# COS DCE FSW Test Methodology

| | |
|---|---|
| Date: | February 12, 2001 |
| Document Number: | COS-01-0006 |
| Revision: | Initial Release |
| Contract No.: | NAS5-98043 |
| CDRL No.: | N/A |

Prepared By: _____ _____

Tim Swanson, Software Test Engineer, Design_Net Eng. Date

Reviewed By: _____ _____

K. Brownsberger, COS Sr. Software Scientist, CU/CASA Date

Reviewed By: _____ _____

Grant Blue, COS Software & Operations Manager, BATC Date

Approved By: _____ _____

Barry Welsh, FUV Detector Program Manager. UCB Date

Approved By: _____ _____

John Andrews, COS Experiment Manager, CU/CASA Date


**Center for Astrophysics & Space Astronomy**
University of Colorado
Campus Box 593
Boulder, Colorado 80309

# REVISIONS

| Letter | ECO No. | Description | Check | Approved | Date |
|---|---|---|---|---|---|
| - | | Initial Release | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |

| Original Release Name | Date |
|---|---|
| Drawn: K Brownsberger | 2-12-01 |
| Reviewed: | |
| Approved: | |
| | |
| | |

**THE UNIVERSITY OF COLORADO**
At Boulder
**The Center for Astrophysics and Space Astronomy**

COS DCE FSW Test Methodology

| Size | Code Indent No. | Document No. | Rev |
|---|---|---|---|
| A | | COS-01-0006 | - |
| Scale: N/A | | | |

## Table of Contents

# 1. INTRODUCTION

This memorandum proposes a concept for the hardware/software configuration to be employed in the testing of DCE Flight Software (FSW). It describes all the components, both hardware and software, required to conduct these tests.

The proposed concept has already been implemented in another form for another testing problem, namely that of DCE stack hardware functional testing. The two custom pieces of software described herein are rewrites of code developed by the author and used successfully for that application.

# 2. COS DCE SOFTWARE TEST CONCEPT

An overview of the hardware/software configuration for conducting most of the DCE software tests is shown in Figure 2-1. The basic software test concept, as illustrated in the Figure, is (under control of a "test-script")

- to induce program activity in the 8051 DCE Board B processor by the EGSE's sending DCE commands (including a special new command) to it, and
- to down-link data from the 8051 to the EGSE from which it can be inferred whether the induced activity "succeeded" or "failed".

## 2.1 SOFTWARE TEST PHASES

In detail, a software test consists, in general, of the phases summarized in Table 2-1:

**Table 2-1: Summary of Phases of A Software Test**

| Phase | Hardware/ Software Used | Description |
|---|---|---|
| Prepare | • PC/editor<br>• PC/IDE<br>• PC/**HEX2CMD** | • edit, compile, link **UniTest** program, producing **.hex** output file<br>• convert **.hex** output of IDE to **.csf**<br>• create test-script<br>• edit **UniScript.pl** as necessary |
| Start | • PC/**TELNET**<br>• EGSE/Unix | • create appropriate EGSE directories |
| | • PC/**FTP**<br>• EGSE/Unix | • refresh **UniScript.pl** if necessary<br>• transfer **UniTest.csf** from PC to EGSE if necessary |

| Phase | Hardware/ Software Used | Description |
|---|---|---|
| | | • transfer test-script file from PC to EGSE if necessary |
| | • PC/**TELNET** | • up-link **UniTest** to DCE 8051 by executing **UniTest.csf** shell script if necessary |
| Setup | • PC/**TELNET** | • Start test-script interpreter (**UniScript**) |
| | • EGSE/**UniScript**<br>• 8051/FSW<br>• 8051/**UniTest** | **UniScript** performs any preliminary computations per pseudo-commands in the script, sends **LFDUTEST** or other DCE commands per the script to:<br>• designate contents of selected RAM/ROM locations to be saved (as necessary)<br>• designate 8051 registers (**A**, **B**, **C**, **PSW**, **R0**...**R7**) to be saved (as necessary)<br>• specify register contents to be used for test (as necessary) |
| Induce Execution | • EGSE/**UniScript**<br>• 8051/FSW<br>• 8051/**UniTest** | • up-link DCE command to invoke a FSW function, or<br>• up-link **LFDUTEST** command to invoke FSW code component via **UniTest** |
| Collect Outputs | • 8051/**UniTest**<br>• 8051/FSW<br>• EGSE/**UniScript** | • restore 8051 registers, RAM locations (as necessary)<br>• let FSW regain control (if necessary)<br>• get contents of selected registers, RAM locations (as necessary) via LFDDNLOD command or standard housekeeping down-link |
| Analyze Outputs | • EGSE/**UniScript** | • place formatted output values in report file(s)<br>• evaluate Boolean expressions involving output values<br>• report results |

The three computer systems represented in Figure 2-1 and referred to in Table 2-1 are described in the following paragraphs.

## 2.2 CATEGORIES OF SOFTWARE USED IN THE VARIOUS COMPUTER SYSTEMS

### 2.2.1 Executive

For each computer, the executive software is either an operating system (Unix for the SPARCstations, Windows for the PC) or a program performing general supervisory control of the host computer (the DCE FSW for the 8051).

### 2.2.2 Pre-existing/Commercial Applications

For each computer except the 8051 certain pre-existing or commercial application modules will be used to supplement the functions provided by the operating system. In the case of the EGSE, these applications have been specially written by the SSL staff. For the PC the pre-existing applications are either commercial packages or freeware downloaded from the Internet.

### 2.2.3 Custom Applications

Two custom applications will be written to support the DCE FSW testing. These execute, respectively, on the EGSE and the DCE 8051.

## 2.3 COMPUTER SYSTEMS USED

### 2.3.1 DCE Board B Processor System

This computer (a space-hardened version of the Intel 8051) *performs* the tests. It receives commands from the EGSE and processes them. For some software tests (especially the Unit 1 and Unit 2 tests) a special test program, **UniTest**, which is not part of the DCE FSW, performs setup, activates some component of the DCE program, and organizes the test-results data for down-link (section 2.3.1.3). The EGSE receives **UniTest** via FTP from the PC system (section 2.3.2.3), then transmits it to the 8051. Once it has been loaded in this way it can be invoked by special non-flight commands temporarily defined only to support software testing. **UniTest** therefore constitutes a temporary extension of the DCE software; it is connected to the latter by means of an entry in the DCE **CMD_JMP** table. A power-On reset (POR) effaces **UniTest**, leaving the DCE FSW in its normal state.

2.3.1.1 Executive (DCE FSW)

The DCE 8051 does not, properly speaking, have an operating system in the normal sense of the term. Instead, all supervisory functions are performed by the FSW itself.

2.3.1.2 Pre-existing/Commercial Applications

No pre-existing applications, other than FSW itself, execute on the 8051.

2.3.1.3 Custom Software: UniTest

**UniTest** will be developed primarily in the Keil µVision2 Integrated Development Environment (section 2.3.3.2.3). **UniTest** is invoked by the FSW Command Processing function whenever the latter recognizes and **LFDUTEST** command. The functions of **UniTest** are as follows:

*2.3.1.3.1 Set Up for Execution*

**UniTest** processes **REG** and **SET** subcommands by storing values up-linked in **LFDUTEST** commands until they are needed to provide the context for the invocation of an FSW module.

*2.3.1.3.2 Execute FSW Code*

**UniTest** processes the **XFER** sub-command, transferring control to and FSW module by

- setting 8051 registers and RAM locations to values that had been pre-specified (section 2.3.1.3.1), having first saved the contents of these registers/locations in a special save area
- either JUMPing to or CALLing an address specified to it in an **LFDUTEST** command
- receiving control back from the FSW module and restoring registers and RAM locations to the previously saved values
- returning control to the FSW command processing routine.

2.3.2 EGSE System

This computer (a Sun SPARCstation) *controls* the tests. It transmits DCE commands to the 8051 via the 1 MHz "three-wire" serial synchronous interface (implemented by the "EDT" boxes in Figure 2-1), and performs other operations, all under the control of a script — that is, a kind of checklist, that it reads from a designated disk file (***.txt** in the Figure). The test-script interpreter, **UniScript.pl** (section 2.3.2.3), opens

the script file and the associated report files as specified in the Unix command that executes it, then reads the script, line by line, performing the operations specified therein. Some of these operations are handled locally; others represent DCE commands (including the commands that invoke **UniTest**) and are converted by **UniScript** to the appropriate DCE command packets, then up-linked to the 8051 by means of the **cmd_client** subsystem (section 2.3.2.2.2). Once a test has been executed, **UniScript** requests appropriate results from **hkprint** (section 2.3.2.2.1). These results are then formatted by **UniScript** and written to a test-report file (***.rp1** or ***.rp2** in the Figure).

#### 2.3.2.1 Executive

The executive program for the EGSE is SunOS 5.7, a variant of Unix. In addition to the Unix file system and shell capabilities, software testing relies upon several Unix functions available through the Perl language (section 2.3.2.3).

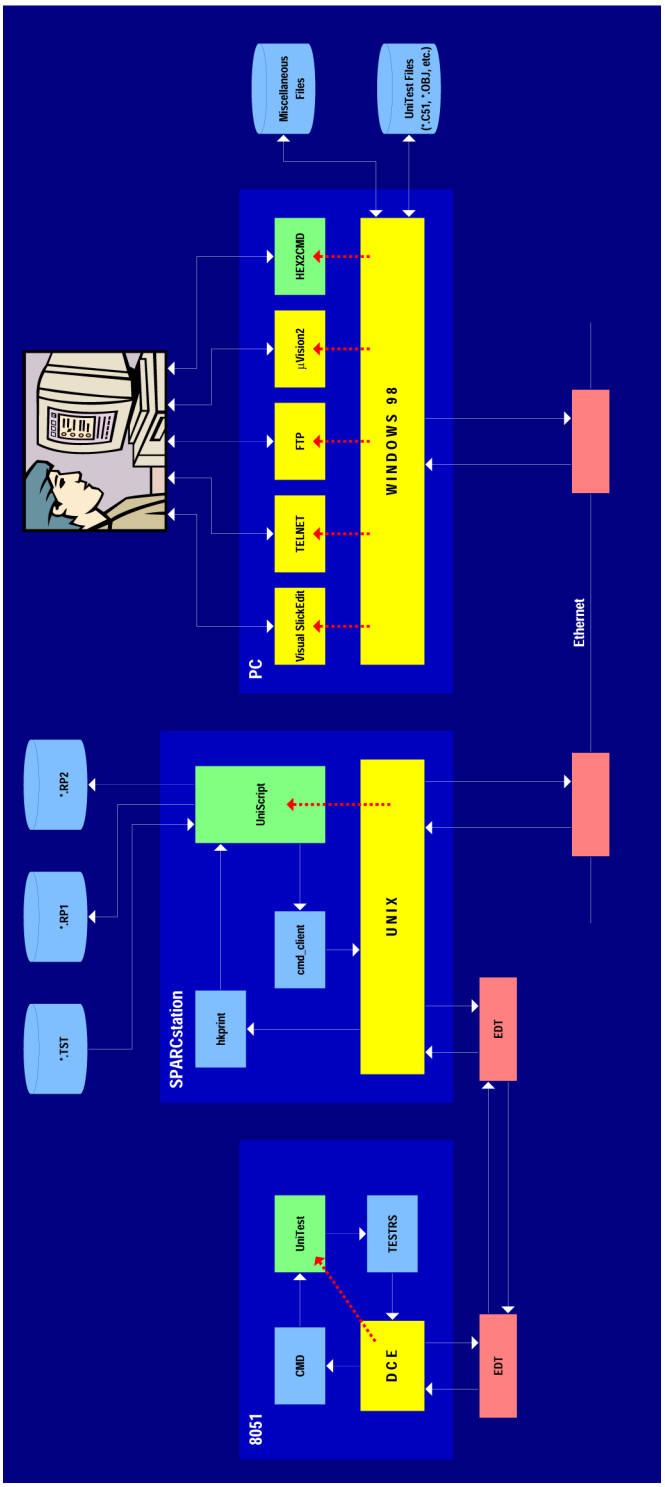


**Figure 2-1: DCE FSW Test Setup**

2.3.2.2 Pre-existing/Commercial Applications

The **hks** test-tool, already developed by D. Blackman of SSL, provides several functions crucial to the logic of the **UniScript** interpreter (section 2.3.2.3).

*2.3.2.2.1 Housekeeping Print (**hkprint**)*

This utility provides the capability of extracting data from the last down-link of housekeeping telemetry or other DCE memory data. The down-linked data include the contents of 8051 RAM locations 0x0040...0x043F, namely the down-link data buffer. This buffer can be populated (by means of the **LFDCOPY** command) with any block of 8051 RAM data (up to $1024 = 4 \times 16^2$ bytes). Hence **hkprint** makes all of DCE memory available to **UniScript**. Input to **hkprint** consists of a character string containing the hex representation of the upper and lower memory bounds of the requested data; output is a hex-formatted string representing the requested data.

