defined projects, use the *adminbug mprj* command to add the CM system. For example:

```
What Configuration Management System does this project use?  sccs
```

To see a list of valid responses, type a question mark (?).

If you are using Hewlett-Packard's SoftBench, you also need to run the *installsb.sh* utility (see the man page for more information). This is all that is required.

A Closer Look at CM Scripts and Utilities

There are several Bourne shell scripts that support the integration between ClearDDTS and your CM system. ClearDDTS also provides some special utilities to check in and check out files and control the release process. The next sections describe these scripts and utilities.

The cm.tty.sh Script

The *cm.tty.sh(1)* script is called by *bugs(1)* as:

```
cm.tty.sh  bugid
```

The *cm.tty.sh(1)* script is interactive and is used by the tty user interface to enable file selection and CM processing. The *cm.tty.sh* script checks out/in files requested from a user menu and then

calls the *cm.sh* utility to perform the CM action. See the *cm.tty.sh* and *cm.sh* man pages for more information.

CM Macro Files

Each CM macro file defines command line macros for checkin, checkout, and other configuration management tasks for a particular CM system. By editing this one file, you can quickly integrate an entire new CM system. (Typically, it takes 30 minutes to write the macros and 30 minutes to test and debug the system.)

For example, the SCCS macro file is located in *~ddts/bin/cm.sccs.sh*. Similarly, there is a *cm.rcs.sh* file for RCS and a *cm.clearcase.sh* file for ClearCase. Each of these files completely defines how ClearDDTS will invoke the appropriate CM system.

The cmsetuser Utility

The *cm.sccs.sh* file calls a utility, *cmsetuser(1)*. This utility must be run *suid root*. If the utility is run *suid root*, it allows you to invoke the CM utilities as any user that you specify (subject to many security checks). The utility is delivered in source form in the *~ddts/etc/cmsetuser.c* file for your reference.

Most CM systems do not need this functionality; however, if your versioned files are owned by a specific user id, you may find this utility extremely useful for integrating your unique CM needs into ClearDDTS. See the man page *cmsetuser(1)* for more information.

Convenience Shell Scripts

Most CM systems provide command line functions to check in and check out files. For example, RCS supplies *co(1)* to check out a file and *ci(1)* to check in a file. ClearDDTS provides a set of convenience functions to all supported CM systems. These utilities are:

- `cm.ci`   Check in
- `cm.co`   Check out

- `cm.uci`   Uncheck in
- `cm.uco`   Uncheck out
- `cm.ui`   Version initialization

These are all links to the *cm.sh* utility. The advantage of this utility is that it works for all supported CM systems. Another advantage is that they all require a bugid to be specified. Lastly, the script provides a centralized location for your own customizatio*ns.*

CM Access Control Process

One extremely useful feature of ClearDDTS' CM integration is that it has the ability to merge the access control policies of the CM system, the bug tracker, and the user. This is done via the checkin and checkout access control scripts. The following scripts are provided:

- cm.checkin.sh   Check in
- cm.checkout.sh   Check out
- cm.release.sh   Check to see that files belong in the release

Check out access control is provided by *cm.checkout.sh(1)* . This script is called *before* a file is checked out. In this script, you may enforce your own CM check out policy. As shipped, the script has examples that show you how to require that the defect be in a particular state or that it be assigned to a particular engineer. See the *cm.checkpout.sh(1)* man page and the script itself for more information.

Check in access control is provided by the *cm.checkin.sh(1)* script. This script is called before a file is checked in. Like *cm.checkout.sh,* it provides the ability to impose a check in policy on the CM system. See the *cm.checkin.sh(1)* man page and the script itself for more information.

How ClearDDTS Supports Roles

Most defect tracking and CM systems do not have specific roles defined in their day-to-day usage. However, some large companies or government procurements define special roles and need role support for formal CM. DDTs can support this requirement with the *~ddts/class/<classname>/mprj3.tmpl* and *~ddts/class/<classname>/aprj3.tmpl* files. These files allow the ClearDDTS administrator to define formal roles and designate the users assigned to those roles.

Some examples of roles might be:

- member of the configuration board
- release manager
- customer liaison
- tech support coordinator
- quality assurance coordinator
- problem assessor
- integration coordinator

Each of these roles may be formal and have certain permissions in the release process. For example, if a user is a member of a Configuration Control Board, that user may be allowed to make certain state transitions or approve changes to the product.

ClearDDTS supports these roles as part of the project creation and modification process (thus on a per project basis). To activate Role support, you need to modify the *~ddts/class/<classname>/mprj3.tmpl* and *~ddts/class/<classname>/aprj3.tmpl* files associated with the Class where role support is to be used.

The *mprj3.tmpl* and *aprj3.tmpl* files have example code for role support as follows:

```
###################################################################
#
#                        R O L E   S U P P O R T
#
#  Add the Roles and the names of the users in those Roles as shown
#  in the two examples below.  You may then use the master.tmpl file
#  to see if the person is assigned to the role
#  defined.
#
#  For example a:
#
#  Foo:         .
#               .
#     whoami
#     if not oneof -d $~/projects/$Project/proj.control CM-board
#          bin/echo "You are not on the CM Control Board! >&2
#          abort
#     fi
#               .
#               .
# placed in the master.tmpl file will cause an error if a user
# attempts an action and is not a member of the CM board.
#.
###################################################################
#
#
#CM-board:  "\n\nThe following relates to CM ROLES.\n\n"
#           "\nList LOGIN names of users that are members of the\n"
#           "Configuration Control Board for this project.\n"
#           "%s"
#           help aprj12.hlp
#           if not null
#                goodusers
#           fi
#
#CM-authority: "\nList LOGIN names of users that may approve changes.\n"
#             "%s"
#             help aprj12.hlp
#             if not null
#                  goodusers
#             fi
#
###################################################################
```

The CM-board and CM-authority are the names of the "roles" and the "users" defined for those fields and are entered at project creation or project modification time. These roles and the users assigned to them may be queried in the *master.tmpl* file using the *oneof* command shown in the previous example. (See *template(5)* man page.)

There is no restriction on the number of roles or on the content of the role field. For example, the content of the role field might be a UNIX group instead of a UNIX login id. The *oneof* command will return true/false as to whether a user is a member of the role. (See also *oneof -d* in the *template(5)* man page.)

A

Contents of a Defect Record

Defect records are maintained in the ClearDDTS database and in regular ASCII files. The ASCII files are located in *~ddts/allbugs/*/**. This appendix describes the format and content of these files.

Sample file

Each defect report file consists of a list of fields in the form:

```
Fieldname: string-of-text
```

Defect records can also include enclosures in the form:

```
Related-file::<comment>:: <title>
 line 1 enclosure text
 line 2 enclosure text
```

Enclosure data always begins in column 2. Column 1 must always contain a blank space (created by pressing the SPACEBAR). A sample defect record file, *QTKqa00475*, is shown below:

```
Start: QTKqa00475
Class: software
Project: Gui
Software: xddts
Version: 3.1
Showstopper: N
Enclosure-count: 1
Headline: temporary files are not using an appropriate umask
How-found: in-house normal use
When-found: alpha test
Test-name:
Test-system:
OS-version:
Severity: 2
Impacts-project:
Need-fix-by:
Submitter-name: Dave Rico
Submitter-org: Rational
Submitter-phone: 241-5813
Notify-submitter: Y
Submitter-id: rico
Submitter-host: verite
Submitter-mail: rico
Enhancement: N
```

```
Submitted-on: 921201
New-on: 921201
Identifier: QTKqa00475
Submitter-path:
Status: R
Last-mod:  921202
History::::
 xddts   921201 180918 Submitted to Gui by rico@verite
 xddts   921201 181039 N -> A (jb) by rico
 xddts   921202 092549 Enclosure "Problem & Fix" added by jb
 xddts    921202 092740 A -> O by jb
 xddts    921202 092749 O -> R (source code) by jb
Transition-stamp: 723317269
Type: BUG COPY
Timestamp: 4
Updated-by: jb
Engineer: jb
Engineer-mail: jb
Assigned-on: 921201
Related-file::Added 921202 by jb:: Problem & Fix
 There are several problems addressed by this bug

 1. I was not calling addfile() properly when handing off my
     edit* file in ~ddts/tmp. I am now using the ADD FILE_TMP
     flag with addfile so that the file stays around with the
    original user's permissions until a Commit.

 2. I added code to temporarily chmod of any file being edited
     to have the original mode permissions 'or'ed with 066.

 The above corrections should correct the deficiencies found
 on 12/1/92.-jb
Problem: design
Recommend-change: source code
When-caused: alpha test
Analyze-hours: 1
Est-fix-hours: 1
Est-fix-date: 921202
Postponed-on:
Opened-on: 921202
Resolution: source code
Changed:
Fix-hours: 1
When-fixed: alpha test
Analyzed-by: jb
Resolver-name: John Backman
Resolver-id: jb
Resolver-host: verite
Resolved-on: 921202
End: QTKqa00475
```

To locate defect records with particular field values, you can use
query tools such as *ddtssql* and *findbug*. For example, to search the

database for open defects with a severity level of 1 that are assigned to the engineer *bill,* your SQL query might look like this:

```
SELECT identifier FROM defects
WHERE state = 'O'
AND severity = '1'
AND engineer = 'bill'
```

With *findbug*, you would use the following command:

```
findbug -O Engineer == bill && Severity == 1
```

Note: For performance reasons, you should use *ddtssql* to search the database. When you use *findbug*, it composes a query and then calls *ddtssql* to perform the search. Using *ddtssql* directly can be up to ten times faster than *findbug* for the same search.

Field descriptions

The following table lists each field in the software class and describes how it is used. These fields represent the fields in a typical defect record in the software class. If you have modified the software class master.tmpl file or you are using a different class, the fields in your defects will be different. For information on how these fields are derived view the master.tmpl file.

