

CRAY T90™ Series CBP Runtime Module

Preliminary Information
HDM-xxx-0
December 1994

PRELIMINARY INFORMATION
DO NOT DISSEMINATE
Cray Research Proprietary

Cray Research, Inc.

Record of Revision

| REVISION | DESCRIPTION |
|----------|-------------|
|----------|-------------|

December 1994. Preliminary information for classroom use only.

| |
|--|
| Any shipment to a country outside of the United States requires a letter of assurance from Cray Research, Inc. |
|--|

This document is the property of Cray Research, Inc. The use of this document is subject to specific license rights extended by Cray Research, Inc. to the owner or lessee of a Cray Research, Inc. computer system or other licensed party according to the terms and conditions of the license and for no other purpose.

Cray Research, Inc. Unpublished Proprietary Information — All Rights Reserved.

Autotasking, CF77, CRAY, CRAY-1, Cray Ada, CRAY Y-MP, HSX, SSD, UniChem, UNICOS, and X-MP EA are federally registered trademarks and CCI, CF90, CFT, CFT2, CFT77, COS, CRAY-2, Cray Animation Theater, CRAY C90, CRAY C90D, Cray C++ Compiling System, CrayDoc, CRAY EL, CRAY J90, Cray NQS, Cray/REELlibrarian, CraySoft, CRAY T3D, CRAY X-MP, CRAY XMS, CRInform, CRI/TurboKiva, CSIM, CVT, Delivering the power . . ., DGauss, Docview, EMDS, HEXAR, IOS, LibSci, MPP Apprentice, ND Series Network Disk Array, Network Queuing Environment, Network Queuing Tools, OLNET, RQS, SEGLDR, SMARTE, SUPERCLUSTER, SUPERLINK, Trusted UNICOS, and UNICOS MAX are trademarks of Cray Research, Inc.

Requests for copies of Cray Research, Inc. publications should be directed to:

CRAY RESEARCH, INC.
Logistics
6251 South Prairie View Road
Chippewa Falls, WI 54729

Comments about this publication should be directed to:

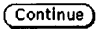

CRAY RESEARCH, INC.
Hardware Publications and Training
890 Industrial Blvd.
Chippewa Falls, WI 54729

CRAY T90 SERIES CBP RUNTIME MODULE

| | |
|---|----|
| Notational Conventions | 2 |
| Overview | 2 |
| Starting the CRAY T90 Series CBP Runtime Module | 3 |
| Command Buffer Programs and Files | 3 |
| CRAY T90 Series CBP Commands | 4 |
| MME-specific Command Descriptions | 4 |
| MMEAlloc() | 5 |
| MMEAssign() | 7 |
| MMECtrlptGet() | 7 |
| MMECtrlptInfo() | 8 |
| MMECtrlptSectionInfo() | 8 |
| MMECtrlptSetup() | 8 |
| MMEDeassign() | 9 |
| MMEEror() | 9 |
| MMEGo() | 9 |
| MMEHalt() | 10 |
| MMELoad() | 10 |
| MMERead() | 10 |
| MMEReload() | 11 |
| MMEReset() | 11 |
| MMETimeout() | 11 |
| MMEUnload() | 11 |
| MMEWait() | 11 |
| MMEWrite() | 12 |

Notational Conventions

This document uses the following notational conventions:

- Buttons are shown the way they appear in a window; for example,  .
- The → symbol indicates that you need to hold the MENU mouse button down and move the mouse pointer to the next menu item.
- Helvetica type indicates a reference to a window of the interface.
- **Helvetica bold** type indicates a menu entry you should choose from the interface; for example, “choose **Dir → CBP Default**” indicates you should choose the CBP Default entry from the  menu button.
- Courier type indicates a command.
- The following conventions are used in the command descriptions:
 - *Italic* type indicates a variable.
 - Square brackets [] indicate an optional entry.
 - A vertical bar | indicates a choice.