*2.3.2.2.2 Command Client (**cmd_client**)*

This utility provides the capability to up-link arbitrary data blocks to the DCE 8051. Its two main uses are to up-link DCE commands and to up-link arbitrary "patches" to DCE code. Input to **cmd_client** is a hex string representing an integral number of 32-bit "d-words". Each d-word consists of a pair of 16-bit words: the first of these contains the absolute RAM target *address* for the second, which is a 16-bit *datum*. **cmd_client** performs no error checking on the input string; any error checking is done within the DCE FSW. The standard procedure for up-linking a block of data is to write it to the memory block starting at 0x0040 (the DCE receive-buffer), then send an **LFDCOPY** command to move the receive-buffer contents to the desired target address.

*2.3.2.2.3 Generate and Fetch a Command Serial Number (**getnextserial**)*

The utility provides the serial number of the next command to be sent (see Section 2.3.8.3.2).

2.3.2.3 Custom Software: UniScript

This program, written in Perl 5, actually "conducts" a software test. It does this by reading a test-script file specified on the Unix command line, then interpreting the lines of this file as comments, DCE FSW commands, or special test directives (such as **LOG**, **DTG**, **WAIT**, and **CHECK**) that are executed within the EGSE (rather than the 8051). **UniScript** can output data to the operator terminal and to report files specified on the command line in response to **WTO, DTG, LOG,** and **FORM** directives. Details of the script language are given in section 2.3.7and 2.3.10.

### 2.3.3 PC System

This computer is used to *prepare* the test-scripts and software; it can also be used to maintain and process (e.g., print) the Report Files. It communicates with the EGSE via a 100 MHz Ethernet link using Telnet or FTP protocols and in this way is able to control execution of **UniScript**, supply the up-linkable image of **UniTest**, and exchange script and report files with the EGSE. The PC also hosts the 8051 IDE and the **HEX2CMD** utility, which transforms the "**.hex**" output of the 8051 linker to a file of text representations of DCE **LFDUPLOD** commands (actually, a Unix shell script) and other sophisticated software, such as word processors and editors that are not supported by the EGSE.

#### 2.3.3.1 Operating System

The PC operates under control of the Windows 98 operating system.

#### 2.3.3.2 Pre-existing/Commercial Applications

##### *2.3.3.2.1 File Transfer (FTP)*

Windows 98 provides a primitive, command-line-oriented facility for transferring files between Windows- and Unix-based file systems. However, several inexpensive FTP programs that provide "GUI" interface to the file-management functionality are downloadable from the Internet.[1]

##### *2.3.3.2.2 Unix Terminal Emulation (Telnet)*

Windows 98 provides, via DOS command, a Telnet window, which can serve as the operator console during software testing.

##### *2.3.3.2.3 8051 Integrated Development Environment (Keil μVision2)*

The IDE contains the tools necessary to convert the source code for the modules comprising the **UniTest** project into either the binary or hexadecimal representations of the absolute executable code. It also provides an 8051 emulator that can be used to test and debug the program. The optimizing **C51** compiler processes a dialectal superset of the ANSI C language; this language takes account of the 8051's intricate storage model. The **A51** assembler processes the standard assembly language for the 8051. Output from either of these language processors (i.e., **.obj** files) is processed by the **L51** linker into absolute (**.exe**) files, or, alternately into Intel **.hex** format.

[1] E.g., BulletProof, NetManager, CuteFTP, FTP Voyager.

*2.3.3.2.4 Editor*

Several editors are currently available on the PC platform, include Visual SlickEdit and the (less capable) editing facility of the IDE (section 2.3.3.2.3)

*2.3.3.2.5 Hex-to-Command Conversion (**HEX2CMD**)*

This application, written by D. Blackman of SSL, transforms the **.hex** output of the **L51** output file into a Unix shell script of **LFDUPLOD** commands (a **.csf** file) embodying the absolute binary text of **UniTest**. Once the **UniTest.csf** file has been "FTPed" from the PC to the EGSE it can be up-linked to the 8051 by merely shelling to it (assuming that the 8051 is alive and that the **hks** test-tool is active).

2.3.4 Custom Software

No custom software for the PC system is required for the software-test implementation described in this document.

2.3.5 Test Report Files

**UniScript** opens one or two files for output (***.rp1** and ***.rp2** in Figure 2-1); these files will receive various kinds of data (determined by the local directives **DTG, ECHO, FORM, LOG, CHECK**) related to the execution of the script. It is not necessary to specify the second file in the Unix command invoking **UniScript**, but it would be meaningless to omit the first. Which of the two report files is to be the recipient of output from the above-mentioned local directives is always specified by an operand, which must be **1** or **2**.

2.3.6 Mnemonics Used in the BNF Productions

Table 2-2 lists the mnemonics of all currently defined DCE commands. Note the addition of the opcode **LFDUTEST** to support FSW testing.

## Table 2-2: DCE Commands

| Mnemonic | Opcode | No. "P" | Boot Mode Only | Description |
|---|---|---|---|---|
| **LFDCOPY** | 83 | 4 | • | . |
| **LFDCRC** | 82 | 3 | • | . |
| LFDDIAGC | F4 | 0 | • | . |
| LFDDNLOD | AE | 2 | • | . |
| LFDGOTO | EA | 1 | • | . |
| LFDHKREQ | FF | 0 | • | . |
| LFDJMPCS | F3 | 1 | • | . |
| LFDMADDR | 81 | 3 | • | . |
| **LFDNOOP** | 80 | 0 | • | . |
| **LFDPROM** | FD | 1 | | |
| **LFDRSTP** | F0 | 0 | • | |
| **LFDRSTW** | F5 | 0 | • | |
| **LFDTEST** | AB | 1 | | |
| **LFDUPLOD** | AD | 3 | • | |
| **LFDWDOG** | F1 | 1 | • | |
| **LFDUTEST** | FC | 5 | | UniTest Command |
| **LFGBWK** | 85 | 3 | | |
| **LFGEWK** | 89 | 3 | | |
| **LFGLQT** | 92 | 2 | | |
| **LFGSHFT** | A4 | 3 | | |
| **LFGSTIM** | 9E | 2 | | |
| **LFGSTR** | A6 | 3 | | |
| **LFGTT** | 97 | 3 | | |
| **LFGUQT** | A2 | 2 | | |
| **LFHQPWR** | CF | 1 | | |
| **LFHRAMPT** | CE | 1 | | |
| **LFHSTATE** | E6 | 1 | | |
| **LFHVENA** | D1 | 1 | | |
| **LFHVILIM** | D0 | 1 | | |
| **LFHVLOW** | B6 | 2 | | |
| **LFHVMAX** | 8D | 2 | | |
| **LFHVNOM** | C6 | 2 | | |
| **LFHVPWR** | C0 | 1 | | |
| **LFHVSET** | CB | 2 | | |
| **LFPCRP** | BA | 3 | | |
| **LFRACT1** | C1 | 1 | | |
| **LFRACT2** | C2 | 1 | | |
| **LFRACTEN** | C3 | 1 | | |
| **LFRACTRS** | C5 | 1 | | |
| **LFRAXPWR** | C4 | 1 | | |
| **LFRILIM** | DC | 1 | | |
| **LFRLSOVD** | 9C | 1 | | |
| **LFRMDIR** | D9 | 1 | | |
| **LFRMENA** | DB | 1 | | |
| **LFRMPWR** | DA | 1 | | |

Table 2-3 lists the mnemonics of all DCE Housekeeping telemetry points:

**Table 2-3: DCE Housekeeping Mnemonics**

| Mnemonic | HK ID | Bits | Description |
|---|---|---|---|
| **LFGPHA** | 1440 | 2048 | PHA Segment A word bins 0..127 |
| **LFGPHB** | 1540 | 2048 | PHA Segment B word bins 0..127 |
| **LFDERR** | 1640 | 64 | HST Errors, 8 error codes |
| **LFDERRP** | 1648 | 128 | HST Errors parameters (8) |
| **LFDCBUF** | 1664 | 224 | Command Buffer Copy |
| **LFCTIME** | 1680 | 32 | DCE Timer |
| **LFCDECA** | 1694 | 32 | Digitized Event Cntr A |
| **LFCDECB** | 1698 | 32 | Digitized Event Cntr B |
| **RFXSVD2** | 169C | 16 | Reserved word |
| **LFCSDC1** | 16A4 | 32 | Science Data Port Ctr 1 |
| **LFCSDC2** | 16A8 | 32 | Science Data Port Ctr 2 |
| **LFCFECA** | 16AC | 32 | Fast Event Cntr A |
| **LFCFECB** | 16B0 | 32 | Fast Event Cntr B |
| **LFGSHFAX** | 16BC | 8 | TDC Image shift AX |
| **LFGSTRAX** | 16BD | 8 | TDC Image stretch AX |
| **LFGSHFBX** | 16BE | 8 | TDC Image shift BX |
| **LFGSTRBX** | 16BF | 8 | TDC Image stretch BX |
| **LFTDVAA** | 16C0 | 8 | Det Vacuum Assy A Temp |
| **LFTACT** | 16C1 | 8 | Actuator temp |
| **LFTHVFM** | 16C2 | 8 | HVFM temp |
| **LFTHVPS** | 16C3 | 8 | HVPS temp |
| **LFTIP** | 16C4 | 8 | Ion Pump temp |
| **LFTDVAB** | 16C5 | 8 | Det Vacuum Assy B Temp |
| **LFTAMPA** | 16C6 | 8 | Amp A temp |
| **LFTAMPB** | 16C7 | 8 | Amp B temp |
| **LFTTDCA** | 16C8 | 8 | TDC A temp |
| **LFTTDCB** | 16C9 | 8 | TDC B temp |
| **LFTLVPC** | 16CA | 8 | LVPC temp |
| **LFTDCE** | 16CB | 8 | DCE temp |
| **LFHFAN** | 16CC | 8 | FHV status analog |
| **LFVP5DC** | 16CD | 8 | DCE-C +5V |
| **RFVP21D** | 16D0 | 8 | DCE +21V (Galex only) |
| **RFRESER0** | 16D1 | 8 | Reserved |
| **LFHIMONA** | 16D2 | 8 | HV Current A |
| **LFHIMONB** | 16D3 | 8 | HV Current B |
| **LFHVMONA** | 16D4 | 8 | HV Output V A |
| **LFHVMONB** | 16D5 | 8 | HV Output V B |
| **LFHQANA** | 16D6 | 8 | QHVA status analog |
| **LFHQANB** | 16D7 | 8 | QHVB status analog |
| **LFVP15TA** | 16D8 | 8 | TDC A +15V |
| **LFVP15TB** | 16D9 | 8 | TDC B +15V |
| **LFVP5TA** | 16DA | 8 | TDC A +5V |
| **LFVP5TB** | 16DB | 8 | TDC B +5V |
| **LFVM15TA** | 16DC | 8 | TDC A -15V |
| **LFVM15TB** | 16DD | 8 | TDC B -15V |
| **LFVM5TA** | 16DE | 8 | TDC A -5V |
| **LFVM5TB** | 16DF | 8 | TDC B -5V |
| **LFVP15D** | 16E0 | 8 | DCE +15V |
| **LFVM15D** | 16E1 | 8 | DCE -15V |
| **LFHVSETA** | 16E2 | 8 | HV Command Level A |
| **LFHVSETB** | 16E3 | 8 | HV Command Level B |
| **LFSBITS2** | 16E4 | 16 | Status bits 2: 8051 Internal Register |