Field	Meaning
Start	Same as the Identifier field; appears as the first field in every record.
Analyzed-by	Engineer who analyzed the problem.
Analyze-hours	Hours it took to analyze the problem.
Assigned-on	Date defect was assigned to an Engineer.
Changed	Name of what was changed to resolve the problem (for example, this may be the name of a source file or a piece of documentation).
Children	The *bugIDs* of children linked to this defect.
Class	Class that this defect belongs to.
DDTs-mail-from	ClearDDTS machine name and return mail path to submitter.
DDTs-mail-to	ClearDDTS machine name and mail path to which the defect is being sent.

Field	Meaning
Duplicate-of	Identifier of the record this one is a duplicate of.
Duplicate-on	Date defect was declared a duplicate.
Enclosure-count	Number of file enclosures for this defect.
Engineer	Login ID of person assigned to analyze/fix the problem.
Engineer-mail	E-mail address to receive engineer notification mail.
Enhancement	Is this an enhancement request (Y/N)?
Est-fix-date	Analyzer's idea of when this problem will be resolved.
Fix-hours	Number of engineering hours expended to resolve problem.
Forwarded-from	Name of project from which record was forwarded.
Forwarded-on	Date this record was last forwarded.
Forwarded-to	Name of project to which record was forwarded.
Forwarder-host	Hostname of machine from which record was last forwarded.
Forwarder-id	Login ID of person who last forwarded this record.
Forwarder-name	Real name of person who last forwarded this record.
Forwarder-org	Organization of person who last forwarded this record.
Forwarder-phone	Phone number of person who last forwarded this record.
Headline	One-line description of the problem.
History	Audit trail of all changes to this defect.
How-found	Strategy or method used to detect the problem.
Identifier	Unique identifier for this record (bug ID).
Impacts-project	Name of a project whose schedule is blocked by problem.
Last-mod	Date this defect record was last modified.
Need-fix-by	Date when fix is required so schedule is not impacted.
New-on	Date defect record was made new.
Notify-submitter	Y/N—Used to indicate whether the submitter is notified when the defect is modified.
OS-version	Name/version of operating system on which problem found.
Other-mail	Additional list of users to whom notification mail is sent for all record modifications.

Field	Meaning
Opened-on	Date this record was last opened.
Parents	The *bugIDs* of parents linked to this defect.
Postponed-on	Date on which record was last postponed.
Problem	Analyzer's idea of what/where the problem is.
Project	Project to which this record belongs.
Recommend-change	Change recommended to repair the defect.
Related-file	Beginning of an enclosure of free-form data.
Resolution	Description of what was changed to correct the problem.
Resolver-id	Login ID of person who resolved the problem.
Resolver-host	Hostname of machine where the problem was resolved.
Resolver-name	Full name of resolver.
Resolved-on	Date on which the problem was resolved.
Severity	Severity of problem (by default, 1 - 5, 1 highest).
Showstopper	Is this bug severe and stopping others' progress (Y/N)?
Software	Name of the software module or program with the problem.
Status	State of this record (N, O, A, R, etc.).
Submitted-on	Date bug was submitted.
Submitter-host	Hostname of machine on which this record was submitted.
Submitter-id	Login ID of person who submitted this record.
Submitter-mail	E-mail address to receive submitter notification mail.
Submitter-name	Full name of submitter.
Submitter-org	Organization of person who submitted this record.
Submitter-path	Name and path from this ClearDDTS machine to submitter's machine.
Submitter-phone	Phone of submitter.
Test-name	Name of test program which detected the problem.
Test-system	Name of machine on which problem was detected.
Timestamp	Integer number of transactions that have occurred on this defect $(1, 2, 3...)$.
Transition-stamp	Date of last state transition.
Type	Used internally for bug routing.

Field	Meaning
Verifier-host	Hostname on which the problem fix was verified.
Verifier-id	Login ID of person who verified that the problem was fixed.
Verifier-name	Name of person who verified that the problem was fixed.
Verified-on	Date this problem fix was verified.
Verify-check	Fix has been verified (Y/N).
Version	Version of the software module or program with the problem.
View	Determines whether customer may view this defect (read access control).
When-caused	Software lifecycle phase in which the problem was introduced.
When-fixed	Software lifecycle phase in which the problem was corrected.
When-found	Software lifecycle phase in which the problem was detected.
End	Same as the Identifier field; appears as the last field in every record.

Fields required by ClearDDTS utilities

The following fields are used by *bugs*, *bugs.in*, *bugmail*, *net.out*, and various administrative utilities. These fields must be present in any ClearDDTS defect record:

Defect State	Fields
All defects require:	Start Class Enclosure-count Headline Identifier New-on Project Severity Status Submitter-id Submitted-on Submitter-path Timestamp Transition-stamp Type End
Assigned (A) defects:	Engineer Assigned-on
Open (O) defects:	Engineer Opened-on
Resolved (R) defects:	Resolved-on Resolver-id
Verified (V) defects:	verified-on Verifier-id

Fields with special significance

The following fields have special significance if they are present:

Field	Description
Forwarded-from	Name of project from which record was forwarded. Automatically updated when F transitions arrive.
Forwarded-to	Name of project to which record was forwarded. Automatically updated when F transitions arrive.
Related-file	Identifies enclosures to various ClearDDTS utilities.
Submitter-mail	Mail address of the person submitting the defect. If not empty, e-mail will be sent to this address whenever the defect is modified.
Engineer-mail	Mail address of the engineer who worked on the problem.
Other-mail	Other people who should be sent notification mail when the defect is modified.

B

Converting to ClearDDTS

This appendix describes how to convert an existing bug database to a ClearDDTS database. In general, this involves the following steps:

1   Identifying your projects and network configuration.

2   Creating projects and classes appropriate for your environment.

3   Creating ClearDDTS defect records (files) from your existing bug records.

4   Running the conversion utility *ddtsconvert* to move the defect records into ClearDDTS.

5   Running the *adminbug dbms* command to merge the defects into the ClearDDTS database.

These steps are described in greater detail in the following sections.

Ensuring a successful conversion

After helping many customers convert their existing system to ClearDDTS, we strongly recommend that you create an ASCII printout in your current bug report format and make this a defect enclosure. This procedure guarantees that no data is lost in the conversion and makes acceptance of the new system much easier for your users who will see how the old system maps to the new system for existing defects.

We also recommend that you first convert a few test defect reports for a test project. After moving these test defects into ClearDDTS, make sure you can manipulate them from within the ClearDDTS

user interface. When you are satisfied that everything is working properly, go ahead and convert all of your defect reports.

Note: You should convert all of the defects in your system, even those that have been resolved. By including all defects, you are able to access historical information about your projects and immediately generate useful management reports.

If you run into problems in your conversion, check the enclosure field. The most common formatting error is not placing a space in column 1 for enclosure data. Even blank lines in an enclosure must have a space in column 1.

Identifying your projects

In ClearDDTS, users always log defects against defined projects that exist on specific machines in the ClearDDTS network. Therefore, the first step in conversion is planning the projects and machines that you will be using. To do this, answer the following questions:

- What projects will be defined?
- Which machines will run ClearDDTS?
- Which projects will reside on which machine(s)?

Experience has shown that if these questions are not correctly answered up front, the conversion process becomes very difficult and will probably have to be done more than once. In addition, it is usually best to bring together future users of the system, explain how projects are used in the system, and ask these users how they would like to see projects organized.

Creating projects and classes

When you have answered the questions posed above, install ClearDDTS on the selected machines and use the *adminbug aprj* command to create the projects desired on those machines. You may also want to configure your classes to suit your environment.

See Chapter 8, Customizing ClearDDTS, for more information about configuring the system.

Creating ClearDDTS defect records

When you have finished creating projects, you are ready to do the data conversion. This conversion is relatively easy. For every defect in the existing database, you need to create a defect record file in the ClearDDTS format. It is easiest if you create a separate directory for each project and place the converted defects into individual files in that directory.

Format of a ClearDDTS record

As described in Appendix A, Contents of a Defect Record, ClearDDTS records contain a number of fields in the format:

```
Fieldname: string of text
```

Defect records can also include enclosures in the form:

```
Related-file::<comment>:: <enclosure name>
 line 1 of enclosure text
 line 2 of enclosure text
```

Enclosure data always begins in column 2. Column 1 must always be a blank space. Therefore, all enclosure contents are offset by one space.

The comment is used internally by ClearDDTS to enter historical data about transactions. Leave the comment blank during conversion. ClearDDTS will fill in the comment with the following information:

```
verb date by username
```

where date is in DDTS format (yymmdd). For example:

```
Added 980312 by ddts
```

You need to write a program that converts a bug from its existing format into a file in the ClearDDTS format. (If you are using some sort of relational database management system, its report writer should make this task easy.) Your conversion program is

responsible for producing all of the defect record fields except the following, which are supplied automatically by the conversion utility, *ddtsconvert*:

```
Start:
Type:
Identifier:
Timestamp:
Transition-stamp:
End:
```

Although your conversion program should create as many of the ClearDDTS fields as possible, these fields can be in any order and many of the fields do not need to be filled in. The required fields are described at the end of Appendix A (see "Fields required by ClearDDTS utilities"). Remember, you do not need to include the fields listed above. Also, include the *Submitter-path* field, but do not supply a value.

Note: The required fields in Appendix A must be present in any defect record. In the examples that follow, additional required fields are listed that reflect customizations made to the default *software* class.

