VIRMOS
Mask Manufacturing Unit
Maintenance Manual

Authors :	**L.Chiappetti, G.Conti,** **D. Maccagni, E.Mattaini**	*Approved by :*	**O. LeFèvre, D.Maccagni**

EVOLUTION PAGE

Note: text items changed between issue *3.0* and *3.1* are indicated in this font and/or colour in both the printed and HTML version, with a bar on the margin in the printed version and on coloured background (for the entire page) in the HTML version..

Issue	Rev.	Paragr.	Date	Observations
0	0	all	19/10/1999	Internal issue continuously updated
1	0	all	29/03/2000	Draft issue for PAE
2	0	all	29/06/2000	MMU delivery at Paranal
2	1	8.2, 9.3, 9.5, 11.1.1, 11.1.1.5, 11.1.1.19, 11.1.1.20, 11.1.3.4, 11.1.3.12, 11.1.5.3, 11.1.5.7, 11.1.6.9, 11.4.6, 11.5.2, 11.5.3, 11.5.6, 11.6, 12.1, 12.2, 12.3	15/01/2002	VIMOS commissioning
2	2	8.2	01/03/02	Erratum on pag. 16, issued only pages 1,2,16
3	0	7.5, 11.1.1, 11.1.4.8, 11.1.5.4, 11.1.6.1, 11.1.6.13, 11.1.6.14, 11.1.14.3, 11.2.3.2, 11.2.3.4, 11.4.3, 12.2, 12.4	12/04/2002	Second VIMOS commissioning
3	*1*	7.1.3, 11.1.5.3, 11.1.5.4, 11.1.6.1, 11.1.6.13, 11.1.7.5, 11.1.7.13, 11.1.8, 11.1.9, 11.1.14, 11.1.14.2, 11.1.14.3, 11.1.20.1, 11.1.23, 11.1.24, 11.1.25, 11.2.2, 11.2.3.1, 11.2.3.3, 11.2.3.5, 11.4.2, 11.5.6, 11.5.7	*28/03/2003*	*Delivery of MHS v6 and Cut Manager v2 and…*
		1, 7.3, 7.4, 8.1, 8.2, 11.4.1, 11.4.3, 11.5.2, 11.6, 12.4		*… removal of references to NIRMOS*

1. Introduction

The present document (**Maintenance Manual**) is the second part of **the VIRMOS Mask Manufacturing Unit (MMU) operation and maintenance manual**. The MMU is the complex of hardware and software developed by the VIRMOS Consortium to make the multi-slit masks for the VIMOS ~~*and NIRMOS*~~ spectrographs. An executive summary of the MMU functions is presented in the MMU Global Description document [ref. 11].

This second part collects the information not needed to the operator for routine operations, and namely, after some chapter common to part 1 (sections 2-5) : a description of the MMU hardware installation (in section 6), a description of the opearting system configuration for the MMU computers (in section 7), a description of the installation of software items supplied by third parties, inclusive of the LPKF programs (in section 8), a description of special installations (in section 9), a description of the development and installation environment for the software developed at IFCTR (in section 10), a complete programmers' reference for the software developed at IFCTR (in section 11), and a description of other extraordinary maintenance activities, inclusive of replacement of LRUs (in section 12).

A companion document (part 1 Operator Manual) will include all information necessary for routine operations, inclusive of step by step description of standard activities.

The present document will be delivered in hardcopy (and computer readable form if required) as well as a set of HTML files for easier on-line consultation.

2. Applicable and Reference Documents

Make reference to the same section of the Operator Manual

3. Acronyms

Make reference to the same section of the Operator Manual

4. Stylistic conventions

Make reference to the same section of the Operator Manual

5. System description

Make reference to the same section of the Operator Manual

6. Hardware installation

The hardware listed in section 5 will be delivered according to a separate detailed packing list.

The installation of the LPKF StencilLaser machine and related equipment will be done with the assistance of LPKF personnel, and involves the connection of the cables and pipes described in section 5.1 of the Operator Manual.

For the SC the installation will consist in just the mounting of the cabinet on its table, and of the rotatable cable hanger on the ceiling of the room.

For the IC robot refer to Antil documentation [ref. 10] for the alignment operations.

For basic computer hardware installation just connect the video, keyboard and mouse to the CPU using the colour-coded cables provided (see figure in section 7.1). Additional steps requiring installation of special cards are described in individual computer configuration below. We intend to deliver hardware with such cards already installed. For the MHCU computer it will also be necessary to connect the additional loudspeakers to the audio jack.

The following items will come preinstalled :

- springs for StencilLaser clamping unit (mounted on StencilLaser)
- Exhauster switch and safety lamp (mounted on StencilLaser)
- Heidenhain and LPKF card (installed in backplane of MMCU computer)
- Mask stand and mask clamp (mounted on Z-displacement unit of IC Robot)
- RS422/RS322 interface converter (mounted on Z-displacement unit of IC Robot)
- National Instruments serial cards (installed in backplane of MHCU and `spare` computers)

7. System software installation

The operating system (Windows NT 4.0 Service Pack 3) came factory-installed from Dell. At first switch on a simple setup utility was automatically run. In case a reinstallation is necessary the Windows NT set-up floppies, and the system and service pack CD-ROM are available with the relevant manuals. Additional (non-standard) post-installation steps are described in individual computer configuration below.

Note that the typical Dell arrangement is to make a copy of the installation CD-ROM for the relevant processor architecture on the C : hard disk (directory `C :\I386` which is different by the runtime directory for Windows NT). Therefore installation of NT software components not originally installed does not require the usage of the CD-ROM. If a setup utility requires `A :\I386` change it to `C :\I386` before proceeding further !

On all computers (from Control Panel ⇒ Display) we have set up the video as 768×1024 pixels, 65536 colours (**required** by Visual Basic), large fonts. This, as the desktop appearance, screen saver, etc., can be changed according to taste. Currently we set a screen saver after an idle period of 10 minutes, and a red desktop background for the `administrator` account to remind this is not the standard `operator` account.