| Mnemonic | HK ID | Bits | Description |
|---|---|---|---|
| **LFPOVDA**[2] | 16E4.00 | 1 | Count Rate Protection Override, Segment A |
| **LFPOVDB** | 16E4.01 | 1 | Count Rate Protection Override, Segment B |
| **RFPA** | 16E4.02 | 1 | CRP Reduction state, Segment A |
| **RFPB** | 16E4.03 | 1 | CRP Reduction state, Segment B |
| **LFHRMPA** | 16E4.06 | 1 | High Voltage Ramp in progress, segment A |
| **LFHRMPB** | 16E4.07 | 1 | High Voltage Ramp in progress, segment B |
| **LFDCODE** | 16E4.09 | 1 | Code Space indicator, (0=Lower, 1=Upper) |
| **LFDWDDIS** | 16E4.14 | 1 | Watchdog disabled |
| **RFRLATCH** | 16E4.15 | 1 | Polling of Door LATCH disabled |
| **LFGUQTA** | 16E6 | 8 | TDC Upper Charge threshold A |
| **LFGUQTB** | 16E7 | 8 | TDC Upper Charge threshold B |
| **LFGBWKAX** | 16E8 | 8 | TDC Begin Walk adjustA |
| **LFGBWKBX** | 16E9 | 8 | TDC Begin Walk adjustB |
| **LFGEWKAX** | 16EA | 8 | TDC End Walk adjust A |
| **LFGEWKBX** | 16EB | 8 | TDC End Walk adjust B |
| **LFGLQTA** | 16EE | 8 | TDC Lower Charge Thresh A |
| **LFGLQTB** | 16EF | 8 | TDC Lower Charge Thresh B |
| **LFGTTAX** | 16F0 | 8 | TDC Timing threshold AX |
| **LFGTTBX** | 16F1 | 8 | TDC Timing threshold BX |
| **LFSBITS0** | 16F2 | 16 | Status bits 0: 8051 Internal Register |
| **LFHVENA** | 16F2.06 | 1 | HVPS SW Enable status |
| **LFRMENA** | 16F2.13 | 1 | Door Motor SW Enable status |
| **LFSBITS1** | 16F4 | 16 | Status bits 1: 8051 Internal Register |
| **RFD1HZ** | 16F4.00 | 1 | 1Hz Flag (internal use) |
| **LFDOPERT** | 16F4.03 | 1 | Boot =0, Operate=1. DCEÕs assertion that it is running code in boot mode or operate mode, |
| **RFDPROJT** | 16F4.04 | 1 | COS=0, GALEX=1. DCEÕs assertion that it is running code specific to one project or another, namely |
| **RFDDETID** | 16F4.14 | 1 | Detector ID bit |
| **LFMRAM** | 16FA | 16 | CRC RAM |
| **LFMROM** | 16FC | 16 | CRC ROM |
| **LFDSWVER** | 16FE | 16 | DCE FSW version |
| **LFCPKT** | 1700 | 32 | DCE Packet Cntr |
| **LFDLSTC** | 1704 | 8 | DCE Last Cmd Opcode executed |
| **LFHSTATE** | 1705 | 3 | HV State Bits: 0 -HV Off, 7 HV On, 1 NomA, 2 NomB, 3 Nom, 4 Low, 5 Undefined, 6 Set |
| **LFGREFA** | 1706 | 8 | Digitizer V ref A |
| **LFGREFB** | 1707 | 8 | Digitizer V ref B |
| **LFDDIAG0** | 1708 | 8 | DCE diagnostic code 0, (most recent) |
| **LFDDIAG1** | 1709 | 8 | DCE diagnostic code 1 |
| **LFRILIM** | 170A | 8 | Door stall current setting |
| **LFHVILIM** | 170B | 8 | HV Current limit setting |
| **LFDCMDX** | 170C | 16 | DCE Cmds Executed Counter |
| **LFPCNTA** | 170E | 16 | CRP cnt thresh A |
| **LFVPMON** | 1710 | 8 | LVPC Power |
| **LFDDIAG2** | 1711 | 8 | DCE diagnostic code 2 |
| **LFRACTAN** | 1714 | 8 | Actuator status analog |
| **LFRMTRAN** | 1715 | 8 | Door motor analog monitor |
| **LFRAXI** | 1716 | 8 | Auxiliary power supply current<br>Renamed from LFVAUXI to group together with other AUX mnemonics. |
| **LFRAXAN** | 1717 | 8 | Aux status analog |
| **LFDCMDR** | 1718 | 16 | DCE Cmds Received Counter |

[2] Mnemonics representing bits correspond, in **UniScript**, to bit-masks. The setting of a bit may thus be determined by forming the logical AND ("conjunction" — Perl bit-operator `&`) of the mask with the containing string. For example, the value of `$LFPOVDA == 1` if and only if `$LFPOVDA & $LFSBITS2 != 0`.

| Mnemonic | HK ID | Bits | Description |
|---|---|---|---|
| **LFMCRC** | 171A | 16 | CRC Transfer Memory, UL/DL, or calc result |
| **LFPINTA** | 171C | 16 | CRP int time A |
| **LFPCNTB** | 171E | 16 | CRP cnt thresh B |
| **LFHVLOA** | 1720 | 8 | HV level for HVLOW, Segment A |
| **LFHVLOB** | 1721 | 8 | HV level for HVLOW, Segment B |
| **LFHVNOMA** | 1724 | 8 | HV Nominal setting A |
| **LFHVNOMB** | 1725 | 8 | HV Nominal setting B |
| **LFHVMAXA** | 1726 | 8 | HV Command level maximum A |
| **LFHVMAXB** | 1727 | 8 | HV Command level maximum B |
| **LFPINTB** | 1728 | 16 | CRP int time B |
| **LFSBITS3** | 172A | 16 | Status bits 3: Bi-Level Mux H/W talkback |
| **LFRLA** | 172A.00 | 1 | Door latch status |
| **LFROP** | 172A.01 | 1 | Door open status |
| **LFRCL** | 172A.02 | 1 | Door closed status |
| **RFHVENA1** | 172A.04 | 1 | HVPS Enable Status Bit 1 -- from safe plug |
| **LFHVPWR** | 172A.04 | 1 | HVPS Power Status Bit |
| **LFHVQPWR** | 172A.05 | 1 | HV Grid Status Bit |
| **LFHPLUG** | 172A.06 | 1 | HV Safe plug presence (0=plug, 1=no plug) |
| **LFDETID0** | 172A.07 | 1 | Detector ID bit 0 |
| **LFRAXPWR** | 172A.08 | 1 | Aux power status bit |
| **LFRACTEN** | 172A.09 | 1 | Actuator enable |
| **RFRACTEN** | 172A.09 | 1 | Actuator enable - (talkback from ctl register) |
| **LFRMPWR** | 172A.10 | 1 | Door motor power |
| **RFHV0** | 172A.13 | 1 | HV0 u27 pin 13 DCE-C |
| **LFRPLUG** | 172A.14 | 1 | Door Safe plug presence (0=plug, 1=no plug) |
| **LFDETID1** | 172A.15 | 1 | Detector ID bit1 |
| **LFSBITS4** | 172C | 8 | Status bits 4: Controls 1 (Locked) H/W talkback<br>Only lower 8-bits valid |
| **RFDCTRL0** | 172C.00 | 1 | Aux Pwr Ena -- Talkback from the protected control register -- |
| **LFRACT1** | 172C.01 | 1 | Actuator Heater 1 On -- Talkback from the protected control register -- |
| **LFRACT2** | 172C.02 | 1 | Actuator Heater 2 On -- Talkback from the protected control register -- |
| **RFDCTRL3** | 172C.03 | 1 | N/C -- Talkback from the protected control register -- |
| **RFDCTRL4** | 172C.04 | 1 | Actuator Ena -- Talkback from the protected control register -- |
| **RFDCTRL5** | 172C.05 | 1 | V21 Spare N/C -- Talkback from the protected control register -- |
| **RFDCTRL6** | 172C.06 | 1 | V22 Spare N/C -- Talkback from the protected control register -- |
| **LFRLSOVD** | 172C.07 | 1 | Door endswitch Override -- Talkback from the protected control register -- |
| **LFSBITS5** | 172D | 8 | Status bits 5: Controls 2 (Unlocked) H/W talkback<br>Only lower 8-bits valid |
| **RFDC2RL0** | 172D.00 | 1 | HV Ena 1 -- Talkback from the safe plug -- |
| **RFDC2RL1** | 172D.01 | 1 | HV Ena 2 -- Talkback from the safe plug -- |
| **RFDC2RL2** | 172D.02 | 1 | Aux Pwr Ena -- Talkback from the arming plug -- |
| **LFRMDIR** | 172D.03 | 2 | Motor Direction (0=Safe, 1=Close, 2=Open, 3 is ignored) |
| **RFDC2RL5** | 172D.05 | 1 | Motor On -- Talkback from the arming plug -- |
| **LFDPROM** | 172D.06 | 1 | PROM Enable |
| **RFDPROM2** | 172D.07 | 1 | PROM 2 Enable |
| **LFGSTIMA** | 172E | 8 | Record of STIM setting to digitizer A |
| **LFGSTIMB** | 172F | 8 | Record of STIM setting to digitizer B |
| **LFDPERF** | 1730 | 16 | DCE Performance monitor. Executive loop counter. |
| **RFDPEEK** | 173³4 | 8 | DCE Memory Peek value |
| **LFRPOS** | 1735 | 8 | Door position 0..255 |
| **LFDMEXTR** | 1736 | 8 | Memory monitor external/internal select bits |
| **LFDMDATA** | 1737 | 8 | Memory monitor code/data select bits |
| **LFDMONS** | 1738 | 64 | 8 MEB Monitor Values |
| **LFDMADD** | 1740 | 128 | 8 MEB Monitor Address Table |
| **LFGBWKAY** | 1750 | 8 | TDC Begin Walk adjustA |
| **LFGEWKAY** | 1751 | 8 | TDC End Walk adjust A |
| **LFGBWKBY** | 1752 | 8 | TDC Begin Walk adjustB |

| Mnemonic | HK ID | Bits | Description |
|---|---|---|---|
| **LFGEWKBY** | 1753 | 8 | TDC End Walk adjust B |
| **LFRTIMER** | 1754 | 8 | Door timeout timer (countdown) |
| **LFGTTAY** | 175E | 8 | TDC Timing threshold AY |
| **LFGTTBY** | 175F | 8 | TDC Timing threshold BY |
| **LFGSHFAY** | 1760 | 8 | TDC Image shift AY |
| **LFGSTRAY** | 1761 | 8 | TDC Image stretch AY |
| **LFGSHFBY** | 1762 | 8 | TDC Image shift BY |
| **LFGSTRBY** | 1763 | 8 | TDC Image stretch BY |
| **RFXSVD1** | 1764 | 16 | Reserved word |
| **LFHRAMPT** | 1774 | 8 | HV Ramp time |
| **LFDDIAG3** | 1776 | 8 | DCE diagnostic code 3 |
| **LFDDIAG4** | 1777 | 8 | DCE diagnostic code 4 |
| **LFMU7** | 1778 | 16 | CRC U7 |
| **LFMU2** | 177A | 16 | CRC U2 |
| **LFMLWR** | 177C | 16 | CRC Lower |
| **LFMUPR** | 177E | 16 | CRC Upper |
| **LFDDIAGS** | 1780 | 512 | DCE diag buffer, 32 words |
| **LFVP5DA** | 17C0 | 8 | DCE-A +5V |
| **LFVP5DB** | 17C1 | 8 | DCE-B +5V |
| **LFHVTGTA** | 17D2 | 8 | HV Target Level A |
| **LFHVTGTB** | 17D3 | 8 | HV Target Level B |

Table 2-4 lists the standard special function registers (SFRs) and bit mnemonics used in 8051 assembler code:

**Table 2-4: Intel 8051 Special Function Registers**

| Register | Direct Address | Bit Address | DCE Variable? | Description |
|---|---|---|---|---|
| **A** | E0 | | • | Accumulator[3] |
| **B** | F0 | | • | Accumulator Extension[3] |
| **DPTR** | 83 | | • | Data Pointer (2 bytes) |
| **DPH** | 83 | | • | Data Pointer High |
| **DPL** | 82 | | • | Data Pointer Low |
| **IE** | A8 | | • | Interrupt Enable |
| **EX0** | | A8 | • | External Interrupt 0 Enable |
| **ET0** | | A9 | • | Timer 0 Overflow Interrupt Enable |
| **EX1** | | AA | • | External Interrupt 1 Enable |
| **ET1** | | AB | • | Timer 1 Overflow Interrupt Enable |
| **ES** | | AC | • | Serial Port Interrupt Enable |
| | | | | Reserved |
| | | | | Reserved |
| **EA** | | AF | • | Enable All Interrupts |
| **IP** | B8 | | • | Interrupt Priority |
| **PX0** | | B8 | • | External Interrupt 0 Priority |
| **PT0** | | B9 | • | Timer 0 Interrupt Priority |
| **PX1** | | BA | • | External Interrupt 1 Priority |
| **PT1** | | BB | • | Timer 1 Interrupt Priority |
| **PS** | | BC | • | Serial Port Interrupt Priority |
| | | | | Reserved |
| | | | | Reserved |
| | | | | Reserved |

[3] Bits in this register are individually addressable, but have not been assigned individual names.