Filling in some special fields

While many fields are optional it is highly desirable to have the following fields filled in:

Field	Value

```When-caused```
```When-fixed```
	For each of these fields, select one of the following:
	```investigation```
	```specification```
	```design```
	```implementation```
	```functional test```
	```integration```
	```installation```
	```alpha test```
	```beta test```
	```post-release```

Field	Value

`Resolution`	Select one of the following:
	`not a bug`
	`unreproducible`
	`design`
	`source code`
	`language tools`
	`configuration`
	`data file`
	`documentation`
	`process`

Assigning states to defects

The most difficult thing your conversion program needs to handle is deciding the appropriate ClearDDTS state to assign each defect (in the *Status* field). We suggest that you select one of the following:

Status Field	Description
Status: N	New bug not yet assigned to an engineer.
Status: A	Assigned to an engineer, but not yet opened.
Status: O	Opened by engineer who is now working on it.
Status: R	Resolved. The bug has been repaired or otherwise handled.

Below is an example of a bug that started in the N (New) state and was progressively moved to the A (Assigned), O (Open), and R (Resolved) states. In addition, a list of changes resembling *diff* output is included to show what fields were added and modified in moving from the old state to the new state. The old state fields are indicated by a <, and the new or changed fields are indicated by a >.

Here is the bug in the N (New) state:

```
Start: SFDaa00271
Status: N
Software: drawit(1)
Version: 2.0
Headline: drawit(1) causes wrap around if string has embedded newline.
Enhancement: N
How-found: interactive test
When-found: implementation
Test-name: draw
```

```
Test-system: moe
OS-version: 5.3
Showstopper: no
Impacts-project: graphics 2.1
Severity: 3
Need-fix-by: 881010
Project: DDT.test
Attention: mike
From: ddts
Submitter-name: ClearDDTS Administrator
Submitter-id: ddts
Submitter-host: mikey
Submitter-phone:
Submitter-org: SFD Software Lab
Submitted-on: 880918
Enclosure-count: 1
New-on: 880918
Identifier: SFDaa00271
Related-file::Added 890918 by ddts:: More info
 I tried to draw a line with drawit.c.  It failed every time I gave it a
 string with a new line. Try it. It fails every time.
Submitter-path: SFDaa mikey!ddts
Transition-stamp: 590628456
Type: BUG COPY
Timestamp: 1
Updated-by: ddts
End: SFDaa00271
```

Note: Enclosure data starts in column 2 not in column 1.

Here is the list of changes required to move a bug from the N (New) state to get to the A (Assigned) state.

```
< Status: N
< Timestamp: 1
---
> Status: A
> Timestamp: 2

> Engineer: rico
> Engineer-mail: rico
> Assigned-on: 880918
```

Here is the resulting defect report in the A (Assigned) state:

```
Start: SFDaa00271
Status: A
Software: drawit(1)
Version: 2.0
Headline: drawit(1) causes wrap around if string has enbedded newline.
Enhancement: N
How-found: interactive test
When-found: implementation
Test-name: draw
Test-system: moe
OS-version: 5.3
Showstopper: no
Impacts-project: graphics 2.1
```

```
Severity: 3
Need-fix-by: 881010
Project: DDT.test
Attention: mike
From: ddts
Submitter-name: ClearDDTS Administrator
Submitter-id: ddts
Submitter-host: mikey
Submitter-phone:
Submitter-org: SFD Software Lab
Submitted-on: 880918
Enclosure-count: 1
New-on: 880918
Identifier: SFDaa00271
Related-file::Added 890918 by ddts:: More info
 I tried to draw a line with drawit.c.  It failed every time I gave it a
 string with a new line. Try it. It fails every time.
Submitter-path: SFDaa mikey!ddts
Transition-stamp: 590628654
Type: BUG COPY
Timestamp: 2
Updated-by: ddts
Engineer: rico
Assigned-on: 880918
Engineer-mail: rico
End: SFDaa00271
```

Here is the list of changes required to move a bug from the A (Assigned) state to the O (Open) state.

```
< Status: A
< Timestamp: 2
---
> Status: O
> Timestamp: 3

> Problem: source code
> Resolution: source code
> When-caused: beta test
> Analyze-hours: 1
> Fix-hours: 1
> Est-fix-date: 881101
> Est-fix-hours: 1
> Opened-on: 880918
```

Here is the resulting defect report now in the O (Open) state:

```
Start: SFDaa00271
Status: O
Software: drawit(1)
Version: 2.0
Headline: drawit(1) causes wrap around if string has enbedded newline.
Enhancement: N
How-found: interactive test
When-found: implementation
Test-name: draw
Test-system: moe
OS-version: 5.3
Showstopper: no
```

```
Impacts-project: graphics 2.1
Severity: 3
Need-fix-by: 881010
Project: DDT.test
Attention: mike
From: ddts
Submitter-name: ClearDDTS Administrator
Submitter-id: ddts
Submitter-host: mikey
Submitter-phone:
Submitter-org: SFD Software Lab
Submitted-on: 880918
Enclosure-count: 1
New-on: 880918
Identifier: SFDaa00271
Related-file::Added 890918 by ddts:: More info
 I tried to draw a line with drawit.c.  It failed every time I gave it a
 string with a new line. Try it. It fails every time.
Submitter-path: SFDaa mikey!ddts
Transition-stamp: 590628654
Type: BUG COPY
Timestamp: 3
Updated-by: ddts
Engineer: rico
Assigned-on: 881101
Est-fix-hours: 1
Problem: source code
Resolution: source code
When-caused: beta test
Analyze-hours: 1
Fix-hours: 1
Est-fix-date: 881101
Opened-on: 880918
End: SFDaa00271
```

Here is the list of changes required to move a bug from the O (Open) state to the R (Resolved) state:

```
< Status: O
< Timestamp: 3
---
> Status: R
> Timestamp: 4

> Changed: drawit.c
> When-fixed: beta test
> Resolver-id: mmanley
> Resolved-on: 880918
> Resolution: source code
> Fix-hours: 1
> Analyzed-by: mmanley
> Resolver-name: Mike Manley
> Resolver-host: verite
```

The Engineer field is used to determine which bugs are the current responsibility of a particular engineer. This is used by *-u* option of *bugs* and the *-e* option of *findbug*.

Here is the resulting resolved defect report:

```
Start: SFDaa00271
Status: R
Software: drawit(1)
Version: 2.0
Headline: drawit(1) causes wrap around if string has enbedded newline.
Enhancement: Y
How-found: interactive test
When-found: implementation
Test-name: draw
Test-system: moe
OS-version: 5.3
Showstopper: no
Impacts-project: graphics 2.1
Severity: 3
Need-fix-by: 881010
Project: DDT.test
Attention: mike
From: ddts
Submitter-name: ClearDDTS Administrator
Submitter-id: ddts
Submitter-host: mikey
Submitter-phone:
Submitter-org: SFD Software Lab
Submitted-on: 880918
Enclosure-count: 1
New-on: 880918
Identifier: SFDaa00271
Related-file::Added 890918 by ddts:: More info
 I tried to draw a line with drawit.c.  It failed every time I gave it a
 string with a new line.  Try it.  It fails every time.
Submitter-path: SFDaa mikey!ddts
Transition-stamp: 590628895
Type: BUG COPY
Timestamp: 590628895
Updated-by: ddts
Problem: source code
Resolution: source code
When-caused: beta test
Analyze-hours: 1
Fix-hours: 1
Est-fix-date: 881101
Opened-on: 880918
Changed: drawit.c
When-fixed: beta test
Resolver-id: mmanley
Resolved-on: 880918
Resolution: source code
Analyzed-by: mmanley
Resolver-name: Mike Manley
Resolver-host: verite
End: SFDaa00271
```

As you can see, beyond the base information gathered when the bug was first submitted, little is added or changed when moving from state to state. Your conversion program should decide the current state of each bug and generate the appropriate fields and

values. Many fields are optional and the format is simple enough that your conversion program should be easy to write.

Note: Remember that you do not need to create the following fields, because *ddtsconvert* will create them for you:

```
Start:
Type:
Identifier:
Timestamp:
Transition-stamp:
End:
```

Running the conversion utility

When you have created all of the defect record files in the ClearDDTS format and placed them in the appropriate project directories, you need to make these files available to ClearDDTS. To do this:

▪ Move the defect record files to the machine where the ClearDDTS project has been created with the *aprj* command.

▪ Run the *ddtsconvert* utility. It takes care of allocating a bug ID and installing the defect into the ClearDDTS database.

From the shell, enter:

```
cd project_dir
ls | ddtsconvert -v
```

The *-v* option (verbose) causes *ddtsconvert* to describe the conversion as it progresses.

Warning: Do not use *"ddtsconvert -v *"* because the shell may not have enough memory to do the expansion if the directory contains a large number of defect files.

Incorporating defects into the database

ClearDDTS will format the defects, assign IDs, and import the defects into the ClearDDTS system. After running *ddtsconvert*, you need to run the *adminbug dbms* command to merge the defects into the ClearDDTS database. See Chapter 2, Using Administration Utilities, for more information on using *adminbug*.

C

Sample Filter Command Script

This appendix provides a sample script to illustrate how you can use ClearDDTS template files and shell scripts to handle various tasks including default values and validation. This script tests whether a project is valid and is an example of a filter program as described in the *template* man page.

Script example

```
#!/bin/sh

#       This is an example of how you can use a shell script with the
#       template files. We used to use this shell script to ask for the
#       project name when a user submitted a bug. The script is no
#       longer used by the system but is preserved here as an example
#       of how to use template files and shell scripts.

#       The bugs(1) program places the current field value on the
#       standard input, so we can get what the user just typed in by
#       reading stdin. Alternatively, the field value could be an
#       invocation parameter. The expanded value is written to standard
#       out if there is a single one.

#       See if project name is in the invocation string if not, read
#       stdin to get the current field value.

#       This file is located in the ~ddts/bin directory.

if [ "$1" = "" ]
then
        line=`ngline`          #  Read stdin
else
        line=$1                # Pick up invocation parameter
fi

#
#  If they just hit return, tell them a response is required.
#

if [ "$line" = "" ]
then
        echo "Response is required." >&2
        exit 1                 #  Force template field to be executed again
fi
#
#  See if they asked for help
```

```
#

if [ "$line" = "\?" ]
then
      echo "The projects available are:" >&2
      projstat -bl >&2    # List projects they can log bugs against
      echo ""
      exit 1              #  Force template field to be executed again
fi

#
#  See if the project name is unique
#

list=`glob.projs $line'*'`
count=$?
if [ "$count" = 1 ]
then
      echo $list
      exit 0              # SUCCESS!  Go back to template file.
elif [ "$count" -gt 1 ]
then
    echo "Ambiguous:
$list" >&2                  # List projects that match
      echo ""
        exit 1              # Ambiguous, try it again
else
      echo "No such project, type '?' for a project list." >&2
      echo ""
      exit 1
fi
```

D E-mail Submission API

ClearDDTS provides users with the ability to mail defects to ClearDDTS for inclusion in the database. This mechanism is provided with the *e-mail submission API*. Look at the *ddtsmailbug* shell script for an implementation of the e-mail API.