Overview

The Command Buffer Parser (CBP) application contains different runtime modules used to troubleshoot the different types of hardware CBP supports. This document describes the CRAY T90 series CBP runtime module that enables you to automate troubleshooting tasks performed by the Mainframe Maintenance Environment (MME) environment 1, MME environment 2, and Logic Monitor Environment (LME) for CRAY T90 series mainframes.

This document includes the procedure to start CBP with the CRAY T90 series CBP runtime module, the command buffer programs that are available for use with the CRAY T90 series CBP runtime module, and the command buffer parser commands that are specific to the CRAY T90 series CBP runtime module.

For general information about the CBP application, including descriptions of the user interface, CBP programming, and general-purpose commands, refer to the *Command Buffer Parser User Guide*, publication number HDM-076-0.

Starting the CRAY T90 Series CBP Runtime Module

Choose **Utilities** → **Command Buffer** in the MME environment 1 base window, MME environment 2 base window, or LME base window to start CBP with the CRAY T90 series CBP runtime module.

NOTE: You cannot start the CRAY T90 series CBP runtime module from MME environment 0. The CRAY T90 series CBP runtime module does not support MME environment 0.

Command Buffer Programs and Files

You can access several command buffer programs specific to the CRAY T90 series CBP runtime module with the **Dir:** menu button in the CBP: Load/Save window. This menu structure is shown in Figure 1.

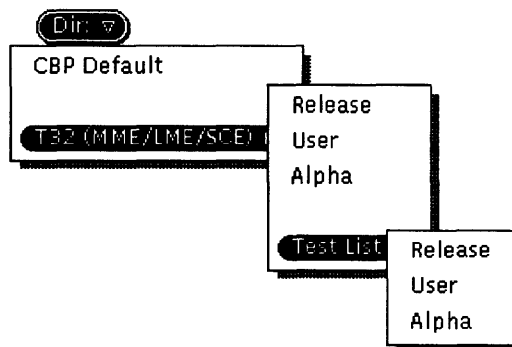


Figure 1. Dir Menu Structure

The following menu options are available:

| <u>Menu Option</u> | <u>Description</u> |
|---------------------|---|
| Release | CRAY T90 series command buffer programs included in the current offline diagnostic release. |
| User | CRAY T90 series command buffer programs that you modify or create, and then save. |
| Alpha | CRAY T90 series command buffer programs that are not officially released. |
| Test List → Release | CRAY T90 series test lists included in the current offline diagnostic release. |

| <u>Menu Option</u> | <u>Description</u> |
|--------------------|--|
| Test List -> User | CRAY T90 series test lists that you modify or create, and then save. |
| Test List -> Alpha | CRAY T90 series test lists that have not been officially released. |

CRAY T90 Series CBP Commands

The CRAY T90 series CBP runtime module includes active CBP commands that perform functions specific to the CRAY T90 series MME, configuration, and LME applications. *Currently, the only commands available for the CRAY T90 series CBP runtime module are MME-specific commands.*

MME-specific Command Descriptions

Use the MME-specific commands (refer to Table 1) in your command buffer programs to manipulate the MME interface and to automate CRAY T90 series mainframe testing. Descriptions of the options for each command follow the table.

Table 1. MME-specific Commands

| Command | Description |
|-------------------------|--|
| MMEAlloc() | Sets up resource allocation options |
| MMEAssign() | Assigns a control point to a specific CPU or all CPUs |
| MMECtrlptGet() | Returns status information about a specific executing control point |
| MMECtrlptInfo() | Returns configuration information about a specific control point |
| MMECtrlptSectionInfo() | Returns configuration information about a specific control point section |
| MMECtrlptSetup() | Sets configuration information for a specific control point or for all current control points |
| MMEDeassign() | Deassigns a specific CPU from the control point it is assigned to or deassigns all CPUs assigned to a specific control point |
| MMEError() | Returns a value that indicates whether a specific control point or any control point has an error |
| MMEGo() | Starts a specific control point or all loaded control points |
| MMEHalt() | Stops a specific control point or all executing control points |
| MMELoad() | Loads a control point |
| MMERead() | Reads a block of data from mainframe memory |