| Register | Direct Address | Bit Address | DCE Variable? | Description |
|---|---|---|---|---|
| **P0** | 80 | | • | Port 0 |
| **AD0** | | 80 | | |
| **AD1** | | 81 | | |
| **AD2** | | 82 | | |
| **AD3** | | 83 | | |
| **AD4** | | 84 | | |
| **AD5** | | 85 | | |
| **AD6** | | 86 | | |
| **AD7** | | 87 | | |
| **P1** | 90 | | • | Port 1 |
| **T2** | | 90 | | |
| **T2EX** | | 91 | | |
| | | 92 | | |
| | | 93 | | |
| | | 94 | | |
| | | 95 | | |
| | | 96 | | |
| | | 97 | | |
| **P2** | A0 | | • | Port 2 |
| **A8** | | A0 | | |
| **A9** | | A1 | | |
| **A10** | | A2 | | |
| **A11** | | A3 | | |
| **A12** | | A4 | | |
| **A13** | | A5 | | |
| **A14** | | A6 | | |
| **A15** | | A7 | | |
| **P3** | B0 | | • | Port 3 |
| **RxD** | | B0 | | |
| **TxD** | | B1 | | |
| **INT0** | | B2 | | |
| **INT1** | | B3 | | |
| **T0** | | B4 | | |
| **T1** | | B5 | | |
| **WR** | | B6 | | |
| **RD** | | B7 | | |
| **PCON** | 87 | | • | Power Control[4] |
| **IDL** | | | | Idle Mode Bit |
| **PD** | | | | Power Down Bit |
| **GF0** | | | • | General-Purpose Flag |
| **GF1** | | | • | General-Purpose Flag |
| | | | | Reserved |
| | | | | Reserved |
| | | | | Reserved |
| **SMOD** | | | • | Double Baud Rate |
| **PSW** | D0 | | • | Program Status Word |
| **P** | | D0 | • | Parity |
| | | | | User-Definable Flag |
| **OV** | | D2 | • | Overflow |
| **RS0** | | D3 | • | Register Bank Select Control Bit 0 |
| **RS1** | | D4 | • | Register Bank Select Control Bit 1 |
| **F0** | | D5 | • | Flag 0 |
| **AC** | | D6 | • | Auxilliary Carry Flag |
| **CY** | | D7 | • | Carry Flag |
| **SBUF** | 99 | | • | Serial Data Buffer |
| **SCON** | 98 | | • | Serial Control |

| Register | Direct Address | Bit Address | DCE Variable? | Description |
|---|---|---|---|---|
| **RI** | | 98 | | Receive Interrupt Flag |
| **TI** | | 99 | | Transmit Interrupt Flag |
| **RB8** | | 9A | | 9th data bit received |
| **TB8** | | 9B | | 9th data bit sent |
| **REN** | | 9C | | Enable Serial Reception |
| **SM2** | | 9D | | Enable Multiprocessor Communication |
| **SM1** | | 9E | | Serial Port Mode |
| **SM0** | | 9F | | Serial Port Mode |
| **SP** | 81 | | • | Stack Pointer |
| **TCON** | 88 | | • | Timer Control |
| **IT0** | | 88 | • | Interrupt 0 Type Control Bit |
| **IE0** | | 89 | • | Interrupt 0 Edge Flag |
| **IT1** | | 8A | • | Interrupt 1 Type Control Bit |
| **IE1** | | 8B | • | Interrupt 1 Edge Flag |
| **TR0** | | 8C | • | Timer 0 Run Control Bit |
| **TF0** | | 8D | • | Timer 0 Overflow Flag |
| **TR1** | | 8E | • | Timer 1 Run Control Flag |
| **TF1** | | 8F | • | Timer 1 Overflow Flag |
| **TH0** | 8C | | • | Timer High 0 |
| **TH1** | 8D | | • | Timer High 1 |
| **TL0** | 8A | | • | Timer Low 0 |
| **TL1** | 8B | | • | Timer Low 1 |
| **TMOD** | 89 | | • | Timer Mode Control[4] |
| **M0** | | | | Timer 0 |
| **M1** | | | | Timer 0 |
| **C/T** | | | | Timer 0 |
| **GATE** | | | | Timer 0 |
| **M0** | | | | Timer 1 |
| **M1** | | | | Timer 1 |
| **C/T** | | | | Timer 1 |
| **GATE** | | | | Timer 1 |

There are also four banks of "general registers" in the 8051 and each bank contains eight registers. The symbols **R0**, **R1**, **R2**, **R3**, **R4**, **R5**, **R6**, and **R7** are used for these registers; the ambiguity of their RAM addresses is resolved by the 8051 during execution (there being, for example, actually *four* **R0**s because of the four Register Banks) by bits **RS1** and **RS0** of the **PSW**.

[4] Bits in this register are not individually addressable.

**Table 2-5: Intel 8051 General Register Addresses**

| Register Nomenclature | Bits RS1 and RS0 | | | |
|---|---|---|---|---|
| | 00 | 01 | 10 | 11 |
| **R0** | 0x00 | 0x08 | 0x10 | 0x18 |
| **R1** | 0x01 | 0x09 | 0x11 | 0x19 |
| **R2** | 0x02 | 0x0A | 0x12 | 0x1A |
| **R3** | 0x03 | 0x0B | 0x13 | 0x1B |
| **R4** | 0x04 | 0x0C | 0x14 | 0x1C |
| **R5** | 0x05 | 0x0D | 0x15 | 0x1D |
| **R6** | 0x06 | 0x0E | 0x16 | 0x1E |
| **R7** | 0x07 | 0x0F | 0x17 | 0x1F |

### 2.3.7 Elements of the Script Language

The following is an informal description of the test-script language interpreted by **UniScript** using an eclectic metalanguage borrowed from BNF and "regular expressions". Section 2.3.6 supplies certain syntactic constants used in the productions.

A DCE FSW test-script consists of *statements*; the statements are processed by **UniScript** in the order in which they are read from the script file (that is, the statements are processed sequentially, with no "loops"). These statements (apart from comments) represent actions to be taken by **UniScript**. Such actions fall into two classes: things to be done *locally* (i.e., by **UniScript** itself); and things to be done *remotely* (i.e., by FSW or by **UniTest** acting as an extension of FSW). Local actions are, of course, performed by Perl code internal to **UniScript** (perhaps with help from **hks**). Remote actions are initiated by means of script statements encoding DCE commands. These DCE-command-statements are processed by **UniScript** into hexadecimal strings which are then up-linked, via the **hks** tool **cmd_client** (section 2.3.2.2.2), to the DCE 8051, where they are processed by the FSW command task. Commands from the normal repertoire (see Table 2-2) are executed by FSW code; the special **LFDUTEST** command (opcode 0xFC) is handed off by the FSW command task to the **UniTest** routine.

The script language is a kind of toy programming language. The statement types are the *local* actions (**DTG, WTO, DIAG, DELAY, WAIT, ECHO, FORM, LOG, WTOR, DATA, CHECK, SYM**), the *remote* actions (DCE FSW commands, Table 2-2), and the *comment* statement (which is useful mainly to annotate parts of a script).

A semi-formal description of the script language syntax is given in 2.3.10. **UniScript** does not attempt to recover from erroneous syntax in a script statement; it produces an error message and terminates execution in such cases. That is, a successful software test requires a syntactically "perfect" script.

#### 2.3.7.1 Comments

Any line of the script file beginning with the semi-colon "**;**" is ignored by **UniScript** except to copy it (as with all other script statements as they are encountered) to the operator console.

#### 2.3.7.2 Variables and Constants

Script statement can contain numeric or character-string parameters. These parameters may be either constants or, in some cases, variables.

##### *2.3.7.2.1 Character Strings*

A character string occurs in **WTO**, **WTOR**, and **DTG** directives. This datum contains arbitrary text that *does not include the double-quote*, enclosed in double-quotes.

##### *2.3.7.2.2 Numeric Constants*

Numeric constants may be represented as either (positive) decimal numerals or as one- or two-byte hexadecimal constants in "C-language syntax" — a sequence of two or four hex digits preceded by **0x**. Numeric constants may also be strung together, separated by commas, to form a list. In addition, a long string containing a multiple of four hex digits, and interspersed with underscore characters, may be used to represent a list of 16-bit words:

**0x028A_D7EE_3C61_0DE4_4B71 = 0x028A, 0xD7EE, 0x3C61, 0x0DE4, 0x4B71**

##### *2.3.7.2.3 **UniScript** Variables*

There are certain pre-defined variables used by **UniScript** but accessible by test-script statements. These are summarized in.

**Table 2-6: UniScript Built-In Variables**

| UniScript Variable | Explanation |
|---|---|
| **$B1** | Utility Buffer 1; implicitly used in **LOG, DATA**, **XMIT**, and **RECV** local directives; can be referenced as Perl string in **CHECK** local directives |
| **$B2** | Utility Buffer 2; implicitly used in **LOG, DATA**, **XMIT**, and **RECV** local directives; can be referenced as Perl string in **CHECK** local directives |
| **$L1** | Length of Buffer 1, computed automatically whenever **$B1** is re-populated (i.e., by **DATA** or **RECV**); can be referenced as Perl numeric in **CHECK** local directives |

| UniScript Variable | Explanation |
|---|---|
| **$L2** | Length of Buffer 2, computed automatically whenever **$B2** is re-populated (i.e., by **DATA** or **RECV**); can be referenced as Perl numeric in **CHECK** local directives |
| **$CRC1** | CRC for Buffer 1; computed automatically whenever **$B1** is re-populated; can be referenced as Perl numeric in **CHECK** local directives |
| **$CRC2** | CRC for Buffer 2; computed automatically whenever **$B2** is re-populated; can be referenced as Perl numeric in **CHECK** local directives |
| **$SEQNO** | Recent history of command sequence numbers; this is a Perl array and can be referenced as such in **CHECK** local directives. It is set so that **$SEQNO**[0] = most recently generated number, **$SEQNO**[1] next most recently, etc. |

*2.3.7.2.4 Housekeeping Variables*

The housekeeping mnemonics are defined in Table 2-3, column 1. Each housekeeping variable is pre-defined in the **UniScript** program as a variable of the appropriate type. These may be employed, for example, in **CHECK** local directives to provide automated verification of test results. In such uses the identifier listed in Table 2-3 must, because of the syntax expected by the Perl interpreter, be preceded by the "de-referencer" '**$**'. For example, the elements of the memory monitor array **LFDMONS** must be referenced in a **CHECK** local directive as **$LFDMONS[0]**, **$LFDMONS[1]**,....

Housekeeping variables receive new (refreshed) values when (and only when) the **LOG** local directive (section 2.3.8.3.9) is executed by the script.

*2.3.7.2.5 DCE Variables*

The script language provides the capability of defining variable names to be associated with RAM or ROM locations in the DCE memory. (These variables have no necessary or intrinsic connection with variable names actually used in DCE code.) Such names are useful in the context of DCE commands. Many such names, representing 8051 registers, are pre-defined (see Table 2-4 and the following paragraph). DCE variables are represented inside the **UniScript** interpreter as entries of a "hash" table whose keys are the variable names assigned (with some exceptions) by means of the **SYM** local directive (section 2.3.8.3.11). The name of this hash table in the Perl code is **%SYM**. **%SYM** associates with each key a data structure that includes the value, storage type (if appropriate), and contents. For example, the *value* of **A** is 0xE0, its *contents* are assigned by **SET A=**(byte value), and its *storage type* is "SFR".

DCE variables may be used in **CHECK** local directives, but the appropriate syntax must be used, namely reference to an element of **%SYM**: **$SYM{"XYZ"}**, for example, if **XYZ** is a DCE variable,

| Script File | Comments |
|---|---|
| **SYM XYZ=X:0xFF00, NBYTES=28** | Define RAM address **XYZ** and block length **NBYTES** for use in **LFDDNLOD.** (This is *not* a DCE command.) |
| **LFDDNLOD XYZ, NBYTES** | Down-link the block. (This *is* a DCE command using symbolic arguments defined by the **SYM** local directive.) |
| **RECV 1, NBYTES** | Get the down-linked data into **$B1** |
| **CHECK 1, $SYM{"XYZ"}==$LFMU7** | Get Perl to compare **$LFMU7** with **XYZ** |

There are pre-defined DCE variables; these exceptions to the rule that the values and contents of DCE variables are assigned by the **SYM** and **REG** local directives are summarized in Table 2-7.

**Table 2-7: DCE Variables Whose Values are Automatically Assigned**

| DCE Variable | Explanation |
|---|---|
| **L1** | Current length of Buffer 1 |
| **L2** | Current length of Buffer 2 |
| **CRC1** | CRC for Buffer 1; computed whenever **$B1** is re-populated (its value is set whenever **$CRC1** is set); can be referenced in DCE commands |
| **CRC2** | CRC for Buffer 2; computed whenever **$B2** is re-populated (its value is set whenever **$CRC2** is set); can be referenced in DCE commands |

### 2.3.8 Script Statements

The following sections enumerate the various local and remote script statements and discuss their semantics — "what they're good for."

#### 2.3.8.1 DCE Commands

The uses of the DCE commands are documented elsewhere. In the script language a DCE command appears as a command mnemonic followed by expressions representing the (maximum of) five permitted argument values. The arguments (whether symbolic or literal) supplied in a DCE command must agree in number with the value specified by DM-05 (see Table 2-2 for the mnemonics and numbers of parameters).