To e-mail a defect to your ClearDDTS system, the following fields must be in the message:

Field	Description
`Project:`	Project to which you want all initial e-mail defects assigned.
`Software:`	Short description of the problem module or feature.
`Version:`	Version of the software.
`Showstopper:`	Y/N if this is a showstopper bug.
`Enclosure-count:`	Number of enclosures (related files).
`Headline:`	One line description of the problem.
`How-found:`	How the defect was detected.
`When-found:`	When in the release cycle was the defect detected.
`Severity:`	Severity (1 - 5) of the problem.
`Submitter-name:`	Customer's real name. If you don't know, put in something like Customer.
`Submitter-org:`	Submitter's company name. If you don't know, put in something like Customer.
`Submitter-phone:`	Customer's phone number. If unknown, put in "unknown."
`Submitter-id:`	Submitter's login ID.
`Submitter-mail:`	If this field is empty, no mail will ever be sent back to the submitter. If anything is in this field, mail will be sent to the submitter for each state transition of the defect.
`Enhancement:`	"Y" if this is an enhancement request; otherwise "N".
`Status:`	The value must be "S" to indicate this is a defect submission.

Field	Description
`History::::`	Optional, will initialize bug history. See example for format. ***Do not forget the space in column 1 of the next line.***
`Type:`	The value of this field must be "ALLOCATE" for a normal bug submission.
`Timestamp:`	The value of this field must be "1".
`Updated-by:`	Login ID of submitter (same as submitter-id).
`DDTs-mail-to:`	This field must look like: `XXXxx ddts@foo.com` where *XXXxx* is your ClearDDTS site identifier (you can run *ddtshostname* to get the name, if necessary) and *ddts@foo.com* is the Internet mail address to your ClearDDTS system.
`DDTs-mail-from:`	This field must look like: `YYYyy yyy@xyx.com` where *YYYyy* is the submitter's ClearDDTS site identifier and *yyy@xyz.com* is the Internet mail address for that machine. **Note:** Do not use the same value for YYYyy that you used for XXXxx above.

The e-mail processing makes the defect report look like it was submitted locally by setting up the following fields:

```
Submitter-host
Submitter-id
Submitted-on
Last-mod
Identifier
```

Here is an example defect submission:

```
Project:  ClearDDTS
Software:  foo.c
Version:  4.0
Showstopper:  N
Enclosure-count:  0
Headline:  The foo.c routine causes a core dump.
How-found:  in-house normal use
When-found:  alpha test
Severity:  3
Impacts-project:
Submitter-name:  ClearDDTS
Submitter-org:  Rational Software
Submitter-phone:  999-9999
Notify-submitter:  Y
Submitter-id:  ddts
Submitter-host:  vertigo
Submitter-mail:  ddts
```

```
Enhancement:  N
Submitted-on:  971203
New-on:  971203
Identifier:  QTKqa00479
Submitter-path:  QTKqa ddts@vertigo
Status:  N
Last-mod:  971203
History::::
 bugs 971203 082801 Submitted to ClearDDTs by ddts@vertigo
Transition-stamp:  723400081
Type:  BUG ORIGINAL
Timestamp:  1
Updated-by:  ddts
DDTs-mail-to:  QTKqa ddts@rational.com
DDTs-mail-from:  YYYyy yyy@yyy.com
```

If you want to mail ClearDDTS new defects and also mail ClearDDTS updates to those defects, see Appendix B for the new fields that must appear at each state. If you mail ClearDDTS an updated defect, the "Type" field must be "BUG ORIGINAL" and the "Timestamp" field must be incremented by 1. For example, if the current value of Timestamp is 5, the updated defect mailed to ClearDDTS must have a Timestamp of 6. You can also mail ClearDDTS information to be included in an enclosure to a defect. For details see *Sending mail to ClearDDTS* on page 11-14.

E

Creating Graphs with Graphbug

The *graphbug* program creates PostScript graphs from data formatted by the *tallybug* program or other report generation scripts. This appendix describes how to use the *graphbug* program and the format of the data used to generate the graphs.

Using Graphbug

Graphbug is a command line program which produces PostScript documents. Documents created with *graphbug* conform to version 2.0 of the Adobe Encapsulated PostScript Specification. Programs which can manipulate encapsulated PostScript will be able to use graphs produced by the *graphbug* program.

Command line options

The command line arguments to *graphbug* determine the overall location of graphs on the printed page. The following syntax is used:

```
graphbug [-slxw] [-ox,y] [-C] [-a] [-cconfig_file] [-flayout] [n of N]
```

The following table describes the command line options:

Option	Description
-slxw	Determines the size of the graphs on the page in inches. The values *l* and *w* are floating point numbers. The letter 'x' is used to separate the two parameters. For example, if you enter: `-s8.5x5.0` *graphbug* draws in an area that is 8.5 inches by 5 inches. The default is 10.5 inches by 8 inches in landscape mode.
-ox,y	Identifies the position of the lower-left corner of the graph area on the page when using the default graph orientation. The default is 0.25 inches from the left side of the page and 0.25 inches from the bottom.

Option	Description
-C	Produces color PostScript output which can be printed on a color PostScript printer such as the Tektronix Phaser 200i color printer.
-a	Produces black and white PostScript graphs using different fill patterns (rather than shades of grey) to represent the data.
-c*config_file*	Defines an alternate configuration file for *graphbug*.
-f*layout*	Defines the layout of graphs on a page. The layout contains three elements: - orientation - vertical dimension - horizontal dimension Orientation is represented by either L (landscape) or P (portrait). Numbers are used to indicate the number of graphs to be drawn vertically and horizontally on the page. For example: `-fP3x2` formats a page in portrait mode for 6 graphs. There will be up to 3 graphs vertically and 2 graphs horizontally.
n of N	Determines which graph on the page should be drawn with the current data. The value n is the number of the current graph, and N is the number of graphs to be drawn on the page. The value N cannot be more than the number of graphs defined by the *-f* option. The graphs are numbered left to right across each row from top to bottom. Graphs should be generated in order to ensure that *graphbug* will produce a complete page definition. The default is 1 of 1.

Description

The options *-s* and *-o* describe the area of the page covered by the graphs produced by the *graphbug* program. The *-f* option allows *graphbug* to combine several graphs on a single page.

To produce a page with more than one graph, *graphbug* can be executed once for each graph and the output concatenated into a single file. The parameters *n of N* identify the current graph when printing multiple graphs on a page.

The *graphbug* program acts as a filter, accepting input from its standard input and writing to its standard output. In most cases, *graphbug* will be used in a pipe where it will transform the data from its standard input into a PostScript definition.

Graphbug is able to print a variety of graphs including line, bar and filled area graphs. The contents of the graph are defined by the input description. Essentially, the command line arguments describe the placement of the graph while the input defines its contents.

Defining graph contents

Each graph description is composed of a header, a data section and an optional notes section.

- The lines of the *header* describe the type, title and dimensions of the graph.
- The *data* section follows the header section and defines a series of data sets which are interpreted according to parameters defined in the graph header.
- The *notes* section is optional and may follow the data section.

Upper-case keywords identify header elements and mark the beginning and end of the data and notes sections. For example:

```
TITLE "PROJECT XXX"
SUBTITLE "Problems xxx"
ADJACENT BAR "Origin" "Found" "Fixed"          Header Lines
.
.
.
DATA
"Concept/Definition": 27 5 15
"Requirements": 5 10 3
"Design": 10 15 10                              Data Lines
"Coding": 40 7 26
"Unit Test": 1 40 27
END DATA
NOTES
note lines                                      Notes Section (optional)
END NOTES
```

Graphbug incorporates a very simple lexical analyzer which recognizes four different data types.

strings	Strings are enclosed between double quote characters and cannot span lines. Strings cannot contain special characters such as double quotes, line feeds or carriage returns.
numbers	*Graphbug* recognizes and uses regular floating point numbers. The program does not support scientific notation.
dates	Dates are provided using the mm-dd-yy format where mm, dd and yy are integers which represent the corresponding month, day and the last two digits of the year.
time	Time is provided in the *hh:mm* format where *hh* and *mm* represent elapsed time in hours and minutes. Both hours and minutes must always be specified even if they are zero. Internally, *graphbug* represents time as the number of elapsed minutes.