It is our intention to deliver the three computers as identically configured as possible, in order to facilitate the interchange of them in case of failures with the minimal reconfiguration. This has not always been possible and deviations occurred in the following cases (refer also to section 9 for details):

- hardware ports are installed and configured as required by the primary usage intended for each computer. Therefore the MMCU will have the LPKF custom cards installed and configured, while the other two computers will have a National Instrument serial card each.
- the Microsoft Visual Basic environment has been installed on the spare computer only. Its installation has updated the Internet Explorer and some other Microsoft components of the system.
- the roughness meter and bar code configuration sofware is installed only on the spare computer
- the VNC software used to make screen dumps (for the present document) from an Unix workstation
- the MMCU computer hosts also a provisional Windows 95 system (italian) for dual boot, which was necessary for using the LPKF calibration mode of BoardMaster until LPKF provided full support to Windows NT with the 32-bit version of StencilMaster (which occurred in April 2000). This is no longer necessary.

Unless otherwise stated, these post-installation actions shall be performed on all three computers after a fresh reinstallation of the operating system. It is assumed that this has already been done on the delivered systems.

We also supply the standard NT "Emergency Repair Disks" updated at the time of writing of this manual.

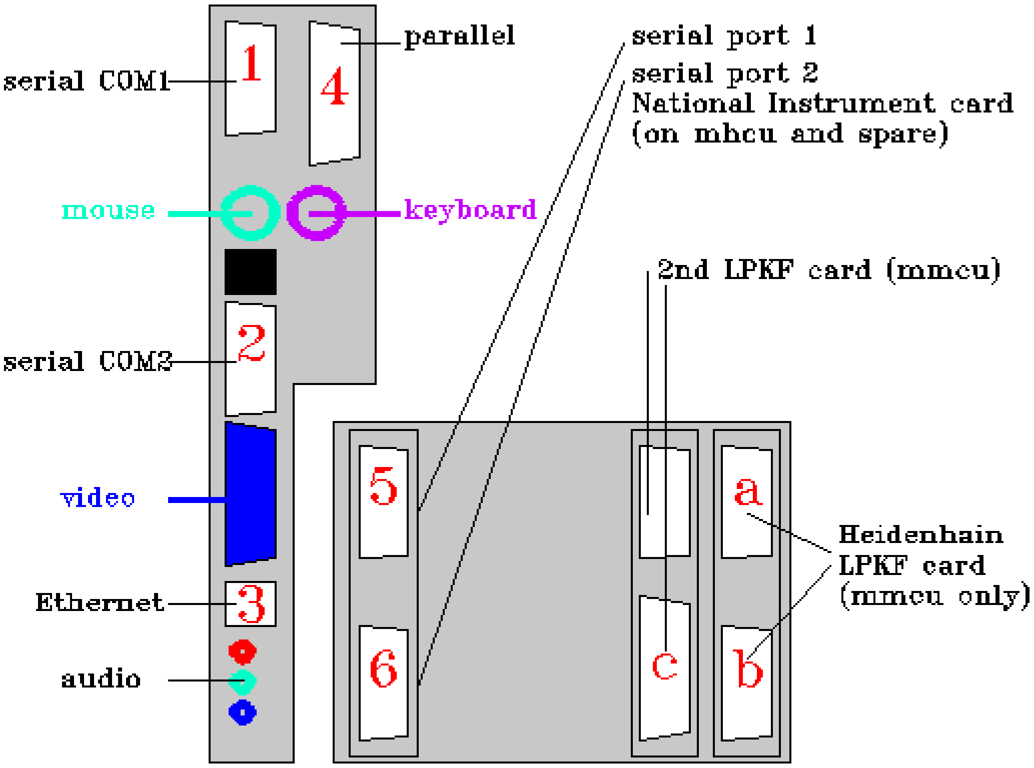
7.1 Port configuration

This might have different details on each machine, which are given in the relevant subsection below. Note that MHCU and `spare` are configured in the same way, but ports are used differently.

The following figure presents a view of the backplane of the computers. Note that the cards in the expansion slots are delivered installed in a different way (as indicated in the figure itself).

In Programs ⇒ Admin Tools ⇒ WinNT Diagnostics ⇒ Resources ⇒ IO Ports; the default `COM1` and `COM2` ports have an entry for range 02F8-02FE and 03F8-03FE serial.

In Programs ⇒ Admin Tools ⇒ WinNT Diagnostics ⇒ Resources ⇒ Devices; clicking on serial, and selecting Properties the default default `COM1` and `COM2` ports have an entry for IRQ 03 and 04with the above addresses.



7.1.1 On MMCU

Two special cards supplied by LPKF have to be installed in the computer backplane. Since one of them (the Heidenhain calibration cards with two connectors) has a non-standard interrupt setting , the following procedure was adopted :

Special note : there are two extra cards and four cables. The cable labelled `PC SL25-1` (with the connector labelled `LASER`) connects the standard `COM2` port, the two cables labelled `PC X-1` and `PC X-2` respectively to the top (a in figure) and bottom (b in figure) port in the rightmost card, and the cable labelled `PC/MBUS` with a 25 pin connector to the lower port (c in figure) in the near-to-rightmost card (the upper port is unused).

The following procedure was applied once by LPKF staff (unfortunately we cannot trace from our notes whether this procedure is applicable to NT or was an LPKF test under Win 95). It shall **not** be necessary to repeat it on other machines (once the jumper on the Heidenhain card is set it is possible to move the card to other machines). Refer instead to the installation procedure described soon afterwards.