In the following example, two DCE-related symbols (section 2.3.7.2.5) are defined as representing a block of RAM of a certain length. They can then be used as symbolic arguments in a DCE command.

| Script File | Comments |
|---|---|
| **SYM XYZ=X:0xFF00, NBYTES=28** | Define RAM address **XYZ** and block length **NBYTES** for use in **LFDDNLOD.** (This is *not* a DCE command.) |
| **LFDDNLOD XYZ, NBYTES** | Down-link the block. (This *is* a DCE command using symbolic arguments defined by the **SYM** local directive.) |
| **RECV 1, NBYTES** | Complete the down-linking to **$B1** |

### 2.3.8.2 UniTest Commands

The special opcode 0xFC is temporarily reserved for a command performing several functions useful in software testing of FSW, particularly unit testing. The mnemonic for this command will be **LFDUTEST**. The structure of the command packet for **LFDUTEST** is as follows (ignoring the complemented words)**:**

| Opcode | | Sequence Number | | Parameter 0 (subcommand) | | Parameter 1 | | Parameter 2 | | Parameter 3 | | Parameter 4 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0440** | **FCFC** | **0444** | | **0448** | | **044C** | | **0450** | | **0454** | | **0458** | |

**Figure 2-2: Structure of LFDUTEST Command Packet**

The "Parameter 0" slot in the packet is given over to a code representing the intended function, as follows:

| Parameter 0 Value | LFDUTEST Subcommand |
|---|---|
| 0x00 | No-Operation |
| 0x01 | Prepare Registers (**REG** subcommand) |
| 0x02 | Prepare RAM (**SET** subcommand) |
| 0x03 | Clear RAM Tables (**CLEAR**) |
| 0x04 | Transfer Control (**CALL**) |
| 0x05 | Transfer Control (**JUMP**) |

#### *2.3.8.2.1 REG — Prepare Registers Subcommand*

The Prepare Registers subcommand permits the tester to specify the values of up to four 8051 registers as part of the environment for the next invocation of a DCE FSW module. The **UniTest** subcommand is formatted as shown in Figure 2-3. (More than one Prepare Registers command may be issued, so any number of 8051 register setups can be scheduled before a transfer-of-control subcommand.) In the example illustrated in Figure 2-3, register **r** is assigned the 8-bit value **v**, and the register pair $\mathbf{r}_1$:$\mathbf{r}_2$ is assigned the 16-bit value whose MSB is $\mathbf{v}_1$ and whose LSB is $\mathbf{v}_2$ — i.e., $\mathbf{r}_1 = \mathbf{v}_1$, $\mathbf{r}_2 = \mathbf{v}_2$. The script statement specifying the command would be

| Script File | Comments |
|---|---|
| **LFDUTEST REG, r=v, r1:r2=V** | Load **r** with **v**; **r1**:**r2** with 256×**v1** + **v2** |

The codes used in the command parameters are taken from Table 2-4: e.g., the code for **A** is 0xE0.

| Opcode | | Sequence Number | | Parameter 0 (subcommand) | | Parameter 1 | | | Parameter 2 | | | Parameter 3 | | | Parameter 4 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0440** | **FCFC** | **0444** | | **0448** | 0x01 | **044C** | 0 | r | **0450** | | V | **0454** | $r_1$ | $r_2$ | **0458** | $v_1$ | $v_2$ |

**Figure 2-3: Format of Example LFDUTEST REG Subcommand**

A value may be assigned to a register (8 bits, except for **DPTR**), a bit (Table 2-4), or a register pair (16 bits). In the latter case a pair of register symbols separated by a colon (":") denotes the register pair; during execution of the register set-up the first register of the specified pair will receive the MSB and the second, the LSB. The setup is actually executed immediately prior to a transfer of control (0). The contents of registers to be modified are retained by **UniTest** in a special "save area"; after — or if! — the transferred-to code returns control to **UniTest** the original contents of the modified registers will be restored (i.e., immediately prior to **UniTest**'s return of control to the command task).

As an example, the prologue to the DCE **BYTE_MOVE** routine explains the calling environment as follows:

```
; R0 holds number of bytes to move
; R3:R2 holds pointer to source
; R5:R4 holds pointer to dest
; F0 selects CRC instead of actual move
```

A unit test of **BYTE_MOVE** could be set up as shown in Figure 2-4.

| Script File | Comments |
|---|---|
| **; Move 6 bytes from 0F00h to 0F0Eh** | Comment |
| **SYM INTERNAL=X:0x9800, BYTE_MOVE=INTERNAL+0x041F** | Define RAM address of **BYTE_MOVE** routine relative to **INTERNAL** segment |
| **SYM ADDR1=X:0x0F00, ADDR2=X:0x0F0E** | Define RAM address of source, target regions |
| **DATA 1,0,6,RAND=0x0053** | Put 6 random bytes in **UniScript $B1** |
| **LFDUPLOD ADDR1,6,CRC1** | Start up-link of 6 bytes in **$B1** |
| **XMIT 1,0,6** | Up-link the test data |
| **REG R0=6, F0=0** | Load R0 with length of string; select "move" rather than CRC |
| **REG R3:R2=ADDR1, R5:R4=ADDR2** | Set up source and destination addresses; **R3**=0x0F, **R2**=0x00, etc. |
| **CALL BYTE_MOVE** | Invoke the unit, restore registers |
| **LFDDNLOD ADDR2,6** | Down-link the target region |
| **RECV 2,0,6** | Put down-linked data in **UniScript $B2** |

| Script File | Comments |
|---|---|
| **CHECK 1,substr($B1,0,6)==substr($B2,0,6)** | Does source = dest? Put output in Report File 1. |

**Figure 2-4: Preparing for and Executing the BYTE_MOVE Unit**

*2.3.8.2.2 SET — Prepare RAM*

The Prepare RAM subcommand permits the tester to specify the values of one or two bytes of 8051 RAM as part of the environment for the next invocation of a DCE FSW module. The **UniTest** subcommand is formatted as shown in Figure 2-5. (More than one Prepare RAM command may be issued, so any number of 8051 RAM setups — up to a maximum of thirty-two[5] can be scheduled before a control-transfer subcommand.)

**Figure 2-5: Formats of SET and CLEAR Subcommands**

| Opcode | | Sequence Number | | Parameter 0 (subcommand) | | Parameter 1 | | Parameter 2 | | Parameter 3 | | | Parameter 4 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0440** | **FCFC** | **0444** | | **0448** | 0x02 | **044C** | RAM Addr | **0450** | RAM Type | **0454** | 0/1 | val | **0458** | 0/1 | Val |

| Opcode | | Sequence Number | | Parameter 0 (subcommand) | | Parameter 1 | | Parameter 2 | | Parameter 3 | | Parameter 4 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0440** | **FCFC** | **0444** | | **0448** | 0x03 | **044C** | 00 | **0450** | 00 | **0454** | 00 | **0458** | 00 |

The coding of Parameter 2 of the **SET** command is given by

Table 2-8. Parameters 3 and 4 contain the one or two bytes of the assigned value in their LSBs; the MSBs contain 0x01 if the corresponding LSB is regarded as data. Suppose the following script statements occur:

| Script File | Comments |
|---|---|
| **SYM ADDR1=X:0x0F00** | Define RAM address |
| **SET ADDR1=0x07** | Put one byte in ADDR1 |

The command generated for the **SET** would be:

| Opcode | | Sequence Number | | Parameter 0 (subcommand) | | Parameter 1 | | Parameter 2 | | Parameter 3 | | | Parameter 4 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0440** | FCFC | **0444** | | **0448** | 0x0002 | **044C** | 0x0F00 | **0450** | 0x0002 | **0454** | 0x01 | 0x07 | **0458** | 0x00 | |

**Table 2-8: RAM-Type Encoding for SET Subcommand**

| Parameter 2 | RAM Type |
|---|---|

[5] However, the number 32 is encoded as a pre-compile variable which can be changed at any time; the intent is to provide an adequate number of RAM-address slots to accommodate Unit Testing, but not an excessive, unnecessary number. The RAM-address slots consume five bytes of 8051 storage each.

| | |
|---|---|
| 0x00 | **D: internal data** (on-chip RAM: 0x00-0x7F) |
| 0x01 | **B: bit data** (on-chip RAM 0x20-0x2F) |
| 0x02 | **X: external data** (0x0000-0xFFFF) |

*2.3.8.2.3 CLEAR —Clear RAM Tables*

This subcommand clears out the tables describing the RAM set-up actions performed by **UniTest** (section 2.3.8.2.2). This is an essential operation to prevent table-full conditions and to "clean the slate" between software tests, since once **UniTest** has been up-linked it is left in 8051 RAM to respond to many test-scripts.

*2.3.8.2.4 CALL or JUMP — Transfer Control*

This subcommand causes control of the 8051 to be transferred to the ROM or RAM address specified in Parameters 1 and 2. Parameter 2 specifies the memory type (Table 2-10). This subcommand exists in order to facilitate Unit Testing. Immediately before executing the **CALL** or **JUMP** subcommand, **UniTest** sets 8051 registers and RAM locations per any previously executed **REG** or **SET** subcommands. The difference between **CALL** and **JUMP** is simply that, in the case of **CALL**, the stack receives the return address (within **UniTest**), whereas for **JUMP** it does not.

**Table 2-9: Memory-Type Encoding for CALL/JUMP Subcommand**

| Parameter 2 | RAM Type |
|---|---|
| 0x02 | **X: external RAM** (0x8000-0xFFFF) |
| 0x03 | **C: code PROM** (0x0000-0x3FFF) |

2.3.8.3 Local Directives

*2.3.8.3.1 CHECK — Pass/Fail per Boolean Expression*

The **CHECK** local directive is an automated means by which it can be decided whether a software test (controlled by a script) "passes" or "fails". It requires two operands. The first, which connotes the Report File, must be either **1** or **2**. The second is an expression involving any desired variables of the Perl **UniScript** program. Normally these would be the "Housekeeping" and "UniScript" variables (see sections 2.3.7.2.3 and 2.3.7.2.4). Since **CHECK** is actually executed by means of the Perl **eval** construct, the second operand *must* be a valid Perl expression which, moreover, evaluates to a Boolean value (i.e., could be used between the parentheses in a Perl **if ()** {...} context). Variables used in the expression must be defined in the **UniScript.pl** program.

**Figure 2-6: Example Output of CHECK Local Directive**

```
CHECK:  (($LFDCMDX & $LFDOPERT) == 0)
eval:   ((ffff & 0008) == 0)
F A I L U R E
```

*2.3.8.3.2 DATA — Generate Test Data*

It is essential to be able to generate, or specify, in a test-script, data to drive an FSW module or capability. Three schemes for doing this are

- specification of *null* data (to reset = clear a buffer)
- specification of *constant* data value(s)
- specification of data *pattern*
- specification of the *next* command sequence number
- specification of *random* data generation.

The **DATA** local directive permits any of these schemes to be chosen. The four parameters for **DATA** are,

1. the buffer (**1** or **2**, representing **$B1** or **$B2**);
2. the offset within the selected buffer at which data deposition is to start (beginning with 0);
3. the length of the generated data string; and
4. the data generation scheme (**RAND**, **CLEAR**, **RAMP**, **NEXT**, or **CONST**).

*The data deposited in the buffer consist of hexadecimal constants*; however, references to lengths and offsets of the data must be for the *binary* constants thus represented. The reason for this is that the data will be passed to (or received from) the **hks** tools as ASCII strings representing hex constants, but will of course be handled by FSW as binary bytes and words.

The keyword beginning the fourth argument *may* (if the fourth argument is **RAND** or **RAMP**) or *must* (if the fourth argument is **CONST**) be followed by an equal sign and a constant. In the case of **RAND** or **RAMP** the constant is the seed for the random number generator or the initial value of the ramp; in the case of **CONST** the constant is the literal data to be used.

In the case of **CONST**, if the length specified exceeds that of the literal data representation, the deposited data are padded with 0x00 bytes out to the specified length.

In the case of **NEXT**, the generated datum has length 16 and consists of four concatenated 16-bit words ("little-endian"). The first and third of these are 0x0446 and 0x0444 respectively. The fourth is obtained by calling the **getnextserial** capability of **hks** tools (section 2.3.2.2.3). The second word is the one's-complement of the fourth:

| Word Number | Complement of Serial Number | Word Number | Serial Number |
|---|---|---|---|
| 0446 | (complement of 4th word) | 0444 | (per **getnextserial**) |

**Figure 2-7: Structure of Datum Generated by NEXT**

In order to preclude "gaps" in the serial numbers of generated commands, the **UniScript** interpreter employs the following logic.

If the designated buffer does not contain data "all the way out" to the designated offset, the unoccupied bytes preceding the deposition point are padded with 0x00 bytes.

Each data generation action causes the "**$L**" and "**$CRC**" variables to be recomputed by **UniScript** (see Table 2-6: **Uniscript** Built-In Variables).