Header section

Graph titles

The keywords "TITLE" and "SUBTITLE" identify the titles of the graph. The keywords must appear at the beginning of a line and are followed by a string. Titles always appear at the top of the graph. Both titles are required although either may be defined as an empty string:

```
TITLE "PROJECT XXX"
SUBTITLE "Problems - per product life cycle phases"
```

Vertical axis

The vertical scale provides the units of measure and the range of each data value in a graph. Values on the vertical axis may be measured as either linear values, logarithmic values or as elapsed time expressed in hours. The vertical axis may appear on either the left or the right side of the graph. Each major division of the vertical axis is labeled with a value. *Graphbug* can also draw grid lines behind the data corresponding to the major and minor

markings on the vertical scale. The vertical axis definition uses one of the following two formats:

```
VERTICAL side measure label format AUTOSCALE
VERTICAL side measure label format MINIMUM n MAXIMUM n MAJOR n MINOR n
```

`side`	One of the keyword phrases "LEFT", "RIGHT", "GRID LEFT" or "GRID RIGHT". The addition of the keyword "GRID" causes *graphbug* to draw background grid lines.
`measure`	One of the keywords "LINEAR", "LOG" or "HOURS".
`label`	A string for the vertical axis label is printed vertically beside the axis.
`format`	A *printf( )* format string which is used to print the scale values at each major scale mark. The scale values are always floating point numbers.
`MINIMUM n`	The lowest number on the vertical scale.
`MAXIMUM n`	The highest number on the vertical scale.
`MAJOR n`	The interval between each of the major scale marks for which the scale values will be printed.
`MINOR n`	The interval between minor scale marks which are marked with a short line. Generally, the minor scale interval should be an evenly divisible number of the major scale interval.
`AUTOSCALE`	The vertical axis is autoscaled.

For logarithmic scales, the *minimum*, *maximum*, *major* and *minor* values are expressed as exponents. The minimum and maximum values will define a vertical scale from 10min to 10max. The number of major divisions on the vertical scale is approximately the difference between the minimum and maximum exponent values divided by the major interval value. The number of minor divisions on the vertical scale is approximately the major value divided by the minor value.

When using the autoscale option, *graphbug* notes the minimum and maximum values in the input data and then determines appropriate values for the MINIMUM, MAXIMUM, MAJOR and MINOR parameters. Autoscaling is available for all vertical scale measures.

Horizontal axis

Graphbug supports a variety of horizontal axis types which correspond to the tags of the input data sets. The horizontal axis type determines the interpretation and representation of the data tags on the graph. The program does not check if data tags actually match the horizontal axis type, so mixed data tag types may produce unintended results. Data sets tagged with values outside the range defined by the horizontal axis definition are simply ignored.

Linear horizontal axis type

The Linear axis type maps floating point numbers within a predefined range. Data tags must be numbers and should be sorted from the lowest to the highest values. The declaration for this axis type is nearly identical to the declaration of the vertical axis type. The horizontal axis, however, does not support an autoscale option:

```
HORIZONTAL LINEAR label format MINIMUM n MAXIMUM n MAJOR n MINOR n
```

Date-based horizontal axis types

Graphbug supports data sets tagged with dates and provides four degrees of granularity ranging from days to years. It marks days individually on the horizontal axis and periods from weeks to years. Weeks always start on Sunday. Months and years always start on the first. *Graphbug* only labels full weeks, months, or years on the horizontal axis:

```
HORIZONTAL period label begin end
```

`period`	One of the keywords "DAY", "WEEK", "MONTH" or "YEAR".
`label`	A string which may be printed underneath the graph.
`begin`	The date of the earliest sample to include in the graph.
`end`	The date of the latest sample to include in the graph.

Enumerated horizontal axis types

Enumerated axis types map data sets tagged with either strings or dates. With the exception of the pareto graph, *graphbug* graphs

the data in the order in which it appears in the data section. Enumerated axis types do not provide a range of values so any number of data sets can be provided in the data section. There are two different enumerated axis types which differ only in the orientation of the graph labels. The basic enumerated axis prints the labels horizontally:

```
HORIZONTAL ENUMERATED label
```

`label`	A string which may be printed underneath the graph.

The sideways enumerated type prints the labels vertically and automatically enlarges the default bottom margin of the graph to provide more room for the labels:

```
HORIZONTAL SIDEWAYS ENUMERATED label
```

The sideways enumerated type also supports labels consisting of multiple lines of text. *Graphbug* will break sideways enumerated labels into lines where it encounters the sequence "\n" in the label string. The program does not perform word wrapping.

Graph types

Several keywords introduce the graph type declaration in the header. For all graphs except pie charts, the graph type definition defines the contents of the graph legend and the interpretation of the data values. The general form of the graph type declaration is:

```
graph_type legend_label ...
```

`graph_type`	One or more Keywords that define the type of graph.
`legend_label`	One or more labels which will appear in the legend to identify the data values.

Adjacent bar graphs

The adjacent bar graph presents data sets as groups of bars. Each bar represents a single data value and is shaded or colored to distinguish it from its neighbors. Each group of bars represents a data set and is separated from other groups by a gap. The keywords "ADJACENT BAR" declare an adjacent bar graph. Figure E.1 shows an adjacent bar graph which was created from the following graph description:

```
TITLE "PROJECT XXX"
SUBTITLE "Problems - per product life cycle phases"
ADJACENT BAR "Origin" "Found" "Fixed"
HORIZONTAL ENUMERATED ""
VERTICAL LEFT LINEAR "Bugs" "%0.0f" MINIMUM 0 MAXIMUM 50 MAJOR 10 MINOR 5
DATA
"Concept/Definition": 27 5 15
"Requirements": 5 10 3
"Design": 10 15 10
"Coding": 40 7 26
"Unit Test": 1 40 27
END DATA
```

Figure E-1: Adjacent Bar Graph Example
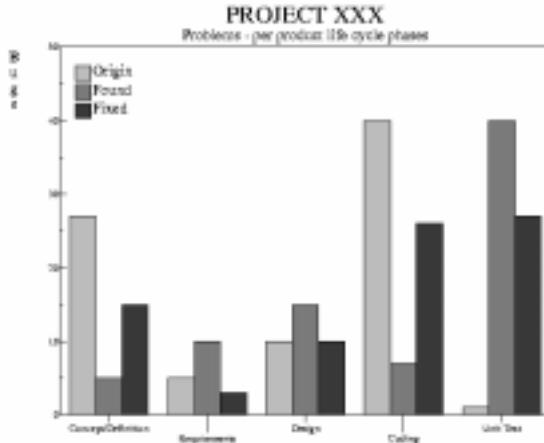
The vertical axis could also have been defined with the AUTOSCALE keyword to automatically scale the vertical axis.

The following graph description defines the same graph with the autoscale option:

```
TITLE "PROJECT XXX"
SUBTITLE "Problems - per product life cycle phases"
ADJACENT BAR "Origin" "Found" "Fixed"
HORIZONTAL ENUMERATED ""
VERTICAL LEFT LINEAR "Bugs" "%0.0f" AUTOSCALE
DATA
"Concept/Definition": 27 5 15
"Requirements": 5 10 3
"Design": 10 15 10
"Coding": 40 7 26
"Unit Test": 1 40 27
END DATA
```

Stacked bar graphs

The stacked bar graph presents data sets as groups of bar graphs which are stacked upon each other to form a single segmented bar. The bottom segments correspond to the first data value in each set and the total height of the bar stack is the sum of the data values. Each stack of bars is separated from other stacks by a gap which can be altered in the graph header. The keywords "STACKED BAR" declare a stacked bar graph. An example graph appears in Figure E.2.

Figure E-2: Stacked Bar Graph Example
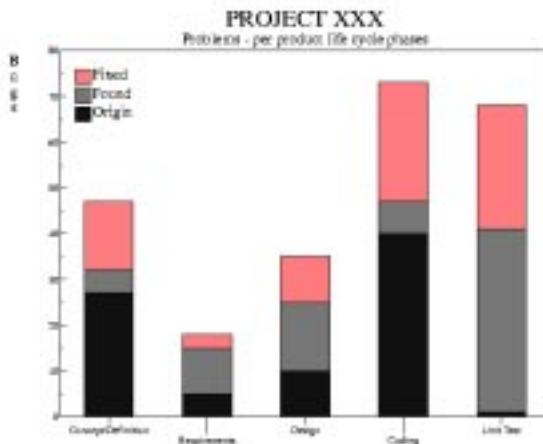
Dashed line graphs

Dashed line graphs present data as a number of line graphs. Each line is drawn using different dash patterns or with different colors when using the *-C* command line option. *Graphbug* uses the keywords "DASHED LINE" to declare a dashed line graph like the one shown in Figure E.3.

Figure E-3: Dashed Line Graph Example
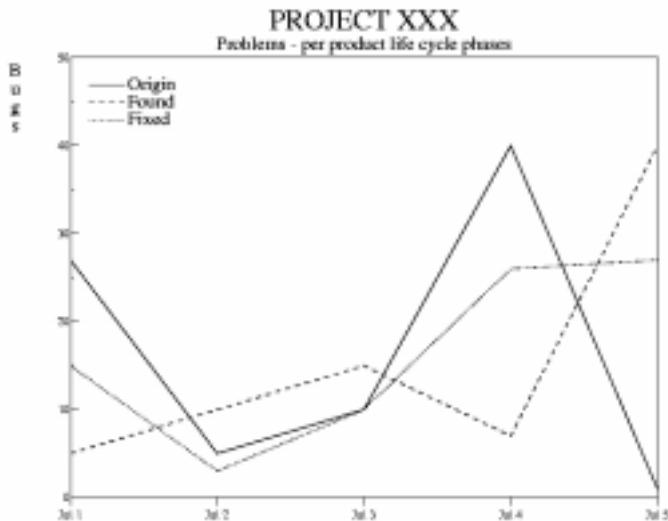
Marked line graphs

A marked line graph is nearly identical to the dashed line graph except that the lines are also marked with symbols at intervals along the lines. The keywords "MARKED LINE" declare a marked line graph.

Stacked line graphs

The stacked line or filled area graph presents data as a set of lines stacked on top of each other. The line for the first data value appears at the bottom of the stack and the top line represents the sum of all of the data values. The keywords "STACKED LINE" declare a stacked line graph like the example graph in Figure E.4.