- In the BIOS (reachable pressing F2 during the earliest phase of the boot) put serial port 2 to off
- Reboot
- Settings ⇒ Control Panel ⇒ Ports: delete port COM2
- Choose an available (non-conflicting) IRQ level and use the jumper to assign it on the Heidenhain card
- Install the cards
- Reboot so that cards are detected. Set and verify ports as described below in additional hardware configuration
- install LPKF Boardmaster (later replaced by) StencilMaster
- connect the cables from the card to the LPKF rack

- StencilMaster will set some port parameters at first run : anyhow verify ports as described below in additional hardware configuration

The actual procedure necessary to enable the use of the Heidenhain card on a given computer requires the installation and configuration of the Heidenhain IK121 driver. The recommended procedure to do it is the following :

- copy the LPKF StencilMaster installation from another computer (or from the IFCTR backup CD ROM)
- insert the second floppy (2/3) of the Heidenhain IK121 package in drive `A:`
- execute `A:\Install\install.bat` or manually execute the following steps
  - copy the driver `A:\IK121drv\Release\IK121drv.sys` into `C:\Winnt\System32\drivers` (or get it from another machine where it is already installed)
  - copy the library `A:\IK121dll\Release\IK121dll.dll` into `C:\Winnt\System32` (or get it from another machine where it is already installed)
  - click on the register file `A:\Install\IK121drv.reg`

- Switch off and install the cards and connect the cables
- Reboot so that cards are detected. Set and verify ports and driver as described below.

Use Settings ⇒ Control Panel ⇒ Devices to verify that the IK121 driver is present,but is not started and set for `Manual` startup. If the driver is not present, although they relevant files have been copied, it means that the register file on the Heidenhain floppy has not been restored to the registry and/or a reboot has not been done.:

Use Settings ⇒ Control Panel ⇒ Ports to verify/set the port settings :

- three ports should be listed in order `COM3/COM1/COM2`
- the Settings of port `COM1`are 38400 | 8 | None | 1 | Xon/Xoff (although this port is not in use)
- the Settings of port `COM2`are identical (this port is in use)
- the Settings of port `COM3`are 9600 | 8 | None | 1 | Hardware
- the Advanced settings of the same port are :
  - Base i/o address : 3E8
  - IRQ 7
  - FIFO disabled

In Programs ⇒ Admin Tools ⇒ WinNT Diagnostics ⇒ Resources ⇒ IRQ; one will see in addition to the standard devices an entry for the Heidenhain card on. This is hilighted in the table below :

	IRQ	name	
	01	i8042prt	0 ISA
	03	serial	0 ISA
	04	serial	0 ISA
	05	cs32ba11	0 ISA
	06	floppy	0 ISA
⇒	**07**	**serial**	**0 ISA**
	11	El90x	0 PCI
	12	i8042prt	0 ISA
	14	atapi	0 ISA
	15	atapi	0 ISA

In Programs ⇒ Admin Tools ⇒ WinNT Diagnostics ⇒ Resources ⇒ IO Ports; one will see an entry for range 03E8-03EE serial.

In Programs ⇒ Admin Tools ⇒ WinNT Diagnostics ⇒ Resources ⇒ Devices; clicking on serial, and selecting Properties one will see in addition to default IRQ 03 and 04, a port with i/o range 03E8-03EE and IRQ 07.

7.1.2 On MHCU and spare

Use Settings ⇒ Control Panel ⇒ Ports to verify/set the port settings for standard ports (note that some settings are modified later at run time byt the various programs).

- the two standard ports should be listed in order `COM1/ COM2`
- the two additional ports (once installed) should be listed first in order `COM5/ COM6`
- the Settings of port `COM1`are9600 | 8 | None | 1 | None
- the Settings of port `COM2`are identical
- the Settings of port `COM5` and `COM6` are so far identical

Refer to the table above in 7.1.1 for MMCU for standard devices (excepted of course the card present on MMCU only). In addition, once the additional ports have been installed, one will see also the following entries (twice because the additional ports have shared IRQ) :

⇒	**10**	**serial**	**0 PCI**
⇒	**10**	**serial**	**0 PCI**

The following procedure is required to install and configure (once, as `administrator`) the additional serial ports on the National Instrument (NI) PCI 232card (as per NI manual).

- Use Settings ⇒ Control Panel ⇒ Add/Remove Programs to install from floppy the NI software (in `C:\niserial`) and the modified driver (in `C:\WinNT\system32\drivers\serial.sys` ; the old driver saved as `serial.bak` ; note that the uninstall procedure will not put back the original driver)
- install the NI card in the leftmost PCI slot (physical port 1 i.e. `COM5` will be the top one, and port 2 i.e. `COM6` the bottom one in the card)
- Use Settings ⇒ Control Panel ⇒ niports to eventually change the `COM` assignment, and to set the IRQ and addresses (the default should be OK ; note that IRQ 10 appears here as hex A).
- Use instead Settings ⇒ Control Panel ⇒ Ports to verify/set the port baud rate and communictation settings only
- Use Programs ⇒ National Instrument Serial ⇒ Diagnostics to test the new ports

In Programs ⇒ Admin Tools ⇒ WinNT Diagnostics ⇒ Resources ⇒ IO Ports; one will see two entries for addresses DCD8 and DCC8 serial.

In Programs ⇒ Admin Tools ⇒ WinNT Diagnostics ⇒ Resources ⇒ Devices; clicking on serial, and selecting Properties one will see in addition to default IRQ 03 and 04, two ports with i/o range starting at DCD8 and DCC8 and (shared) IRQ 10.

In the case the spare computer or MHCU was tested with the special cards used by MMCU, port `COM3` might appear already defined in Settings ⇒ Control Panel ⇒ Ports but *will not show up elsewhere unless* the cards are physically installed. In the case they have never been tested, it will be necessary to install the special cards, configure port `COM3` in Settings ⇒ Control Panel ⇒ Ports as described in 7.1.1, then reboot as instructed and terminate to verify the settings of all ports. In any case during routine operations the special cards will be installed in MMCU, *not* in spare or MHCU.

7.1.3 On MHCU

The LPKF hardware key necessary to run CircuitCam has to be inserted in the parallel port (labelled 4 in figure).
In addition, to allow full operation (i.e. not just demo mode) of CircuitCam on a machine with the hardware key, it is necessary to install the driver `hardlock.sys` and the library `hlvdd.dll`, and to start the driver. The driver and library can be found in the `C:\LPKF31` directory once CircuitCam has been installed. Use the following procedure to verify and eventually install it.