Table 1. MME-specific Commands (continued)

| Command | Description |
|---------------|--|
| MMEReload() | Reloads a specific control point or all control points |
| MMEReset() | Resets MME to the current environment or environment 1 or 2 |
| MMETimeout() | Returns a value that indicates whether a specific control point or all control points timed out |
| MMEUnload() | Unloads a specific control point or all control points |
| MMEWait() | Waits a specified number of seconds and returns a value that indicates whether any control points reached timeout values or found an error |
| MMEWrite() | Writes a block of data to mainframe memory |

MMEAlloc()

```
MMEAlloc(CACHE, value|ALL, ENABLE|DISABLE);
```

This command enables or disables the scalar cache for the CPU specified by *value* or for all CPUs if ALL is specified.

```
MMEAlloc(CPUMODE, AUTO|MANUAL);
```

This command enables or disables automatic CPU assignment when a control point is loaded.

```
MMEAlloc(IOICM, value|ALL, ENABLE|DISABLE);
```

This command enables or disables the interrupt on correctable memory error (ICM) option for the CPU specified by *value* or for all CPUs if ALL is specified; this option sets or clears the ICM flag in the exchange package for the specified CPU(s).

```
MMEAlloc(IOCPU, value);
```

This command sets the I/O CPU (the CPU path used to write to memory and read from memory) to *value*; *value* can be a constant or variable.

```
MMEAlloc(IOIRP, value|ALL, ENABLE|DISABLE);
```

This command enables or disables the interrupt on register parity error (IRP) option for the CPU specified by *value* or for all CPUs if ALL is specified; this option sets or clears the IRP flag in the exchange package for the specified CPU(s).

```
MMEAlloc( IOIUM, value|ALL, ENABLE|DISABLE );
```

This command enables or disables the interrupt on uncorrectable memory error (IUM) option for the CPU specified by *value* or for all CPUs if ALL is specified; this option sets or clears the IUM flag in the exchange package for the specified CPU(s).

```
MMEAlloc( MEMDELAY, value|ALL, 0|4|16|63 );
```

This command delays sending the signal to common memory that indicates a CPU is ready to accept another word of data. You can set a delay of 0, 4, 16, or 63 clock periods for the CPU specified by *value* or for all CPUs if ALL is specified.

For more information about this delay, refer to the “Memory Input Ports Maintenance Functions” description in Section 4 of the *Triton Maintenance System Engineering Note*, publication number PRN-0957.

```
MMEAlloc( MEMMODE, BOTTOM_UP|RANDOM|TOP_DOWN );
```

This command sets the memory mode to bottom up, random, or top down.

```
MMEAlloc( MEMMODE, PARTITION_UP, SIZE|COUNT, value );
```

This command sets the memory mode to bottom up with partitions that have the size specified by *value* or with *value* number of partitions, depending on whether size or count was specified.

```
MMEAlloc( MEMMODE, PARTITION_DOWN, SIZE|COUNT, value );
```

This command sets the memory mode to top down with partitions that have the size specified by *value* or with *value* number of partitions, depending on whether size or count was specified.

```
MMEAlloc( SBCDBD, value|ALL, ENABLE|DISABLE );
```

This command enables or disables the single-byte correction/double-byte detection (SBCDBD) feature for the CPU specified by *value* or for all CPUs if ALL is specified.

```
MMEAlloc( SECDED, value|ALL, ENABLE|DISABLE );
```

This command enables or disables the single-error correction/double-error detection (SECDED) feature for the CPU specified by *value* or for all CPUs if ALL is specified.

MMEAssign()

```
MMEAssign( tag, value|ALL);
```

This command assigns the CPU specified by *value* or all CPUs to the control point referenced by *tag*.