Figure E-4: Stacked Line Graph Example
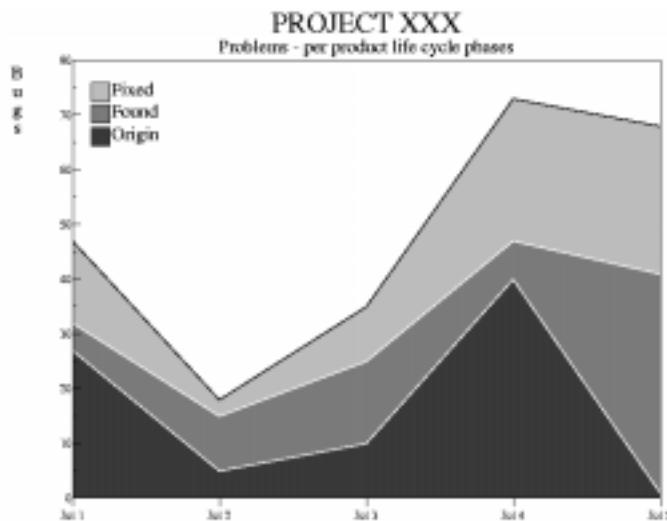
Pareto graphs

Pareto graphs are the combination of a stacked bar graph and a line graph. The stacked bar graph represents data values that are added within each set and then sorted left-to-right from highest to lowest sums. *Graphbug* calculates the total for all the sums which it then uses to create a line graph over the bars showing the percentage of the total sum that the current and preceding bars contribute to the overall total. The keyword "PARETO" introduces a pareto graph.

The pareto graph has two scales. The first scale is defined in the graph header and defines the heights of the bars. *Graphbug* adds a second scale for the line graph on the opposite side of the graph. The second scale is marked in percentages from 0% to 100%. Figure E.5 contains an example of a pareto graph.

Figure E-5: Pareto Graph Example
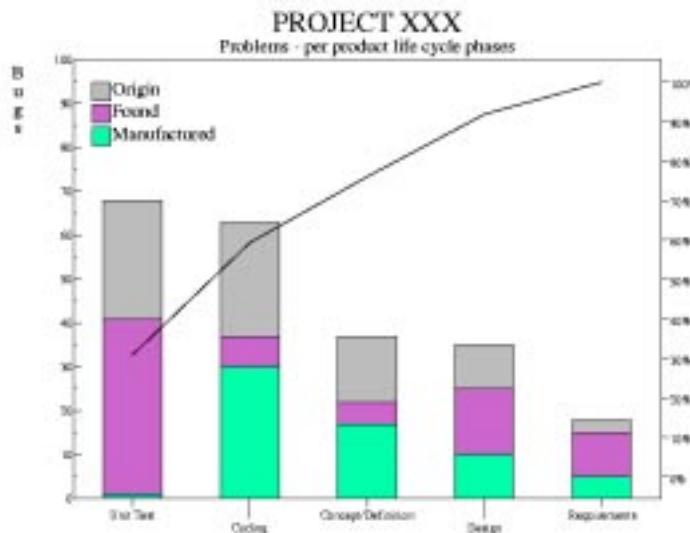
Scatter graphs

The scatter graph draws sets of markers in an x-y plane defined by horizontal and vertical axes. The set of markers are declared in the legend and the coordinates are declared in the data section. For scatter graphs, *graphbug* expects either a linear or date-based

horizontal axis type. The keyword "SCATTER" defines a scatter graph like the example graph in Figure E.6.

Figure E-6: Scatter Graph Example
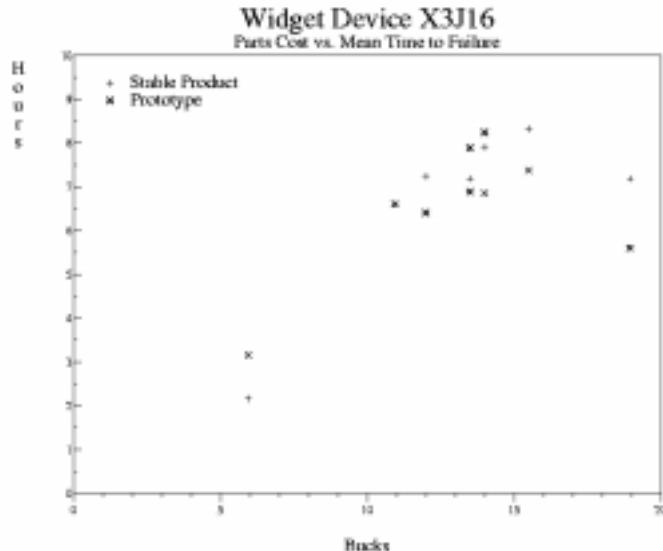
The scatter graph uses a different format in the data section. The legend of the scatter graph provides a list of names which serve as marker types within the graph. A scatter graph can define up to seven different marker types. Each data set is tagged by a marker name and contains the coordinates of a single point on the graph. Data sets for a scatter graph must use the following format:

```
marker_tag x_value : y_value
```

marker_tag	A string which exactly matches one of the strings defined in the graph legend. This parameter relates a point to a marker type.
x_value	The horizontal value of the point. This may be a date or a number.
y_value	The vertical value of the point. This may be a number or an elapsed time.

The following is part of the graph description used to create the graph in Figure E.6:

```
TITLE "Widget Device X3J16"
SUBTITLE "Parts Cost vs. Mean Time to Failure"
SCATTER "Stable Product" "Prototype"
HORIZONTAL LINEAR "Bucks" "%0.0f" MINIMUM 0 MAXIMUM 20 MAJOR 5 MINOR 1
VERTICAL LEFT HOURS "Hours" "%0.0f" MINIMUM 0 MAXIMUM 10 MAJOR 1 MINOR 0.5
DATA
"Stable Product" 5.95 : 2:10
"Prototype" 5.95 : 3:10
"Stable Product" 12.00: 7:15
"Prototype" 12.00: 6:25
"Stable Product" 13.50: 7:12
"Prototype" 13.50: 7:54
END DATA
```

Pie charts

Unlike other graphs, the legend of a pie chart is defined by the data tags present in the data section of the graph definition rather than as part of the graph type definition. Instead, the string in the graph type declaration defines an additional title displayed under the pie chart. The chart title string may be empty but must be present. Slices are drawn clockwise around the chart starting at the 12 o'clock position on the circle. The keyword "PIE" declares a pie chart.

```
PIE chart_title
```

chart_title	A title which will be printed under the pie chart.

If present, horizontal or vertical axis definitions are simply ignored for a pie chart and may be conveniently left out. The legend of a pie chart only displays string values so the input data format is restricted to data tagged by a string. The data tag may also contain the keyword "EXPLODE" which marks pie slices which should be exploded or offset from the center of the pie chart.

Any number of slices may be exploded. The general forms for pie chart data are the following:

```
label: value
label EXPLODE: value
```

label	The label for a single pie slice. The label appears in the legend of the graph.
value	A floating point number. Missing or negative values in the input are treated as a value of zero.

Graphbug can display the numeric value of each slice as a percentage of the total, as a raw number, or as hours of elapsed time. The keywords "PIE VALUE" introduce the pie value label definition. The pie value definition is optional and slices will only be labeled if it is present in the graph header:

```
PIE VALUE type format SIZE n
```

type	One of the keywords "PERCENT", "NUMBER" or "HOURS".
	▪ For the *PERCENT* type, *graphbug* calculates the percentage based on the total of all data values.
	▪ The *NUMBER* type displays the value as a raw number.
	▪ The *HOURS* type divides the data value by 60.
format	Optional. A format string can override the default format string for any of the value types. The default strings are *%.1f%%*, *%.0f*, and *%.1f*, for the percent, number and hours types.
SIZE n	Optional. The point size of the label text. The size parameter must be preceded by the keyword "SIZE" when present. The default label size is 12 points.

Figure E.7 shows an example of a pie chart which includes value labels for each slice and an exploded pie slice.

Figure E-7:   Pie Chart Example
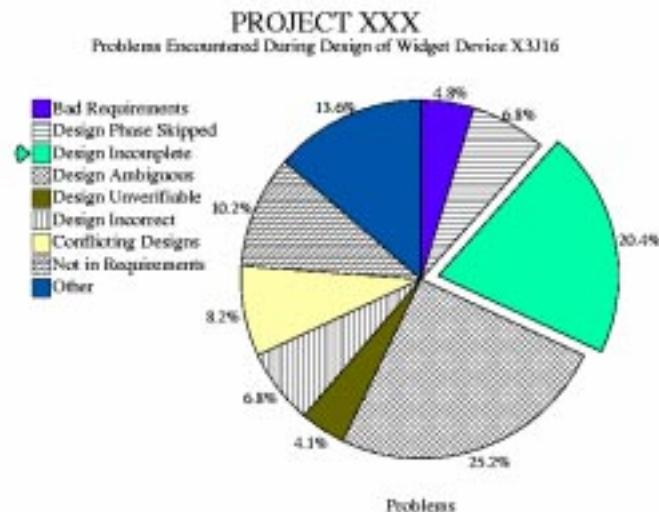
Margin definitions

The header may contain one or more lines to redefine the graph margin, the gap between bars in bar graphs, or the offset of an exploded pie slice in a pie chart. Margin declarations use the keyword "MARGINS" followed by one or more keyword-value pairs. Graph margins are defined as a number from 0.0 to 1.0 which represents a fraction of the vertical or horizontal length devoted to the margin. The bar gap is calculated as the fraction of the total width of a single stack or group of bars. The pie offset is calculated as a fraction of the radius of the pie chart.

```
MARGINS keyword value keyword value ...
```

keyword	One of the keywords "TOP", "BOTTOM", "LEFT", "RIGHT", "BAR" or "PIE".
value	A number from 0.0 to 1.0.

Legend parameters

Legend parameters affect the size and placement of the graph legend. The Keyword "LEGEND" introduces a legend parameters line. The legend of a graph can be placed on either the left or right side of the graph. The size of the legend is based on the size of the legend text. The size parameter defines the point size of the text as it appears on a full-size graph. As a special case, setting the size to zero eliminates the legend from the graph.

```
LEGEND place SIZE n
```

place	Optional. Use "LEFT" or "RIGHT" to place the legend on the left or right side of the graph. Legends are on the left side by default.
SIZE n	Optional. Determines the point size of the legend text and the size of the legend. The size parameter must be preceded by the keyword "SIZE" if it is present. The default legend font size is 16 points.