- Use Settings ⇒ Control Panel ⇒ Devices to verify that the Hardlock driver is present, started and set for `Automatic` startup.

- If the driver and/or library are not present, from a DOS Command window invoke the following command to install, configure and start it. Note the mandatory syntax: `C:\LPKF31\hlinst C:\LPKF31`

The serial ports are used for the following connections :

port	card	location	connected to
COM1	main	top (1 in figure)	IC robot barcode reader
COM2	main	mid (2 in figure)	SC barcode reader
COM5	National	top (5 in figure)	IC robot
COM6	National	bottom (6 in fig)	unused

In addition connect the loudspeaker boxes to the middle (green) audio jack in order to have the beep audible signal.

In addition note that a dedicated power distribution unit (white box) is necessary to distribute power to the two barcodes, while the RS422/332 converter used by the IC robot serial command line (installed on the outside of the cabinet containing the robot control electronics) also needs to be powered, and derives its power from the inside of the robot electronics cabinet.

A description of the pin-out of the various cables follows.

Each bar code reader exits with sub D-25 pin female connector with the pinout described in the Datalogic documentation. This connects to a short stretch of 1:1 cable inside the mounting support (starting with a sub D 25-pin male and ending with a sub D 25-pin female connector). The long cable connecting to the MHCU starts with a 25-pin male connector on the barcode reader side, and ends with a sub D 9 pine female connector on the PC side. This cable also carries the 12 V power for the barcode reader, connected via a further stretch of cable to a 2-pin BNC connector to be attached to the power supply "white box". The pin function is :

barcode side pin	PC side pin	notes
4	1	
2	2	
3	3	
5	4	
7	5	
13	--	+12 V power cable
25	--	0 V power cable

The robot electronics box exits with a RJ 11 cable and a +9 V jack entering an RS 422 to RS 232 converter mounted outside. The switches on the converter must be set so that SW1=DTE, SW2=SIMU, SW3=2. The converter exits with a sub D 25-pin male connector. The long cable connecting to the MHCU starts with a 25-pin female connector on the robot side, and ends with a sub D 9 pine female connector on the PC side. The pin function is :

robot side pin	PC side pin
2	2
3	3
7	5

7.1.4 On spare

TheTaylor Hobson hardware key necessary to run the roughness meter s/w has to be inserted in the parallel port (labelled 4 in figure). In addition the appropriate driver shall be present and active.

- Use Settings ⇒ Control Panel ⇒ Devices to verify that the Hardlock driver is present, started and set for `Automatic` startup.

The version of the hardlock.sys driver (and related DLL) present on spare are different from the one used for Circuit Cam and present on the other two computers (see 7.1.3).They were installed as part of the Taylor Hobson software installation. However they are fully consistent with the use of Circuit Cam, therefore no separate additional installation is necessary.

The serial ports are used for the following connections (the remaining ports are spare) :

port	card	location	connected to
COM1	main	top (1 in figure)	roughness meter

The roughness meter s/w will configure its port using internal configuration data, hence no changes to default settings listed in 7.1.2 are needed.

A description of the pin-out of the cable connecting to the spare computer follows. The cable has sub D 9-pin female connectors at both ends, therefore they are labelled as "PC Comm1" and "Taly Surf blue" (the latter must be connected to the roughness meter connector with the blue cap, i.e. the upper one). The pin function is :

roughness meter side pin	PC side pin
2	3
3	2
5	5
7	8
8	7

7.2 conversion of D and E drives to NTFS

Since the disk partitions are originally set up as FAT, we need to convert the data drives D and E to NTFS in order to be able to use the NT security features (accounts and file protections). The C drive with system software remains FAT.

To convert to NTFS as soon as possible (this is faster on empty disks) do from a DOS window :

```
    convert D :/fs :ntfs/v
    convert E :/fs :ntfs/v
```

Also make sure in the Properties of the Recycle Bin on the desktop that (in the Global settings) one has selected "Configure drives independently" and in the settings for drives D and E one has ticked "Do not move files to the Recycle Bin"

7.3 creation of accounts

- In Programs ⇒ Administrative tool ⇒ User Manager
- create a new account Operator (password assigned, not expiring, is paranal), assign it to the Power User group (no home directory is necessary)
- create a new group Ftpusers (for FTP only users)
- create the accounts vimos, ~~nirmos,~~ fors2 and contingency (password assigned, not expiring, is derived appending ftp to the account name e.g. vimosftp), assign it to the Ftpuser group (no home directory is necessary)
- In the Policies ⇒ User Right panel make sure that the right Access this computer from the network is removed to Everyone, and is assigned to Administrators, Power Users and Ftpusers
- In the Policies ⇒ User Right panel make sure that the right Logon locally is removed to Everyone, and is not assigned to Ftpusers
- the purpose of this is not to allow the ftp-only accounts to log on at console. Since normally this conflicts with user authentication over TCPIP, in order to enforce the above, one need to perform the following registry setting
    - make sure the ftp service is not started
    - run the utility program Regedt32 (make sure you are not in readonly mode)

- select the HKEY_LOCAL_MACHINE panel
- select the key SYSTEM\CurrentControlSet\Services\MSFTPSVC\Parameters
- verify an item LogonMethod is present as a REG_DWORD with value 2
- If not use Edit=>AddValue to enter a value named LogonMethod , of type REG_DWORD and with data value 2
- exit `Regedt32` and eventually restart the ftp service (see 7.6 and 7.7)

7.4 creation of work directories and relevant protections

All standard directories (as described in section 11.4) can be created by administrator (there is no need to log on as the user). The administrator can assign permissions to directories selecting them in Explorer from the Properties⇒ Security ⇒Permission panel. Remove pre-existing permissions and add new as described below.