The following script statements illustrate the above description:

| Script File | Comments |
|---|---|
| ; Various methods of generating data | |
| DATA 1, 0, 4, CONST=0x_03_02_01_00 | Set first four bytes of **$B1** = 0x03020100 |
| DATA 2, 0, 6, RAMP=0 | Set first six bytes of **$B2** = 0x000102030405 |
| DATA 1, 0,28, RAND=53 | Set first 28 bytes of **$B1** = random values |
| ; Build a DCE 56-byte command "the HARD way" | |
| DATA 1, 0,16, CONST=0x_045A_0001_0458_FFFE_0456_0000_0454_FFFF | 4th and 3rd arguments = 0, 1 |
| DATA 1,16,16, CONST=0x_0452_FFFF_0450_0000_044E_FFFF_044C_0000 | 2nd and 1st arguments = 0, 0 |
| DATA 1,32, 8, CONST=0x_044A_FFFC_0448_0003 | 0th argument = 3 |
| DATA 1,40, 8, NEXT | DCE command serial number |
| DATA 1,48, 8, CONST=0x_0442_0303_0440_FCFC | Opcode |

It is obviously important that the generation and use of serial numbers, which occurs when **NEXT** occurs in a **DATA** local directive and also, implicitly, whenever a DCE command is generated and up-linked, be controllable so that new serial numbers aren't generated when the current one should be used. The logic controlling the generation and use of serial numbers in **UniScript** is as follows. The value introduced in data generation by the **NEXT** argument is provided by the **UniScript** Perl variable **$NEXTSEQ**. The "use count" of **$NEXTSEQ** is a variable **$NN**. **$NN** is initially zero. *Whenever a command is synthesized,*

```
if $NN == 0
    $NEXTSEQ = `getnextserial`;
    unshift $SEQNO, $NEXTSEQ;
    use $NEXTSEQ;
```

```
else
    $NN = 0;
    use $NEXTSEQ;
endif.
```

*Whenever **NEXT** occurs as an argument in the **DATA** local directive,*

```
$NEXTSEQ = `getnextserial`;
$NN = $NN + 1;
unshift $SEQNO, $NEXTSEQ;
use $NEXTSEQ;
```

*2.3.8.3.3 DELAY — Delay Prescribed Interval*

This local directive "wastes time" until the interval specified in its sole parameter has elapsed. The interval is represented as an integral number of 10 ms (i.e., .01 second) increments. The directive does not suspend execution of the script (as does **WAIT** — see paragraph 12.3.8.3.13); it merely iterates, the specified number of times, a calculation lasting 10 ms.

*2.3.8.3.4 DIAG — Check for Presence or Absence of Diagnostic Codes*

A very useful capability is that of ascertaining the presence or absence of DCE diagnostic codes in the 32-word array **LFDDIAGS** in the housekeeping data structure. This local directive passes or fails a test-script on the basis whether *any* of the specified codes is present (if the first argument is **ANY**) or *none* of the specified codes is present (if the first argument is **NOTANY**).

The operands are

1. a report file number (**1** or **2**)
2. a keyword (**ANY** or **NOTANY**) and
3. a list of symbols and/or constants whose values will be looked for in the **LFDDIAGS** array.

If the second operand is **ANY** then **DIAG** permits the script to continue if any of the remaining operands is found. If the remaining list consists of a single diagnostic code the operation causes the script to halt if the code is not found in **LFDDIAGx**.

On the other hand, if the second operand is **NOTANY** then **DIAG** permits the script to continue only if none of the remaining operands is found. If the remaining list consists of a single diagnostic code the operation permits the script to continue only if the code is not found in **LFDDIAGx**.

As an example, in the STP, for the verification of requirement 5.1.1.1, it is stated that "... the correct diagnostic code [must be] produced (i.e., DIAG001B, but not DIAG001C or DIAG001D)". The following script accomplishes this:

| Script File | Comments |
|---|---|
| **; Verify that POR initializes DCE to Boot Mode with the ; correct diagnostics** | Comment |
| **SYM DIAG001B0=0x1B, DIAG5A1C=0x1C, DIAG001D=0x1D** | Define diagnostic codes |
| **DTG 1,"POR command being sent"** | Annotate Report 1 file |
| **LFDRSETP** | Initiate POR |
| **WAIT 1** | Wait 1 second |
| **LOG HK** | Regenerate variables for housekeeping telemetry |
| **DIAG 1, ANY, DIAG001B** | Verify that **LFDDIAGS** contains **DIAG001B** |
| **DIAG 1, NOTANY, DIAG001C, DIAG001D** | Verify that **LFDDIAGS** *does not* contain **DIAG001C** or **DIAG001D** |

**Figure 2-8: Checking Diagnostic Codes after POR**

*2.3.8.3.5 DTG — Place Date/Time Group with Annotation in Report*

This local directive performs the useful function of "time-stamping" a report file and supplying a short annotation. The operands are

1. the report file number (**1** or **2**)
2. a string enclosed in double quotes *but not containing double quotes*.

As an example, the script segment

| Script File | Comments |
|---|---|
| **DTG 1,"POR command being sent"** | Annotate Report 1 file |

will place in Report File 1 the text

**Ver 0 Tue Aug 8 14:02:51 PDT 2000 "POR command being sent"**

The **DTG** local directive also executes the function of **WTO** (2.3.8.3.5)

*2.3.8.3.6 ECHO — Echo Command Strings (in Hex) to Report File*

This local directive is a diagnostic tool which forces a hexadecimal representation of every command sent (following the occurrence of the directive) to be placed in the designated Report File (**1** or **2**, representing Report File 1 or Report File 2).

| Script File | Comments |
|---|---|
| **ECHO 2** | Echo Commands to Report File 2 (for diagnostics) |
| | |

The preceding piece of test-script will place in Report File 2 text similar to the following:

```
--------------------------------------------------------------------------------
                    C O M M A N D   P A C K E T
--------------------------------------------------------------------------------
         PARM4          PARM3          PARM2          PARM1          PARM0
045AFFFF 04580000 0456FFFF 04540000 0452FFFF 04500000 044EFFFF 044C0000 044AEBFF 04481400
--------------------------------------------------------------------------------
           SN          OPCODE
0446DAD5 0444252A 04425454 0440ABAB
--------------------------------------------------------------------------------


--------------------------------------------------------------------------------
                    C O M M A N D   P A C K E T
--------------------------------------------------------------------------------
         PARM4          PARM3          PARM2          PARM1          PARM0
045AFFFF 04580000 0456FFFF 04540000 0452FFFF 04500000 044EFFFF 044C0000 044AFFFF 04480000
--------------------------------------------------------------------------------
           SN          OPCODE
0446DAD1 0444252E 04427F7F 04408080
--------------------------------------------------------------------------------


--------------------------------------------------------------------------------
                    C O M M A N D   P A C K E T
--------------------------------------------------------------------------------
         PARM4          PARM3          PARM2          PARM1          PARM0
045AFFFF 04580000 0456FFFE 04540001 0452AAFF 04505500 044E55FF 044CAA00 044AF4FF 04480B00
--------------------------------------------------------------------------------
           SN          OPCODE
0446DACC 04442533 04420303 0440FCFC
--------------------------------------------------------------------------------


--------------------------------------------------------------------------------
                    C O M M A N D   P A C K E T
--------------------------------------------------------------------------------
         PARM4          PARM3          PARM2          PARM1          PARM0
045AFFFF 04580000 0456FFFF 04540000 0452FFFF 04500000 044EFFFF 044C0000 044AFFFF 04480000
--------------------------------------------------------------------------------
           SN          OPCODE
0446DAC7 04442538 04427F7F 04408080
--------------------------------------------------------------------------------


--------------------------------------------------------------------------------
                    C O M M A N D   P A C K E T
--------------------------------------------------------------------------------
         PARM4          PARM3          PARM2          PARM1          PARM0
045AFFFF 04580000 0456FFFF 04540000 0452FFFF 04500000 044EFFFF 044C0000 044AEBFF 04481400
--------------------------------------------------------------------------------
           SN          OPCODE
0446DAC1 0444253E 04425454 0440ABAB
--------------------------------------------------------------------------------
```

**Figure 2-9: Typical Contents of "Echo" File**

*2.3.8.3.7 FSYM — Define File*

This local directive assigns a Unix path-file name to a symbol. This is convenient for the coding of **SUBS** local directives (section 2.3.8.3.11). **FSYM** may have multiple operands, each consisting of, in order, a symbol, an equal sign "=", and a valid Unix path/file description; the operands must be separated by commas. The symbols defined by the **FSYM** local directive are used in Perl open function calls to obtain the handles for the symbolized path/files. They are kept in the Perl hash table **%FSYM**; hence they may duplicate numeric symbols defined by means of the **SYM** local directive (which are kept in **%SYM**). The following script snippets illustrate the use of **FSYM**.

| Script File | Comments |
|---|---|
| **FSYM SUBTEST1=~/FSWSCR/ST1.scr** | Define sub-script file 1 |
| **FSYM SUBTEST2=~/FSWSCR/ST2.scr** | Define sub-script file 2 |
| **FSYM SUBTEST1=~/FSWSCR/ST1.scr,SUBTEST2=~/FSWSCR/ST2.scr** | Same as previous two directives |
| **SUBS SUBTEST1,ABC,0x0008** | Invoke sub-script in file SUBTEST1=~/FSWSCR/ST1.scr |

*2.3.8.3.8 FORM — Place Formatted Report in Report File*

This local directive causes a formatted report to be placed in either the Report File 1 or the Report File 2, depending on whether its first operand is **1** or **2**. The name of the report is the second argument. The currently permitted names are given in Table 2-10.

**Table 2-10: UniScript Report Formats**

| Name of Report | Description |
|---|---|
| **HK0** | Assuming the script segment<br>**LOG 1,HK**<br>**FORM 1,HK0,"Annotation"**<br>This report converts the raw housekeeping data last down-linked from the DCE to a formatted report that it easier to read than the hexadecimal "dump" provided by **LOG**. |
| TBS | TBS |

As illustrated in Table 2-10, the third argument is an optional annotation to be placed in the report header. The following two Figures show the format of the **HK0** report.

```
1440  LFGPHA    0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
          0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
          0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
          0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
          0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
          0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
          0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000

1540  LFGPHB    0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
          0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
          0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
          0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
          0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
          0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
          0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000

1640  LFDERR    0000 0000 0000 0000

1648  LFDERRP   0000 0000 0000 0000 0000 0000 0000 0000

1658  *GDSBITS4   0000 0000 0000 0000 0000 0000

1664  LFDCBUF   FCFC 0303 0001 FFFE 0000 FFFF 0000 FFFF 0000 FFFF 0000 FFFF 0000

1680  LFCTIME   00000000  16C5  LFTDVAB   00      16E4.2  RFPA     0       16F2.7          0
1684          0000      16C6  LFTAMPA   00      16E4.3  RFPB     0       16F2.8          0
1686          0000      16C7  LFTAMPB   00      16E4.4           0       16F2.9          0
1688          0000      16C8  LFTTDCA   00      16E4.5           0       16F2.A          0
168A          0000      16C9  LFTTDCB   00      16E4.6  LFHRMPA  0       16F2.B          0
168C          0000      16CA  LFTLVPC   00      16E4.7  LFHRMPB  0       16F2.C          0
168E          0000      16CB  LFTDCE    00      16E4.8           0       16F2.D  LFRMENA  0
1690          0000      16CC  LFHFAN    00      16E4.9  LFDCODE  0       16F2.E          0
1692          0000      16CD  LFVP5DC   00      16E4.A           0       16F2.F          0
1694  LFCDECA   00000000  16CE            00      16E4.B           0       16F4  LFSBITS1  0000
1698  LFCDECB   00000000  16CF            00      16E4.C           0       16F4.0  RFD1HZ    0
169C  RFXSVD2   0000      16D0  RFVP21D   00      16E4.D           0       16F4.1          0
169E          0000      16D1  RFRESER0  00      16E4.E  LFDWDDIS 0       16F4.2          0
16A0          0000      16D2  LFHIMONA  00      16E4.F  RFRLATCH 0       16F4.3  LFDOPERT 0
```

```
16A2          0000     16D3  LFHIMONB  00      16E6  LFGUQTA   00      16F4.4  RFDPROJT  0
16A4  LFCSDC1  00000000  16D4  LFHVMONA  00      16E7  LFGUQTB   00      16F4.5          0
16A8  LFCSDC2  00000000  16D5  LFHVMONB  00      16E8  LFGBWKAX  00      16F4.6          0
16AC  LFCFECA  0000     16D6  LFHQANA   00      16E9  LFGBWKBX  00      16F4.7          0
16AE          0000     16D7  LFHQANB   00      16EA  LFGEWKAX  00      16F4.8          0
16B0  LFCFECB  0000     16D8  LFVP15TA  00      16EB  LFGEWKBX  00      16F4.9          0
16B2          0000     16D9  LFVP15TB  00      16EC          00      16F4.A          0
16B4          0000     16DA  LFVP5TA   00      16ED          00      16F4.B          0
16B6          0000     16DB  LFVP5TB   00      16EE  LFGLQTA   00      16F4.E  RFDDETID  0
16B8          0000     16DC  LFVM15TA  00      16EF  LFGLQTB   00      16F4.F          0
16BA          0000     16DD  LFVM15TB  00      16F0  LFGTTAX   00      16F6  *CRC_Mask_A  0000
16BC  LFGSHFAX  00      16DE  LFVM5TA   00      16F1  LFGTTBX   00      16F8  *CRC_Mask_B  0000
16BD  LFGSTRAX  00      16DF  LFVM5TB   00      16F2  LFSBITS0  0000    16FA  LFMRAM      0000
16BE  LFGSHFBX  00      16E0  LFVP15D   00      16F2.0          0       16FC  LFMROM      0000
16BF  LFGSTRBX  00      16E1  LFVM15D   00      16F2.1          0       16FE  LFDSWVER    0000
16C0  LFTDVAA   00      16E2  LFHVSETA  00      16F2.2          0       1700  LFCPKT      00000000
16C1  LFTACT    00      16E3  LFHVSETB  00      16F2.3          0       1704  LFDLSTC     00
16C2  LFTHVFM   00      16E4  LFSBITS2  0000    16F2.4          0       1705  LFHSTATE    00
16C3  LFTHVPS   00      16E4.0  LFPOVDA  0      16F2.5          0       1706  LFGREFA     00
16C4  LFTIP     00      16E4.1  LFPOVDB  0      16F2.6  LFHVENA  0      1707  LFGREFB     00
```