Data section

The data section follows the header section and begins with a line that contains only the keyword DATA and ends with a line that contains the keywords END DATA. Each line of the data section contains a data set which consists of a tag value and one or more data values:

```
tag : value ...
```

tag	Data set tag which may be of any type but is interpreted according to the definition of the horizontal axis. Generally, the tag type is implied by the horizontal axis definition.
value	Either a number or a time value which is interpreted according to the vertical axis definition. The number of expected data values is defined in the definition of the graph type.

Most graphs require data sets with a single tag value and one or more data values. The only exceptions are the scatter graph which uses a slightly different data format and pie charts which allow an optional keyword in the data tag. *Graphbug* does not check if the data values actually match parameters provided by the vertical axis definition in the graph header. *Graphbug* may interpret

missing or non-numeric data values as either zero or undefined depending on the graph type.

Data sets must be provided in the order that they appear along the horizontal axis of the graph. With the exception of the pareto graph, *graphbug* does change the order of the data sets. For some graphs, unsorted data sets may produce unintended results.

Notes section

An optional notes section may follow the data section. If defined, the notes section begins with the keyword NOTES and ends with the keywords END NOTES. The notes section defines textual notations that can be placed anywhere on or around the graph. Each line in the notes section defines the location, size, color, and text of a single notation:

```
color alignment size x_location y_location text
```

`color`	Optional. A color may be defined by the keyword "COLOR" followed by a color number. Colors may be defined in the configuration file. The default text color is numbered 0.
`alignment`	Optional. One of the keywords "LEFT", "RIGHT" or "CENTER". Text alignment defines where the text will be printed relative to the reference point defined by the parameters *x_location* and *y_location*. Notations are left aligned by default.
`size`	The size of the notation text in points when printed on a full-size graph.
`x_location`	The horizontal location of text. This is a number from 0.0 on the left side of the graph to 1.0 on the right side.
`y_location`	The vertical location of the baseline under the text. This is a number from 0.0 at the bottom of the graph to 1.0 at the top.
`text`	A string containing the text of the notation. This string must be quoted. The string may contain date and time references using the popular %-notation.

Graph notations may contain references to the beginning and ending dates of a graph if defined by the horizontal axis definition. Otherwise, the beginning and ending dates will be defined as the current date. Dates or elements of dates may be incorporated into a text notation using the standard '%'-format conversions. The

format conversions supported by *graphbug* are listed in the following table.

Begin date	End date	Substituted string
%%		the character '%'
%A	%a	an abbreviation for the name of the month
%C	%c	the year including century
%D	%d	the day of the month
%M	%m	the number of the month (Jan = 1)
%S	%s	the string "%m/%d/%y"
%W	%w	an abbreviation for day of the week
%X	%x	the string %d %a %c
%Y	%y	the last two digits of the year
	%t	current date and time in standard format

Graphbug configuration file

The *graphbug* program references a configuration file which may be used to override the default color tables or to add textual notation to a graph. A configuration file may be specified using the *-c* command line option. If a configuration file is not provided on the command line, *graphbug* will look for a file named *.graphbug* in the user's home directory first or for the file *~ddts/etc/graphbug.cfg*, if a user configuration file is not found.

Color definitions

Graphbug uses three separate color tables for lines, filled areas and text. Additionally, the background grid color, available with the vertical axis, can be defined separately. Colors, in *graphbug*, may be defined as RGB triplets representing the red, green and blue

components or by a color name if an X-Window *rgb.txt* file is available. Colors are defined as follows:

```
table color: red green blue
table color: "string"
```

table	Either "line", "fill", "text" or "grid".
red	A value from 0.0 to 1.0 to represent the red content of a color.
green	A value from 0.0 to 1.0 to represent the green content of a color.
blue	A value from 0.0 to 1.0 to represent the blue content of a color.
string	A color name from an X11 *rgb.txt* file. The color name must be enclosed in quotes.

If defined, *graphbug* will use the environment variable RGB_TXT as the name of an *rgb.txt* file. Otherwise *graphbug* looks for an *rgb.txt* file in the directory */usr/lib/X11*. On Sun workstations, *graphbug* also looks in the directory */usr/openwin/lib* and uses the environment variable OPENWINHOME, if it is defined.

The configuration file may override any or all of the standard color tables but not individual colors. The first color defined for a table replaces the default color table. Additional colors are added to the color table. The first color in the *text* and *line* color tables should be black. The *grid* color is not defined as a color table; defining the grid color simply changes it.

Textual notations

Textual notations can be placed anywhere on the graph. Notes defined in the configuration file support all of the options available in the graph definition and the syntax of a note declaration is nearly identical:

```
note: color alignment size x_location y_location text
```

color	Optional. A color may be defined by the keyword "COLOR" followed by a color number.
alignment	Optional. One of the keywords "LEFT", "RIGHT" or "CENTER".
size	The size of the notation text in points when printed full size.
x_location	The horizontal location of text. This is a number from 0.0 on the left side of the graph to 1.0 on the right side.

y_location	The vertical location of the baseline of the text. This is a number from 0.0 at the bottom of the graph area to 1.0 at the top.
text	A string containing the text of the notation. This string must be quoted. The string may contain date and time references using the popular %-notation defined in the table on page E-19.

The following example configuration file replaces the text color table with a table containing the colors black and red. It also adds the string "Company Confidential" in red text in the lower left corner of the graph and prints the ending date of the graph in the lower right corner:

```
text color: 0 0 0
text color: 1.0 0 0
note: COLOR 1 14 0.10 0.02 "Company Confidential"
note: RIGHT 18 0.9 0.02 "%x"
```

F

Database Reference

This appendix describes the standard table definitions in their as shipped state. You can modify the ClearDDTS database any number of ways as you add or delete fields in existing tables, add your own tables, or expand the functionality in other ways.

For the most up-to-date information, you can view the contents of the schema file directly in *$DDTSHOME/dbms/ddts/schema_file*.

As shipped, the ClearDDTS database consists of the following tables:

```
defects
enclosures
change_history
```

This appendix describes the fields associated with each of these tables.

Defect information table (defects)

The *defects* table stores all of the basic defect tracking information. It holds all of the actual defect records.

Field	Data type	Size	Description
assigned_on	datetime	*	Date defect was assigned to an engineer.
children	character	66	Children linked to this defect.
class	character	14	Class this defect belongs to.
ddts_view	character	1	Determines whether customer may view this defect. (See Chapter 9 for details.) **Note:** This field is derived from the View field in defect records. The "*ddts_*" is prepended to the field name because *view* is an SQL keyword.
duplicate_of	character	10	Record (ID) this is a duplicate of.

Field	Data type	Size	Description
duplicate_on	datetime	*	Date defect was declared a duplicate.
enclosure_count	character	2	Number of file enclosures for this defect.
engineer	character	8	Login ID of person assigned to analyze/fix the problem.
enhancement	character	1	Indicates whether this is an enhancement request (Y/N).
est_fix_date	datetime	*	Estimated date this problem will be fixed.
est_fix_hours	character	5	Estimated number of engineering hours needed to solve the problem.
fix_hours	character	5	Number of engineering hours expended to resolve problem.
forwarded_on	datetime	*	Date this record was last forwarded.
forwarded_to	character	14	Name of project to which record was forwarded.
headline	character	72	One-line description of the problem.
how_found	character	21	Strategy or method used to detect the problem.
identifier	character	10	Defect identifier.
last_mod	datetime	*	Date this defect was last modified.
new_on	datetime	*	Date defect was made new.
os_version	character	10	Name/version of operating system on which problem was found.
opened_on	datetime	*	Date this record was last opened.
origin	character	8	Flag that marks where a defect record originates from (for example, Call, Gripe, Null).
parents	character	66	Parents linked to this defect.
postponed_on	datetime	*	Date on which record was last postponed.
project	character	14	Project to which this record belongs.
resolution	character	14	Description of what was changed to correct the problem.
resolved_on	datetime	*	Date the defect was resolved.
resolver_id	character	8	Login ID of person who resolved the problem.
security_token	character	30	Mechanism for controlling who can view the defect. (See Chapter 9 for details.)

Field	Data type	Size	Description
severity	character	1	Severity of problem (by default, 1 - 5, with 1 highest).
showstopper	character	1	Indicates whether this defect is severe and stopping others' progress (Y/N).
software	character	20	Name of software module or program with the problem.
status	character	1	State of the record (N, O, A, R, etc.).
submitted_on	datetime	*	Date the record was submitted.
submitter_id	character	8	Login ID of person who submitted this record.
updated_by	character	8	Login ID of person who the last updated the record.
verified_on	datetime	*	Date this problem's fix was verified.
verifier_id	character	8	Login ID of person who verified the problem was fixed.
version	character	10	Version of software module or program with problem.
when_caused	character	15	Software lifecycle phase in which the problem was introduced.
when_fixed	character	15	Software lifecycle phase in which the problem was corrected.
when_found	character	15	Software lifecycle phase in which the problem was detected.

Note: The datetime fields marked with the asterisk (*) generally use the format YYMMDD, but can be controlled using a *date_convert* function. For more information about acceptable formats and date conversion, see Chapter 14, Managing and Customizing the ClearDDTS Database.

The following indexes are available for this table:

Index	Key(s)	Characteristics
bugid	identifier	Descending, no duplicates
bug.eg.st.sv	engineer, status, severity	Duplicates allowed
bug.pj.st.eg.sv	project, status, engineer, severity	Duplicates allowed

Index	Key(s)	Characteristics
bug.pj.st.sb.sv	project, status, submitter_id, severity	Duplicates allowed
bug.pj.st.sv	project, status, severity	Duplicates allowed
bug.sb.st.sv	submitter_id, status, severity	Duplicates allowed
bug.st.sv	status, severity	Duplicates allowed

Enclosures table (enclosures)

The *enclosures* table stores defect enclosures (related files).