- The root directory of drive D must have Full Control by `Administrators` and by user `Operator`.
- So will be for the `D:\Mhs` subdirectory and any of its subdirectories.
- Create instead directory `D :\Staging`
- Give it Full Control by `Administrators` and by user `Operator`. Give it List access by the the users `vimos`, ~~`nirmos,`~~ `fors2` and `contingency` (this is necessary to enable ftp access)
- Any further created subdirectory under `D :\Staging` will inherit such setting.
- Hence create four subdirectories for each of the four users users `vimos`, ~~`nirmos,`~~ `fors2` and `contingency` with names `D :\Staging\Vimos`, ~~`Nirmos,`~~ `Fors2` and `Contingency`
- Give it Full Control by `Administrators`, by user `Operator` and by *the relevant one only* of the four users `vimos`, ~~`nirmos,`~~ `fors2` and `contingency` (this way IWS can ftp into there, and operator can remove files and create reports)
- The root directory of drive E must have Full Control by `Administrators` and by user `Operator`.

The above operations shall be made explicitly when creating the directories. The disk mirroring software will not propagate such permissions to directories it creates.

7.5 network configuration

This might have different details on each machine, which are given in the relevant subsection below.

1. In Settings ⇒ Control Panel ⇒ Network ⇒ Ident assign the name (machine specific) and the workgroup as `VIRMOS`
2. In Settings ⇒ Control Panel ⇒ Network ⇒ Services remove the"Client service for Netware". The services to be listed should be only 3Com, Computer Browser, NETBIOS i/f, RPC Conf, Server, Workstation. This has to be done before proceeding to the further step.
3. In Settings ⇒ Control Panel ⇒ Network ⇒ Protocols remove the two entries for NWLink, and add TCP-IP. The protocols to be listed should then be only NetBEUI and TCP-IP. It is likely that you have to exit from the control panel; and enter again in order to be able to configure TCP-IP as follows :
4. In Settings ⇒ Control Panel ⇒ Network ⇒ Protocols ⇒ TCPIP ⇒ Properties configure the items IP address and DNS and leave the rest unchanged. These settings will be modified for Paranal before delivery as indicated by ESO; currently they are :

Menu	Item	Value in Milan	Value in Paranal
IP address	address	*machine specific*	*see next sections*
	subnet	255.255.255.0	255.255.255.0
	gateway	155.253.16.1	134.171.233.254
DNS	hostname	*machine specific*	*see next sections*
	domain	ifctr.mi.cnr.it	pl.eso.org
	nameserver	155.253.16.49 155.253.16.87	134.171.180.99

Note that with this arrangement each machine is able to share and connect (equivalent of export and mount, in Unix parliance) disks over the Microsoft network, and to connect via TCP-IP (ftp, telnet, ping) to other machines, but not to act as ftp server. You should then proceed with the next steps.

7.5.1 on MMCU

Follow the general procedure in section 7.5 with the specific variations described here.

In step 1 Settings ⇒ Control Panel ⇒ Network ⇒ Ident assign the name as MMCU
In step 4 Settings ⇒ Control Panel ⇒ Network ⇒ Protocols ⇒ TCPIP ⇒ Properties configure these items :

Menu	Item	Value	Value in Paranal
IP address	address	155.253.16.251	134.171.233.241
DNS	hostname	mmcu	ommcu

Note that with this arrangement the MMCU is able to share and connect (equivalent of export and mount, in Unix parliance) disks over the Microsoft network, and to connect via TCP-IP (ftp, telnet, ping) to other machines, but not to act as ftp server. Proceed then to the installation and configuration of the ftp server as described in sections 7.6 and 7.7 but at the end make sure the server is not started at boot.

7.5.2 on MHCU

Follow the general procedure in section 7.5 with the specific variations described here.

In step 1 Settings ⇒ Control Panel ⇒ Network ⇒ Ident assign the name as MHCU
In step 4 Settings ⇒ Control Panel ⇒ Network ⇒ Protocols ⇒ TCPIP ⇒ Properties configure these items :

Menu	Item	Value	Value in Paranal
IP address	address	155.253.16.252	134.171.233.240
DNS	hostname	mhcu	ommhcu

Note that with this arrangement the MHCU is able to share and connect (equivalent of export and mount, in Unix parliance) disks over the Microsoft network, and to connect via TCP-IP (ftp, telnet, ping) to other machines. The following further steps described in sections 7.6 and 7.7 are necessary to configure the MHCU as ftp server (in order to communicate with IWSs) : at the end make sure the server **is** started at boot.

7.5.3 on spare

Follow the general procedure in section 7.5 with the specific variations described here.

In step 1 Settings ⇒ Control Panel ⇒ Network ⇒ Ident assign the name as SPARE
In step 4 Settings ⇒ Control Panel ⇒ Network ⇒ Protocols ⇒ TCPIP ⇒ Properties configure these items :

Menu	Item	Value	Value in Paranal
IP address	address	155.253.16.253	134.171.233.242
DNS	hostname	spare	ommsp

Note that with this arrangement the spare is able to share and connect (equivalent of export and mount, in Unix parliance) disks over the Microsoft network, and to connect via TCP-IP (ftp, telnet, ping) to other machines, but not to act as ftp server. Proceed then to the installation and configuration of the ftp server as described in sections 7.6 and 7.7 but at the end make sure the server is not started at boot.

7.6 installation of ftp server

- In Settings ⇒ Control Panel ⇒ Network ⇒ Services add the"Peer Web Server". This module includes an ftp and a lightweight http server. Be sure not to include the (useless) gopher server. It is likely that the installation will automatically configure and start the server. This has to be reset, as described in the next step and section.

- The servers are started and configured in Program ⇒ Microsoft programs ⇒ Internet Server Manager . Since by default both ftp and http daemons are started, it is necessary to configure only ftp to start at boot on the MHCU, and have both stopped on the other machines. This is done in Settings ⇒ Control Panel ⇒ Services where one selects respectively the ftp and httpd/WWW "publishing services", and uses the Startup button to configure them respectively for automatic and manual startup on MHCU, and both for manual startup on the other machines.