MMECtrlptGet()

```
MMECtrlptGet( tag, CPU_SELECT);
```

This command returns a bit mask that indicates the CPU(s) assigned to the control point referenced by *tag*.

```
MMECtrlptGet( tag, CURRENT_CPU);
```

This command returns the number of the CPU that is executing the control point referenced by *tag*.

```
MMECtrlptGet( tag, CURRENT_SECTION);
```

This command returns the number of the section that is currently executing for the control point referenced by *tag*.

```
MMECtrlptGet( tag, ERROR_COUNT);
```

This command returns the current global error count for the control point referenced by *tag*.

```
MMECtrlptGet( tag, IS_ACTIVE);
```

This command returns a 1 if the control point referenced by *tag* is active and returns a 0 if the control point is not active.

```
MMECtrlptGet( tag, PASS_COUNT);
```

This command returns the current global pass count for the control point referenced by *tag*.

```
MMECtrlptGet( tag, SECTION_SELECT);
```

This command returns a bit mask that indicates the sections that are selected for the control point referenced by *tag*.

MMECtrlptInfo()

```
MMECtrlptInfo(tag, section_select, cpu_select,
pass_count, error_count);
```

This command returns information about the control point referenced by *tag*. All parameters following *tag* are used to return values.

The `MMECtrlptinfo()` command returns the section select mask, CPU select mask, pass count, and error count to the variables in the *section_select*, *cpu_select*, *pass_count*, and *error_count* positions. You can use any valid variable to receive these values. If you do not need a return value, set the appropriate parameter to 0 or NULL.

MMECtrlptSectionInfo()

```
MMECtrlptSectionInfo(tag, section_select, cpu_select,
pass_count, error_count);
```

This command returns information about a specific section of a control point executing in a specific CPU. The *section_select* parameter specifies which section of the control point is referenced by *tag*. The *cpu_select* parameter specifies which CPU to take the information from.

The `MMECtrlptSectionInfo()` command returns the pass count and error count to the variables in the *pass_count* and *error_count* positions. You can use any valid variable to receive these values. If you do not need a return value, set the appropriate parameter to 0 or NULL.

MMECtrlptSetup()

```
MMECtrlptSetup(tag|ALL, section_mask|ALL_SECTIONS, PASS |
TIME, value);
```

This command sets the section select and the pass or time limit for the control point referenced by *tag* or for all current control points.

The *section_mask* parameter contains a mask of sections to select; use 0 to indicate no change from the current sections selected and ALL_SECTIONS to select all sections of the control point. The PASS|TIME option specifies that the control point executes for a specific number of passes or a specific amount of time. If you use PASS, *value* is a pass count; if you use TIME, *value* is a time limit (in seconds).

MMEDeassign()

```
MMEDeassign(value);
```

This command deassigns the CPU specified by *value* from whatever control point is assigned to it.

```
MMEDeassign(ALL, tag);
```

This command deassigns all CPUs assigned to the control point referenced by *tag* or all CPUs assigned control points if ALL is specified.

MMEError()

```
MMEError(tag|ALL);
```

This command returns a nonzero (true) value if the control point referenced by *tag* (or any control point if ALL is specified) has an error and returns a zero (false) value otherwise.

MMEGo()

```
MMEGo(tag|ALL);
```

This command starts the control point referenced by *tag* or all loaded control points if ALL is specified. If no conditions were previously set for the specified control points by MMECtrlptSetup() or MMEGo() commands, the pass and error counts default to -1. This causes the control point to run indefinitely.

```
MMEGo(tag|ALL,section_mask|ALL_SECTIONS,PASS|TIME,  
value);
```

This command performs the `MMECtrlptSetup()` command and then starts the control point referenced by *tag* or all loaded control points. (Refer to the description of the `MMECtrlptSetup()` command for more information.)

MMEHalt()

```
MMEHalt(tag|ALL[,NO_DUMP|EXCHANGE_DUMP|  
REGISTER_DUMP]);
```

This command stops the control point referenced by *tag* or stops all control points if `ALL` is specified. The optional parameter specifies the halt dump mode; if you omit this parameter, no dump occurs.