**Figure 2-10: HK0 Report Format (Page 1 of 2)**

```
1708  LFDDIAG0   00      172D.6  LFDPROM   0       17D4          0000 0000 0000 0000 0000 0000 0000 0000
1709  LFDDIAG1   00      172D.7  RFDPROM2  0       17E4          0000 0000 0000 0000 0000 0000 0000 0000
170A  LFRILIM    00      172E  LFGSTIMA   00      17F4          0000 0000 0000 0000 0000 0000 0000 0000
170B  LFHVILIM   00      172F  LFGSTIMB   00      1804          0000 0000 0000 0000 0000 0000 0000 0000
170C  LFDCMDX    0000    1730  LFDPERF    00000000  1814          0000 0000 0000 0000 0000 0000 0000 0000
170E  LFPCNTA    0000    1734  RFDPEEK    00      1824          0000 0000 0000 0000 0000 0000 0000 0000
1710  LFVPMON    00      1735  LFRPOS     00      1834          0000 0000
1711  LFDDIAG2   00      1736  LFDMEXTR   00
1712  *STATUS_BITS300     1737  LFDMDATA   00
1713  *DOOR_POS_A 00
1714  LFRACTAN   00      1738  LFDMONS    0000 0000 0000 0000

1715  LFRMTRAN   00
1716  LFRAXI     00      1740  LFDMADD    0000 0000 0000 0000 0000 0000 0000 0000
1717  LFRAXAN    00
1718  LFDCMDR    0000    1750  LFGBWKAY   00      1838  LIMIT_INDEX 0000
171A  LFMCRC     0000    1751  LFGEWKAY   00      183A  TENCOUNT    0000
171C  LFPINTA    00      1752  LFGBWKBY   00      183C  RESE...NTER 0000
171D  *SAA_INT_TIME00     1753  LFGEWKBY   00      183E  END...STAMP 0000
171E  LFPCNTB    0000    1754  LFRTIMER   0000
1720  LFHVLOA    00      1756  *DUMMY     0000
1721  LFHVLOB    00      1758          0000
1722  *HV_AUTO_R  0000     175A  *DSBITS5   0000
1724  LFHVNOMA   00      175C          0000
1725  LFHVNOMB   00      175E  LFGTTAY    00
1726  LFHVMAXA   00      175F  LFGTTBY    00
1727  LFHVMAXB   00      1760  LFGSHFAY   00
1728  LFPINTB    0000    1761  LFGSTRAY   00
172A  LFSBITS3   0000    1762  LFGSHFBY   00
172A.0  LFRLA    0       1763  LFGSTRBY   00
172A.1  LFROP    0       1764  RFXSVD1    00
172A.2  LFRCL    0       1775  LFHRAMPT   00
172A.3  RFHVENA1  0      1776  LFDDIAG3   00
172A.4  LFHVPWR   0      1777  LFDDIAG4   00
172A.5  LFHVQPWR  0      1778  LFMU7      0000
172A.6  LFHPLUG   0      177A  LFMU2      0000
172A.7  LFDETID0  0      177C  LFMLWR     0000
172A.8  LFRAXPWR  0      177E  LFMUPR     0000
172A.9  LFRACTEN  0
172A.A  RFRACTEN  0      1780  LFDDIAGS   0000 0000 0000 0000 0000 0000 0000 0000
172A.B  LFRMPWR   0                0000 0000 0000 0000 0000 0000 0000 0000
172A.C          0              0000 0000 0000 0000 0000 0000 0000 0000
172A.D  RFHV0     0                0000 0000 0000 0000 0000 0000 0000 0000
172A.E  LFRPLUG   0
172A.F  LFDETID1  0      17C0  LFVP5DA    00
172C  LFSBITS4   00      17C1  LFVP5DB    00
172C.0  RFDCTRL0  0      17C2  *MUX_VCC   00
172C.1  LFRACT1   0      17C3  *MUX_M52V  00
172C.2  LFRACT2   0      17C4  *MUX_VREF  00
172C.3  RFDCTRL3  0      17C5  *MUX_GND1  00
172C.4  RFDCTRL4  0      17C6  *MUX_GND2  00
172C.5  RFDCTRL5  0      17C7  *MUX_GND3  00
172C.6  RFDCTRL6  0      17C8          0000
172C.7  LFRLSOVD  0      17CA          0000
172D  LFSBITS5   00      17CC          0000
172D.0  RFDC2RL0  0      17CE  *DIBexception00
172D.1  RFDC2RL1  0      17CF  *PMONlimit  00
172D.2  RFDC2RL2  0      17D0  *HV_A_SET   00
172D.3  LFRMDIR   0      17D1  *HV_B_SET   00
172D.4              17D2  LFHVTGTA   00
172D.5  RFDC2RL5  0      17D3  LFHVTGTB   00
```

**Figure 2-11: HK0 Report Format (Page 2 of 2)**

*2.3.8.3.9 LOG — Refresh Housekeeping Data*

The first operand of this local directive is a Report File number, **1** or **2**. The second operand is a list of housekeeping variables (Table 2-3: DCE Housekeeping Mnemonics).

If the second operand is a list of housekeeping variables the directive places a sequence of lines in the specified Report File, each containing the name of one of the variables in the list, paired with an appropriately formatted representation of its current value. A (conceptual) example is shown in Figure 2-12. This output might have been produced by the following directive:

| Script File | Comments |
|---|---|
| **LOG 1,LFDERR,LFDDIAGS,LFCFECA,LFDCMDX,LFDCNTA,LFRACTAN,LFCPKT,RFRLATCH,LFRACT1,1** | Request data report |

```
Addr Addr HK-Name
0000-001F LFDERR       01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01
0020-0022              01 01 01

0100-011F LFDDIAGS     2345 2345 2345 2345 2345 2345 2345 2345 2345 2345 2345 2345 2345 2345 2345 2345
0120-0125              2345 2345 2345

0200-021F LFCFECA      6789ABCD  6789ABCD  6789ABCD  6789ABCD  6789ABCD  6789ABCD  6789ABCD  6789ABCD
0220-022B              6789ABCD  6789ABCD  6789ABCD

0300-0301 LFDCMDX      0000
0302-0303 LFPCNTA      0000
0304-0304 LFRACTAN     00
0308-030B LFCPKT       00000000

Addr Mask HK-Bit-Name
0000 8000 RFRLATCH     0
0000 02   LFRACT1      1

Len  CRC  Buffer
0023 F16D 1            EF EF EF EF EF EF EF EF EF EF EF EF EF EF EF EF EF EF EF EF EF EF EF EF EF EF EF EF EF EF EF
                       EF EF EF
```

**Figure 2-12: Example Output of LOG**

*2.3.8.3.10 RECV — Receive Down-link Data*

This local directive is assumed to have been preceded by an **LFDDNLOD** command; **RECV** calls upon **hkprint** to supply the down-linked data and reformats it for placement in the buffer specified by its first operand (**1** or **2**). The amount of down-linked data captured by this directive is specified by its second operand. The following script illustrates the process.

| Script File | Comments |
|---|---|
| **SYM NBYTES=106, ABC=X:0xF0F0** | Define RAM address, length of block |
| **LFDDNLOD ABC, NBYTES** | Initiate down-linking of the data |
| **RECV 2, NBYTES** | Down-link to **$B2** |

*2.3.8.3.11 SUBS — Execute Sub-Script*

A script file may transfer control of **UniScript** to another script file and receive control back from the "sub-script" file when its actions terminate. The "sub-script" may in turn invoke another "sub-sub-script".

*2.3.8.3.11.1 SUBS Nesting Levels*

A "main" script (one that is specified on the command line that starts **UniScript**), is considered as executing at level 0. It may, in turn, **SUBS** to another script, which would then be executing at level 1; this level-1 script may invoke a level-2 script, and so on. There is no maximum permitted nesting depth.

*2.3.8.3.11.2 Form of the SUBS Directive*

The **SUBS** local directive requires at least one, and possibly as many as nine, arguments. The first operand is a Unix file specification. Additional operands take the forms specified in 2.3.7.2, separated, if necessary, by commas. An example is:

| Script File | Comments |
|---|---|
| **FSYM SS1=../SCRIPTS/SUBSCRIPT1.SCR** | Definition of sub-script file |
| **FSYM SS2=../SCRIPTS/SUBSCRIPT2.SCR** | Definition of sub-script file |
| **SUBS SS1,ABC,1,SS2** | Invoke sub-script; pass arguments **ABC,1,SS2** to it |

*2.3.8.3.11.3 Sub-Script Parameters*

A sub-script may employ from zero to eight "parameters"[6]. These are correlated, of course, with the zero to eight arguments specified in a **SUBS** to it. The parameters are indicated by the eight symbols **#0**, **#1**, …, **#7**. **UniScript** examines each line of the script, before executing it, for the occurrence of parameters. *This examination takes absolutely no account of **UniScript** syntax.* Whenever one of the forms "**#0**", "**#1**", …, "**#7**" is encountered as a substring of a script statement — which is regarded simply as a string of ASCII characters — it is replaced with the corresponding argument of the **SUBS** local directive that invoked the sub-script[7].

The replacement is non-recursive: the strings that replace the parameters are not re-examined. The values of the parameter symbols **#0**, **#1**, …, **#7** are "pushed down" when a **SUBS** is interpreted and replaced with new values specified in the **SUBS** directive. When

[6] The "main" (level-0) script may also be supplied with up to eight arguments. These are coded in the Unix command that invokes **UniScript**: e.g., `perl ../UniScript.pl stp5_1_2_5a "0A80,C000,0,0,0,0,0,0"`. The eight arguments are enclosed in double quotes.

[7] Assuming that a corresponding argument has been specified! Missing arguments are replaced by the null string.

the invoked sub-script terminates the "pushed" values are "popped" back into the parameter symbols.

The following illustrates invocation of a sub-script with replacement.

| Main Script File | Sub-Script File SS1 | SS1 with Parameter Substitution |
|---|---|---|
| **FILE SS1=../SPECIAL/SUBSCRIPT1.SCR** | | |
| **FILE SS2=../SPECIAL/SUBSCRIPT2.SCR** | | |
| **SUBS SS1,LFDCOPY,ABC,XYZ,26,LFDDIAGS** | | |
| | **WAIT 2,HK** | **WAIT 2,HK** |
| | **DTG 1,"Sending #0"** | **DTG 1,"Sending LFDCOPY"** |
| | **#0 #1,#2,#3** | **LFDCOPY ABC,XYZ,26** |
| | **WAIT 2,HK** | **WAIT 2,HK** |
| | **LOG 1,#4** | **LOG 1,LFDDIAGS** |
| | **DIAG NOTANY,DIAG0011** | **DIAG NOTANY,DIAG0011** |
| | **DTG 1,"#0 accepted"** | **DTG 1,"LFDCOPY accepted"** |
| **•••** | | |

*2.3.8.3.11.4 Sub-Script Success or Failure*

Each script returns a "success" or "failure" condition to the script that invoked it. (Failure can be the outcome of execution of a **CHECK**, **DIAG**, or **SUBS** local directive, or the detection by **UniScript** of a syntax error in a (sub-) script. The return of a "failure" condition causes the invoking script to fail, returning failure to the script that invoked it, and so on — up to the "main" level-0 script which, upon receiving a failure return from a subscript, terminates execution. Hence the invocation of a sub-script will cause the main script to fail unless all sub-scripts succeed that were executed as a result of the invocation. A sub-script which occasions no failures ultimately terminates successfully (when **UniScript** encounters the end of the sub-script file) and returns "success" to the invoking script.