7.7 configuration of ftp server

This might have different details on each machine

- In Programs ⇒ Internet Peer Server ⇒ Service Manager make sure you are in the Service View
- Click on the FTP icon, make sure the "traffic light" indicates the ftp server is running (traffic light green)
- If it is not running it start it *temporarily* from the relevant icon in the tool bar
- From the Properties menu select Service properties ⇒ Service
- disable anonymous ftp (and take note of the user account eventually already assigned to it, to be eliminated)
- From Service properties ⇒ Directories add a new ftp root directory on disk `D :\Staging` and define it as Home
- Remove the previous default home (in the meantime re-alasied to a different alias) `C :\InetPub\ftproot`
- make sure the home directory has ftp read and write access
- Make sure the directory and subdirectory have been created as said in 7.4
- Make sure to create the accounts as said in 7.3
- *Interactive access for the four ftp users should be disabled*
- *ftp access to Operator and Administrator should be disabled*
- Note that an ftp connection as one of the ftpuser (e.g. `vimos`) will start in home (e.g. `D :\Staging\Vimos`). In order to write there permissions on this and on the parent directory *must be set* as described in 7.4
- If the FTP server was originally stopped, stop it in Programs ⇒ Internet Peer Server ⇒ Service Manager

7.8 schedule service

In order to allow the disk mirroring program (described in section 6.3 of the Operator Manual) to start for periodic schedule, follow these instructions :

- log on as `Administrator` (both on `mmcu` and `mhcu`)
- From Settings ⇒ Control Panel => Services check that the Schedule service is configured for automatic startup (on MHCU and MMCU) or for manual startup (on spare), using the Startup button
- Then start if if not started already(on MHCU and MMCU)
- From a DOS window issue the `at` command to check that the mirroring program is in the periodic schedule list (normally it should be so automatically even after a reboot)
- if not, invoke the `C:\Mirror\mirrorsetup.bat` batch to insert it in the schedule list.

7.9 timezone setting

Machines have been used with a timezone set to local European time (GMT+1 with DST) which will be changed to UT (GMT without DST) before delivery to Paranal. This is done in Settings ⇒ Control Panel => Date/Time=>TimeZone

The Regional Settings for date and time have to be set in the form yy/mm/dd HH:mm:ss (24 hr) : this is required by the Visual Basic routines used by the IFCTR MHS software.

If it is desired to keep the clocks of the various machines in synch, this can be done at any time issuing as `Administrator` the `net time \\`*`machine`* `/set` command. This is currently done once per day on `mmcu` using `mhcu` as reference machine in the above mentioned `C:\Mirror\mirrorsetup.bat` batch.

7.10 arrangement of disk shares

Each machine shall be configured *once forever* in order to export its D and E drives.

- Log in as `Operator`
- In NT Explorer click once on the D drive icon

- In File ⇒Properties ⇒Sharing select New Share (you cannot use the default administrative share `D$)`
- assign as name `Dclone`
- assign as description `D drive of` *`computername`*
- Click on Permissions, remove Everyone and add Full Control for Authenticated users
- Repeat the same steps for the `E` drive (with name `Eclone` and respective description)

The two active machines (MMCU and MHCU) shall be configured to connect the drive of the other, whenever `Operator` logs on . The configuration is done *once and applies automatically at any further logon*. If SPARE replaces one of MMCU or MHCU, this configuration *must be re-done* afresh.

- Log in as `Operator`
- In NT Explorer select Tools⇒ Map Network Drive
- Enter the path of the remote drive to be connected (see table below)
- Leave the Connect as blank (Operator will connect as Operator, same user and password)
- Make sure Reconnect at logon is ticked

On each of the active machines you should connect the D and E drives of the other machine *in the same order*, as drives G and H (F is the CD-ROM) :

Machine	will see path	as drive
MHCU	`\\MMCU\Dclone` `\\MMCU\Eclone`	`G` `H`
MMCU	`\\MHCU\Dclone` `\\MHCU\Eclone`	`G` `H`

8. Third party software installation

For all these make reference to installation done by LPKF or Taylor Hobson staff, and vendor provided documentation (and standard WinNT "setup"). Customised or modified files will be listed below.
The software described in 8.1 and 8.2 will be installed on all 3 computers delivered (although its functionality will not necessarily be tested everywhere). The remaining support software is delivered installed only on the `spare` computer.

8.1 LPKF Circuit CAM

The installation of CircuitCAM has been performed from floppies on MHCU by LPKF personnel in `C :\Lpkf31`. However the `Ccam/cam.exe` file has to be replaced by the latest version supplied separately by LPKF (3.2 with batch mode support) and which will be made available on a separate medium (which will also contain a `default.cat` file which has to be installed in `New_Templates`). The program does show up in Settings ⇒ Control Panel ⇒Add/Remove Programs. The current default directory (to be changed) for CircuitCAM is associated to the icon in the Properties ⇒ Shortcut panel of CircuitCAM.

Note that CircuitCam requires a hardware key to be plugged in the parallel port of the machine. Also one must perform the license registration described below.
As `Operator` create one or more shortcuts for CircuitCam (from root of desktop).

- Application path is `C:\lpkf31\ccam\cam.exe`
- Path it will start is `D:\Mhs\Vimos\Gbr` for the VIMOS shortcut~~, and `D :\Mhs\Nirmos\Gbr` for the NIRMOS shortcut~~

We have replicated the installation also on `spare` and MMCU. CircuitCAM is fully functional on both once the hardware key is inserted in the parallel port, and the appropriate driver installed (see 7.1.3). Note that on `spare` it uses the hardlock driver for the roughness meter (see 7.1.4). If the hardware key is not installed, Circuit CAM will run in demo mode.

Note that the first time CircuitCam is run by any user, it will display a registration panel. It is necessary to enter the serial number and the enabling number of our license (respectively `981116x01` and `9cf2c038809833c5`).
If you forget to do it, it will go in Demo mode. From here one can invoke the User registration panel in Configuration ⇒ General Settings, enter the number, exit and restart.