Field	Data type	Size	Description
identifier	character	10	Defect identifier.
name	character	32	Name of the related file.
operation	character	16	Operation performed (for example, the enclosure was added or modified).
op_date	datetime	*	Date this enclosure was added, modified, or deleted.
engineer	character	16	Login ID of the user who made the change.
text	variable	No limit	Text of the enclosure associated with the defect.

Note: Datetime field (*) generally use the format YYMMDD, but can be controlled using a *date_convert* function. For more information about acceptable formats and date conversion, see Chapter 13, Managing and Customizing the ClearDDTS Database.

The following indexes are available for this table:

Index name	Key	Characteristics
encl_id	identifier	Ascending, duplicates allowed
encl_name	name	Ascending, duplicates allowed

History table (change_history)

The *change_history* table stores the complete history for each defect as it changes states (for example, as it is submitted, assigned, resolved, etc.).

Field	Data type	Size	Description
identifier	character	10	Defect record that was changed.
change_date	datetime	*	Date the record was last changed.
engineer	character	16	User who made the change.
text	variable	No limit	History of changes made to the defect.

Note: Datetime fields (*) generally use the format YYMMDD, but can be controlled using a *date_convert* function. For more information about acceptable formats and date conversion, see Chapter 13, Managing and Customizing the ClearDDTS Database.

The following index is available for this table:

Index name	Key	Characteristics
chg_hist_def_id	identifier	Ascending, duplicates allowed

G

Using an Oracle Database

Although ClearDDTS includes its own internal SQL database, you can also use an Oracle database. This appendix provides some general information about using ClearDDTS with an Oracle database and provides guidelines for setting up your database to handle defect tracking. You should be sure to consult the documentation from Oracle for more complete information about installing and managing your system.

This appendix does not provide complete definitions for vendor-specific terms; if you are unfamiliar with the terminology used in this appendix, consult your Oracle database documentation for more information.

Identifying the database vendor

When you install ClearDDTS, the internal SQL database is installed by default. To configure ClearDDTS to work with an Oracle RDBMS, you use the *adminbug chdb* command to identify the database. This information is stored in the file *~ddts/dbms/conf/dbvendor* using the following format:

```
Vendor: <database>
```

When you change databases using the *adminbug chdb* command, this file is updated with the appropriate information.

The *adminbug chdb* command also creates two other important database-related files *in the ~ddts/dbms/conf* directory. These files are named according to the database vendor for which they contain information. For an Oracle database, these files are *oracle* and *oracle.priv.*

Before running the *adminbug chdb* command, read the following section and complete the Oracle database setup.

Working with an ORACLE database

After installing ClearDDTS, you can run the *adminbug chdb* command to indicate that you want to use an Oracle database. Running this command sets up your *oracle* and *oracle.priv* files. These files identify the specific database instance to use for ClearDDTS and provide database security so that all users can run queries (read-only access) but only the database administrator (the special "user" *ddts*) can make changes to the database.

The *oracle* file makes the database public (readable) for most users, using the following format:

```
Instance: <database_instance>
ReadOnlyUser: <username>
ReadOnlyPassword: <password>
```

The "private" file (*oracle.priv*) restricts access to the database so that only ClearDDTS and the DBA can perform updates:

```
ReadWriteUser: ddts
ReadWritePassword: <ddts_password>
```

These two files refer to special database users — the read-only user and *ddts* — who have specific privileges. You can use Oracle's SQL*DBA or SQL commands to create these users.

This section describes the steps you must take to set up Oracle tablespaces, rollback segments, and finally the special users. When these steps are completed you can run the *adminbug chdb* command to switch to the Oracle database.

Creating tablespaces

Before you begin creating tables for ClearDDTS, you need to create some additional tablespaces and rollback segments to hold your ClearDDTS installation. For a typical installation, you can use the following guidelines as a starting point; however larger sites may want to increase these values:

Tablespace	Recommended Size
ddts_tables for ClearDDTS data tables	60MB
ddts_tmp for temporary tables	5MB

To create tablespaces for ClearDDTS, use Oracle's SQL*DBA or the SQL command *create tablespace*. Remember to set aside enough tablespace to handle growth, and enough to rollback table space to commit the data.   For example:

```
CREATE TABLESPACE ddts_tables
datafile'/disk1/clearddts_oracle/ddtstbl01.dbf'
SIZE 30 M;

CREATE TABLESPACE ddts_tmp
datafile'/disk1/clearddts_oracle/ddtstmp01.dbf'
SIZE 5 M;

CREATE TABLESPACE ddts_rbs
datafile'/disk1/clearddts_oracle/ddtsrbs01.dbf'
SIZE 5 M;
```

Note: The *ddtsdbbuild* program commits after inserting 250 records. To change the commit rate run *ddtsdbbuild* directly, instead of through *adminbug dbms,* as follows:

```
ddtsdbbuild -commit_every <N>
```

where N is the number of records you want to process before a commit.

You can also have *ddtsdbbuild* print more verbose SQL information in the *~ddts/spool/ADMINLOG* by running:

```
ddtsdbbuild -verbose
```

Creating rollback segments

After you have created the tablespaces for ClearDDTS, you are ready to create rollback segments. To get started, use SQL*DBA or the *create rollback segment* command to create a temporary rollback segment in the SYSTEM tablespace. For example:

```
CREATE ROLLBACK SEGMENT tmp_rbs
TABLESPACE system
STORAGE (INITIAL 250K MINEXTENTS 5);
```

Bring this rollback segment online and add it to the *init<ORACLE_SID>.ora* file. You are now ready to create the *ddts_rbs* rollback segment. For example:

```
CREATE ROLLBACK SEGMENT ddts_rbs1
TABLESPACE ddts_rbs
STORAGE (INITIAL 250K MINEXTENTS 5);

CREATE ROLLBACK SEGMENT ddts_rbs2
TABLESPACE ddts_rbs
STORAGE (INITIAL 250K MINEXTENTS 5);

CREATE ROLLBACK SEGMENT ddts_rbs3
TABLESPACE ddts_rbs
STORAGE (INITIAL 250K MINEXTENTS 5);

CREATE ROLLBACK SEGMENT ddts_rbs4
TABLESPACE ddts_rbs
STORAGE (INITIAL 250K MINEXTENTS 5);
```

Creating database users

Although any database user can be the owner of the ClearDDTS tables, for security reasons, you may want to create a *ddts* database user and make this user the owner of the ClearDDTS database tables. For example:

```
CREATE USER ddts
IDENTIFIED BY ddts
DEFAULT TABLESPACE "DDTS_TABLES"
TEMPORARY TABLESPACE "DDTS_TMP"
PROFILE "DEFAULT";
ALTER USER ddts
DEFAULT ROLE ALL
PROFILE "DEFAULT";
GRANT "CONNECT" TO "DDTS";
```

In addition to the user *ddts*, you should create a read-only user (if one does not already exist) to provide normal users with read access to all ClearDDTS tables. For example:

```
CREATE USER readonly
IDENTIFIED BY readonly
DEFAULT TABLESPACE "USERS"
TEMPORARY TABLESPACE "TEMP"
PROFILE "DEFAULT";
ALTER USER readonly
DEFAULT ROLE ALL
PROFILE "DEFAULT";
GRANT "CONNECT" TO "READONLY";
```

Depending on the requirements of your installation, the users, privileges, and roles you need to establish may vary. Refer to your Oracle documentation for more complete information.

Creating tables

Once the Oracle setup is complete, log in as the user *ddts* and run the *adminbug chdb* command. After running *adminbug chdb*, ClearDDTS automatically runs the *adminbug dbms* command to build the database. This command creates the ClearDDTS tables using Oracle's default storage parameters. If you want to make changes to tables, follow the procedures in your Oracle documentation. You can use the information in Appendix A, to help you identify the fields you will need. Here is a simplified example of table creation:

```
CREATE TABLE defects (
      identifier  CHAR(10),
      class       CHAR(14),
      .
      .
      .                 )
```

Since ClearDDTS creates the tables with the default storage parameters, you should use your Oracle tools to monitor and tune your system as appropriate for your environment. Refer to your Oracle documentation for complete information about estimating table size, adjusting table parameters, and managing the database.

Searching enclosures

The ClearDDTS SQL database server allows users to search unlimited length enclosures in queries. However, this feature is not available if you are using Oracle as the SQL server.

As a workaround, you can use fixed length text of 1 or 2Kb to store the enclosure fields. This allows you to search the first 1 or 2Kb of enclosure text in your searches.

H
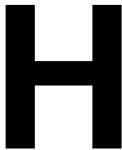
Information Resources on the Web

This appendix lists some locations of resources on the World Wide Web that describe HTTP servers, HTML, and CGI scripts.

HTTP servers

Apache

```
http://www.apache.org
```

Netscape

```
http://home.netscape.com/comprod/server_central/index.html
```

General

```
http://webcompare.iworld.com/compare/chart.html
```

```
http://www-genome.wi.mit.edu/WWW/faqs/www-security-faq.html
```

FastCGI

```
http://www.fastcgi.com
```

HTML

```
http://www.ncsa.uiuc.edu/General/Internet/WWW/HTMLPrimer.html
```

```
http://www.ncsa.uiuc.edu/SDG/Software/Mosaic/Docs/fill-out-form
s/overview.html
```

```
http://lcweb.loc.gov/global/internet/html.html
```

```
http://www.nlc-bnc.ca/ifla/I/training/colour/colour.htm
```

CGI scripts

```
http://hoohoo.ncsa.uiuc.edu/cgi
```

```
http://www.cgibook.com/
```

Index