MMELoad()

```
MMELoad("string"[,ALL]);
```

This command loads the control point contained in the file specified by *string*. This command returns an integer value called a *tag*, which you use with other commands to reference the control point. The optional `ALL` parameter assigns all unassigned CPUs to the control point.

MMERead()

```
MMERead([tag,]buffer,address,length);
```

This command reads a block of data from mainframe memory. The block begins at the word address specified by *address* and ends with the address specified by *length*. If you specify a *tag*, addresses are relative to the beginning of the control point referenced by *tag*; otherwise, the addresses are absolute.

The `MMERead()` command copies data into *buffer*, which must be an array of byte, short, uint, or long data and must be large enough to store the amount of data requested.

MMEReload()

```
MMEReload(tag|ALL);
```

This command reloads the control point referenced by *tag* or reloads all control points if ALL is specified.

MMEReset()

```
MMEReset([1|2]);
```

This command resets MME in the current environment or resets MME to the specified environment.

NOTE: The CRAY T90 series CBP runtime module does not support environment 0.

MMETimeout()

```
MMETimeout(tag, label);
```

This command returns a nonzero (true) value if the control point referenced by *tag* (or any control point) timed out and a zero (false) value otherwise.

MMEUnload()

```
MMEUnload(tag|ALL);
```

This command unloads the control point referenced by *tag* or all control points if ALL is specified.

MMEWait()

```
MMEWait(limit);
```

This command waits for *limit* seconds and returns logical false (zero) if any control points reach their time limits or find an error. This command returns logical true (nonzero) if the control points reach their pass limits or the user of your command buffer parser program clicks the button in the CBP base window to stop the MMEWait() command.

MMEWrite()

```
MMEWrite([tag,]buffer,address,length);
```

This command writes a block of data to mainframe memory. This command reads the data from *buffer*, which must be an array of byte, short, uint, or long data and must contain at least as much data as was requested.

The *MMEWrite()* command writes data to the word address specified by *address* and ends with the address specified by *length*. If you specify a *tag*, addresses are relative to the beginning of the control point referenced by *tag*; otherwise, addresses are absolute.

Reader Comment Form

Title: **CRAY T90™ Series**
CBP Runtime Module
Preliminary Information

Number: **HDM-xxx-0**
December 1994

Your feedback on this publication will help us provide better documentation in the future. Please take a moment to answer the few questions below.

For what purpose did you primarily use this document?

- Troubleshooting
- Tutorial or introduction
- Reference information
- Classroom use
- Other - please explain _____

Using a scale from 1 (poor) to 10 (excellent), please rate this document on the following criteria and explain your ratings:

- Accuracy _____
- Organization _____
- Readability _____
- Physical qualities (binding, printing, page layout) _____
- Amount of diagrams and photos _____
- Quality of diagrams and photos _____

Completeness (Check one)

- Too much information _____
- Too little information _____
- Just the right amount of information

Your comments help Hardware Publications and Training improve the quality and usefulness of your publications. Please use the space provided below to share your comments with us. When possible, please give specific page and paragraph references. We will respond to your comments in writing within 48 hours.

NAME _____
JOB TITLE _____
FIRM _____
ADDRESS _____
CITY _____ STATE _____ ZIP _____
DATE _____

[or attach your business card]

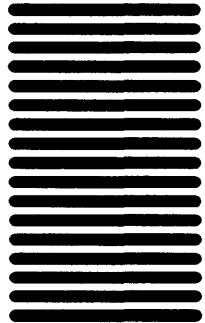


CUT ALONG THIS LINE

Fold



NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES



BUSINESS REPLY CARD

FIRST CLASS PERMIT NO 6184 ST. PAUL, MN

POSTAGE WILL BE PAID BY ADDRESSEE



**Attn: Hardware Publications and Training
890 Industrial Boulevard
Chippewa Falls, WI 54729**

Fold

STAPLE