Also the first time CircuitCam is run by any user, it will show the full set of toolbars and menus. To assign the simplified *look-and-feel* setting (setup by LPKF in Administrator and saved *into the NT registry*)

- make sure to exit from CircuitCam
- In Explorer go in `C:\LPKF31\RestoreIFCTRConfig` (whose content will be saved to separate medium) and double-click on `NewCircuitCamSavedConfig.reg` (this will copy the settings to the current registry)
- Then restart CircuitCam

The above file has been created by the following procedure (it must be done by the right user, if e.g. the CircuitCam look and feel is customized by `operator`, the file must be created by `operator`, and applied by `administrator`)

- Using program `regedit` select the registry branch `Hkey_Current_user\Software\LPKF`
- export it to a `reg` file

The only **critical** file (of which a backup copy on hard medium will be provided) which contains the current *operational* customisation is `C:\Lpkf31\New_templates\default.cat`. This file contains (in encoded form) all settings accessible via the Config⇒ Job Configuration⇒ *any settings*. Clicking on the third button one can split the screen into an Explorer-like view and navigate through the configuration tree. The configuration items can be accessed clicking either with the left or right mouse button. They are supplied in a stable configuration and are not intended for editing. Relevants items to be noted include :

- the Layers branch just defines the colours used to display the layers in the input Gerber files (see 11.5.3) and the other layers into which they are translated (see here below)
- the Jobs\JobsBatch branch contains the characteristics of the batch mode (see below)
- the Jobs\JobsStencil branch contains items like StencilDefaultSlits and StencilDefaultBarcode. They associate a *base* layer (i.e. input layer defined in the Gerber file) to a *destination* layer (i.e. after the stencil calculation using parameters like the laser beam size)
- the Jobs\JobsOutput\LPKFStencilLaser branch contains items like SlitsCut_CutStd, BarcodeCut_CutStd and CONTOUR. They associate the previously defined *destination* layer (in the case of contours directly the Gerber layer since it does not require stencil calculation) to a cutting phase and to a virtual tool (defined in StencilMaster, see 8.2). The resulting associations are summarized in the following table.

Base layer	destination	output	phase	tool
SLITS	SlitsCut	SlitsCut_CutStd	CuttingSide	StencilCuttingFine
BARCODE	BarcodeCut	BarcodeCut_CutStd	CuttingSide	StencilCutting
CONTOUR	n/a	CONTOUR	Outline	StencilCutting

The configurations saved in default.cat are inclusive of the *arrangement of the batch mode* which are accessible by the Config⇒ Job Configuration⇒ Batch settings. If the sequence has to be changed, save it to another .cat file from the File⇒ SaveAs menu selecting the CAT format. Then, exit CircuitCam, rename the old default.cat into something else and rename the newly saved file as default.cat, and restart CircuitCam to pick it up. A few sample files are available in C:\Lpkf31\New_Templates

In particular nobkg-default.cat allows to run the batch mode with limited interaction. I.e. the user will see any error messages (and has to acknowledge them), then it will be presented the SaveAs...CAM popup. In case of no errors he has to press Save and then OK to acknowledge creation of the LMD file (a feature of the present program is that a CAM file is also created and must be deleted). In case of errors he has to press Cancel and then No to exit

8.2 LPKF Board Master / StencilMaster

The installation of the Boardmaster program (the original LPKF program for controlling the StencilLaser machine) has been performed on MMCU by LPKF personnel in C:\LPKF30 using the supplied floppy and running the Setup utility. The program will *not* show up in Settings ⇒ Control Panel ⇒Add/Remove Programs , which will show just the default stuff. LPKF personnel has then deleted some unwanted .ini files in C :\ lpkf30\bmaster and edited file SL600dcs.mch.

BoardMaster is linked with a 16-bit driver for the Heidenhain card (which is used only in calibration mode), therefore it was *originally* not possible to use the calibration mode of the laser machine under Win NT, and it was necessary to boot Win 95 for this purpose only. Later (April 2000) LPKF supplied a 32-bit version called StencilMaster. The installation was done from a supplementary floppy copying all previous installation in C:\lpkf30\smaster32 and dumping over it the content of the floppy.

We have replicated the installation (copying only the StencilMaster directory from MMCU) also on spare and MHCU. We have tested it (with the physical insertion of the LPKF cards in the backplane) only on spare. Note that the installation also requires the installation of the Heidenhain drivers (see 7.1.1), which has been done on both.

In addition as Operator one can create shortcuts for Boardmaster and StencilMaster manual operation (from root of desktop).

- Application path is c:\lpkf30\bmaster\bmaster.exe bm-eng.txt sl600dcs.ini or respectively c:\lpkf30\smaster32\SMaster.exe sm-eng.txt sl600dcs.ini
- Path it will start in can be set to appropriate subdirectory of E :\ (e.g. for VIMOS ~~or NIRMOS~~ operation)

The main **critical** file (of which a backup copy on hard medium will be provided) which contains the current *operational* customisation is C:\lpkf30\smaster32\SL600dcs.mch which makes reference (includes) the files stencil.phs and stencil.bls (which are *updated at runtime* and describe respectively the colour assignment for the various layers on the screen and the physical characteristics of the virtual tools). The tool characteristics can be accessed via the Config =>Tool Library=> BLS PLS Laser menu. The tool library contains a number of tentative tools plus the *two tools actually used* and associated to the layers defined in the LMD file by CircuitCam (see 8.1), whose characteristics are reported here :

Values used in Milan before installation in Paranal

tool	laser mode	power regulation	power	actual voltage	voltage	frequency	pulse	focus	delay	speed	diameter
`Cutting`	engraving	enabled	*24 W*	*266 V*	*258 V*	*2500 Hz*	*0.077 ms*	0	10 s	*20 mm/s*	0.04 mm
`CuttingFine`	engraving	enabled	*16 W*	*234 V*	*231 V*	*1200 Hz*	*0.19 ms*	0	10 s	*6 mm/s*	0.04 mm

Values set up by LPFK during installation in Paranal (August 2000)

tool	laser mode	power regulation	power	actual voltage	voltage	frequency	pulse	focus	delay	speed	diameter
`Cutting`	engraving	enabled	21 W	269 V	269 V	1600 Hz	0.11 ms	0	10 s	*20 mm/s*	0.04 mm
`CuttingFine`	engraving	enabled	19 W	235 V	235 V	1200 Hz	0.19 ms	0	10 s	*6 mm/s*	0.04 mm

Note that current values may be different if they have been updated by LPKF personnel during maintenance interventions.