*2.3.8.3.12 SYM — Define Symbolic Value*

This local directive defines symbols that will be used later in the script. A single SYM directive can define multiple symbols. The symbols thus defined are stored as elements of a Perl hash, **%SYM**, keyed by symbol name; their values may be referred to symbolically in DCE command parameters and in local directives (see examples, e.g., section) their values may therefore be accessed by means of the Perl hash-reference construct (e.g., **$SYM {"ABC"}** for the symbol **ABC** in a **CHECK** directive).

*2.3.8.3.13 WAIT — Wait Prescribed Interval or until Housekeeping Available*

This local directive suspends interpretation of the script until a condition specified in its arguments has been satisfied. There is one mandatory argument and one optional one. The mandatory first argument specifies an interval in seconds. The interval must be

represented as an integer, since the Unix function used to implement this suspension provides a granularity of 1 second only. The second, optional argument is **HK**; if present, this argument allows script interpretation as soon as a complete Housekeeping Data packet is available from the **hks** tools — but no later than the interval specified in its first parameter has elapsed.

| Script File | Comments |
|---|---|
| **WAIT 10** | Wait 10 seconds |
| **WAIT 2,HK** | Wait for Housekeeping Data, but no longer than 2 seconds |

*2.3.8.3.14 WTO — Write Message to Test Operator*

This local directive displays the parameter (a double-quoted string — section 2.3.7.2.1) on the test operator console. This is useful for keeping the operator informed as to the progress of the script.

| Script File | Comments |
|---|---|
| **WTO "Test concluded"** | Signal conclusion of test to operator. |

*2.3.8.3.15 WTOR — Wait until Operator Reply*

This local directive performs the same function as does **WTO**, but has the additional characteristic of suspending interpretation of the script until the operator has entered a **Y** or an **N** on the test console. If the operator enters **N** the execution of the script is terminated. An example is:

| Script File | Comments |
|---|---|
| **WTOR "Attach scope now; enter Y to continue test"** | Wait for test hardware to be hooked up. |

*2.3.8.3.16 XMIT — Transmit Up-link Data*

This local directive is assumed to be followed by an **LFDUPLOD** command; **XMIT** calls upon **cmd_client** to up-link the initial portion of the buffer specified by its first operand (**1** or **2**). The amount of up-linked data emitted by this directive is specified by its second operand. The following script illustrates the process.

| Script File | Comments |
|---|---|
| **SYM NBYTES=106, ABC=X:0xF0F0** | Define RAM address, length of block |
| **XMIT 2, NBYTES** | Up-link from **$B2** |
| **LFDUPLOD ABC, NBYTES** | Move data to target address |

### 2.3.9 Examples of Use of the Script Language

#### 2.3.9.1 Requirement 5.1.1.1 — Initialize to Boot State after Reset

In the STP, section 4.1.4, two of the test steps for verifying Requirement 5.1.1.1 are

- Verify that DCE is in Boot State after a Power On Reset (POR)
- Verify that DCE is in Boot State after a Watchdog Reset (WDR).

The following script performs this verification. Note that **UniTest** is not required for this test.

| Script File | Comments |
|---|---|
| **ECHO 2** | Echo Commands to Report File 2 (for diagnostics) |
| **DTG 1,"Verify Boot Mode after POR"** | Annotation to Report File 1 |
| **LFDRSTP** | Send DCE Command to execute POR |
| **WAIT 10** | Wait 10 seconds (reset takes about 10 seconds) |
| **LOG 1,$LFSBITS** | Get **LFSBITS1** from **hkprint** |
| **CHECK 1,(($LFSBITS1 & hex("0080")) != 0)** | Verify bit **LFDOPERT** of **LFSBITS1** is set — fail test if not |
| **DTG 1,"Verify Boot Mode after WDR"** | Annotation to Report File 1 |
| **WAIT 10** | Wait for POR to complete |
| **LFDRSTW** | Send DCE Command to execute WDR |
| **WAIT 2** | Wait 2 Seconds |
| **LOG 1, $LFSBITS1** | Get **$LFSBITS1** from **hkprint** |
| **CHECK 1,(($LFSBITS1 & hex("0080")) != 0)** | Verify bit **LFDOPERT** of **LFSBITS1** is set — fail test if not |

### 2.3.10 Script Language Syntax

The following BNF productions (with a slight admixture of "regular expression" language) describe the statements of the script language. In these productions, terminals are printed in **Courier Bold**, non-terminal constructs are printed as names within angle-brackets, as in ⟨*nonterminal*⟩, and the punctuation associated with regular expressions appears in Arial Regular. In several of the productions informal reference is made to a table in section 2.3.5 in lieu of an awkward alternating list of several tens of terminal symbols. The references to these pre-defined terminal constants are in ***(Arial Bold-Italic)***; as are other informal shortcut descriptions.

The productions describe the grammar of a script in which *parameter substitution has already been performed* (section 2.3.8.3.11.3).

⟨*alpha*⟩ → [**A-Za-z_?**]
⟨*digit*⟩ → [**0-9**]
⟨*white*⟩ → [**\s**]*

⟨*hexdigit*⟩ → [**A-Fa-f0-9**]
⟨*alphanum*⟩ → [**A-Za-z0-9_?**]
⟨*id*⟩ → ⟨*alpha*⟩ | ⟨*alpha*⟩ ⟨*alphanum*⟩*
⟨*Dconst*⟩ → ⟨*digit*⟩+
⟨*Xconst*⟩ → **0x** (⟨*hexdigit*⟩{2} | ⟨*hexdigit*⟩{4})
⟨*Xlist*⟩ → **0x** ⟨*hexdigit*⟩ (⟨*hexdigit*⟩* _ )*
⟨*const*⟩ → ⟨*Dconst*⟩ | ⟨*Xconst*⟩
⟨*XLconst*⟩ → ⟨*Dconst*⟩ | ⟨*Xconst*⟩ | ⟨*Xlist*⟩
⟨*c-list*⟩ → ⟨*XLconst*⟩ (⟨*comma*⟩⟨*XLconst*⟩)*
⟨*opnd*⟩ → ⟨*const*⟩ | ⟨*id*⟩
⟨*dceop*⟩ → ⟨*white*⟩ ***(see table Table 2-2, column 1)***
⟨*sfr*⟩ → ***(see Table 2-4, column 1)***
⟨*rx*⟩ → **R0** | **R1** | **R2** | **R3** | **R4** | **R5** | **R6** | **R7**
⟨*reg-spec*⟩ → ⟨*sfr*⟩ | ⟨*rx*⟩ : ⟨*rx*⟩
⟨*HKvar*⟩ → ***(see Table 2-3, column 1)***
⟨*comment*⟩ → **;**
⟨*comma*⟩ → ⟨*white*⟩ **,** ⟨*white*⟩
⟨*eq*⟩ → ⟨*white*⟩ = ⟨*white*⟩
⟨*file*⟩ → **1** | **2**
⟨*buffer*⟩ → **1** | **2**
⟨*dq*⟩ → **"**
⟨*non-dq*⟩ → [~**"**]+
⟨*string*⟩ → ⟨*dq*⟩⟨*non-dq*⟩⟨*dq*⟩
⟨*PerlExp*⟩ → ***(any valid Perl 5 "boolean" expression)***
⟨*FileExp*⟩ → ***(any valid Unix path/file expression)***
⟨*reg-assign*⟩ → ⟨*reg-spec*⟩ = ⟨*const*⟩
⟨*mem-assign*⟩ → ⟨*id*⟩⟨*eq*⟩ (**B:** | **X:** | **C:** | **D:**) ⟨*const*⟩
⟨*val-assign*⟩ → ⟨*id*⟩⟨*eq*⟩⟨*opnd*⟩ (⟨*comma*⟩⟨*opnd*⟩)*
⟨*sym-assign*⟩ → ⟨*id*⟩⟨*eq*⟩⟨*opnd*⟩ (+ ⟨*opnd*⟩)?
⟨*file-assign*⟩ → ⟨*id*⟩⟨*eq*⟩⟨*FileExp*⟩
⟨*dtg*⟩ → ⟨*white*⟩ **DTG** ⟨*white*⟩⟨*file*⟩⟨*comma*⟩⟨*string*⟩
⟨*wto*⟩ → ⟨*white*⟩ **WTO** ⟨*white*⟩⟨*string*⟩
⟨*sync*⟩ → ⟨*white*⟩ **WTOR** ⟨*white*⟩⟨*string*⟩
⟨*wait*⟩ → ⟨*white*⟩ **WAIT** ⟨*white*⟩⟨*Dconst*⟩ (⟨*comma*⟩ **HK**)?
⟨*delay*⟩ → ⟨*white*⟩ **DELAY** ⟨*white*⟩⟨*Dconst*⟩
⟨*echo*⟩ → ⟨*white*⟩ **ECHO** ⟨*white*⟩⟨*file*⟩
⟨*form*⟩ → ⟨*white*⟩ **FORM** ⟨*white*⟩⟨*file*⟩⟨*comma*⟩⟨*id*⟩
⟨*log*⟩ → ⟨*white*⟩ **LOG** ⟨*white*⟩⟨*file*⟩ (⟨*comma*⟩⟨*HKvar*⟩)*

⟨*data*⟩ → ⟨*white*⟩ **DATA** ⟨*white*⟩⟨*buffer*⟩⟨*comma*⟩(⟨*opnd*⟩⟨*comma*⟩){2}
((**RAND** | **RAMP**) ⟨*eq*⟩⟨*const*⟩ | (**CONST** ⟨*eq*⟩⟨*c-list*⟩ | **NEXT**

⟨*check*⟩ → ⟨*white*⟩ **CHECK** ⟨*white*⟩⟨*file*⟩⟨*comma*⟩⟨*PerlExp*⟩

⟨*diag*⟩ → ⟨*white*⟩ **DIAG** ⟨*white*⟩⟨*file*⟩⟨*comma*⟩ (**ANY** | **NOTANY**)
(⟨*comma*⟩⟨*opnd*⟩)+

⟨*sym*⟩ → ⟨*white*⟩ **SYM** ⟨*white*⟩⟨*sym-assign*⟩ (⟨*comma*⟩⟨*sym-assign*⟩)*

⟨*fsym*⟩ → ⟨*white*⟩ **FSYM** ⟨*white*⟩⟨*file-assign*⟩ (⟨*comma*⟩⟨*file-assign*⟩)*

⟨*subs*⟩ → ⟨*white*⟩ **SUBS** ⟨*white*⟩⟨*id*⟩ (⟨*comma*⟩⟨*id*⟩){0,8}

⟨*local*⟩ → ⟨*dtg*⟩ | ⟨*wto*⟩ | ⟨*sync*⟩ | ⟨*wait*⟩ | ⟨*delay*⟩ | ⟨*echo*⟩ | ⟨*form*⟩ | ⟨*log*⟩ |
⟨*data*⟩ |
⟨*diag*⟩ | ⟨*check*⟩ | ⟨*sym*⟩ | ⟨*fsym*⟩

⟨*reg*⟩ → ⟨*white*⟩ **REG** ⟨*white*⟩⟨*reg-assign*⟩ (⟨*comma*⟩⟨*reg-assign*⟩){2,4}

⟨*set*⟩ → ⟨*white*⟩ **SET** ⟨*white*⟩⟨*val-assign*⟩

⟨*clear*⟩ → ⟨*white*⟩ **CLEAR**

⟨*xfer*⟩ → ⟨*white*⟩ (**CALL** | **JUMP**) ⟨*white*⟩⟨*opnd*⟩

⟨*DCEcmd*⟩ → ⟨*white*⟩⟨*dceop*⟩ | ⟨*dceop*⟩⟨*white*⟩⟨*opnd*⟩ (⟨*comma*⟩⟨*opnd*⟩)*

⟨*Utest*⟩ → ⟨*reg*⟩ | ⟨*sym*⟩ | ⟨*set*⟩ | ⟨*clear*⟩ | ⟨*xfer*⟩

⟨*cmd*⟩ → ⟨*DCEcmd*⟩ | ⟨*Utest*⟩ | ⟨*local*⟩

⟨*stmt*⟩ → ⟨*cmd*⟩ | ⟨*comment*⟩

⟨*script*⟩ → ⟨*stmt*⟩+

## 3. GLOSSARY

| | |
|---|---|
| BNF | Backus-Naur Form |
| COS | Cosmic Origins Spectrograph |
| DCE | Device Control Electronics (sc. COS) |
| EGSE | Electronic Ground Support Equipment |
| FSW | Flight Software |
| GUI | Graphic User Interface |
| IDE | Integrated Development Environment |
| POR | Power-On Reset |
| SFR | Special Function Register |
| STP | Software Test Plan |
| WDR | Watchdog Reset |