Other *runtime critical files* are the various `measure.*` files which are the result of a Calibration operation (see section 9.2.1 of the Operator Manual).

8.3 Taylor Hobson roughness meter s/w

The Taylor Hobson roughness meter s/w includes two modules :

- the standard "Mountain" software used for data analysis, and
- the FTS+ software (developed by the Italian branch of Taylor Hobson) used to acquire data from the TalySurf roughness meter (and as a front end to the analysis software).

Installation has been performed by Taylor Hobson personnel. Installation should occur in 3 steps from CD using standard Windows "setup" as described in the Help file of the FTS+ module. The CD for FTS+ was originally not available and installation was done by floppies using a temporary directory `C:\Master`.
Directories `C:\Program Files\Taylor Hobson` and `C:\Program Files\Ftsp` contain the two software modules. . The programs show up in Settings ⇒ Control Panel ⇒Add/Remove Programs and can be handled via standard NT install/disinstall.

The third installation step is the registration of a software key. While the above installation must be done from Administrator, the registration of the key has to be repeated also as Operator. One uses the `WinProt32` program (present on the FTS+ CD) to obtain a first key, then phones Taylor Hobson to obtain a second key and types it in, then transfers the protection. Keys used were : SNDGC and McbV] for Administrator and tSYRh and MhxPF for Operator.

Additional installation/configuration tasks done were :

- In `C:\Program Files\Ftsp\Plog` change the value with a single zero (0)
- In the FTS+ Setup panel verify the serial port settings
  - number of serial port 1
  - configuration of serial port 19200,n,8,1
  - handshaking of serial port 2

- In the FTS+ Setup panel change the file path and program path : the WinNT installation is arranged for the Italian version and presets programs in `C:\Programmi` instead of `C:\Program Files`
- The installation procedure predefines the two programs to be launched as Start ⇒ Programs ⇒ Ftsp and Start ⇒ Programs ⇒ Taylor Hobson....1.3 ⇒ TalyProfile for the Administrator. To propagate this to the `Operator` copy the file `Ftsp` and the content of the entire directory `Taylor Hobson TalyProfile 1.3` from the directory `C:\Winnt\Profiles\Administrator\StartMenu\Programs` to the same location of `C:\Winnt\Profiles\Operator`
- We have further edited the Properties of the shortcuts for the FTSP and Mountains programs so that both programs start in directory `C:\Roughness` (where all roughness measurement data will be stored).

- The original messages in the program were in Italian. We have translated into English those indicated to us by Taylor-Hobson staff (the `.dat` and `.msg` files for which a backup `Copy of ...` exists).

8.4 Barcode reader s/w

Additional software installed in `C:\Barcode` is used to load (once) the hardware configuration of the barcode reader in its EEPROM. It is installed via standard setup from floppies supplied by Datalogic.
It installs by default in `C:\Barcode` while it also installs a shortcut in Start ⇒ Programs ⇒ Datalogic. The installation has been done by `Operator` as the operator may need to use such program in the rare eventuality the EEPROM configuration has to be reloaded (see section 9.4).

8.5 Microsoft Visual Basic development environment

Installed from Microsoft CD using standard setup information. Refer to section 9.3 for side effects of this installation.

The Microsoft Visual Basic CD automatically starts the setup program when inserted. It is generally sufficient to select the "typical" installation the default action and to restart the computer when instructed . Peculiarities to be noted are :

- The product number to supply when prompted is `045 0032367`
- The procedure will also install an upgraded version of Internet Explorer
- It is suggested to disable the active desktop update feature when requested
- The directory for installation chosen is `C:\Development`
- A separate disk is required to install the MSDN help library. Select the typical installation.

9. Special installations

It was our intention to keep the three MMU computers as identically configured as possible (inclusive of having the same operational software installed on all three). However during development and testing there were situation which forced us to introduce deviations from the basic configuration. This is recalled here below.

9.1 On MMCU

Since the original installation (March 1999) and until April 2000 BoardMaster did not support the full calibration protocol using the Heidenhain card driver under Windows NT. This was implemented only with the 32-bit StencilMaster version.. To overcome this, MMCU was *provisionally* been set-up for dual booting with Windows 95 and this feature has not been removed. The calibration had to be run under Windows 95.

The configuration for dual boot was done following the instructions in the NT FAQ :

1. Make an emergency repair disk (rdisk - Update Repair Info)
2. Ensure you have NT installation disks (can make by winnt32 /ox)
3. Reboot the machine and boot off a bootable floppy disk with the drivers for your CD-ROM drive (supplied by G.Bonelli and available c/o G.Conti) and run setup.exe off of the CD-ROM
4. Install Windows 95 as normal
5. Once the Windows 95 installation has finished, reboot the machine and boot off of the NT installation disks
6. After second disk will give options, press R for repair
7. Deselect all options except "Inspect Boot Sector" and continue
8. Press Enter to detect hardware and insert disk 3
9. The procedure will ask if you have an Emergency Repair Disk (ERD), say Yes and insert the ERD.
10. The machine will then reboot into NT again
11. Once in NT goto a DOS session
12. Type - `attrib c:\boot.ini -r -s`
13. edit `boot.ini` and insert at the bottom `C :\=Microsoft Windows`
14. Type - `attrib c:\boot.ini +r +s`
15. Reboot the machine and you will have Windows95 and NT options.

As a result the arrangement of the system disk C : is at the moment different from the other computers :

- There is a directory `Ntstart` missing elsewhere

- The `Program Files` directory is called `Programmi` (sic !)
- There is a `Win95` directory

At boot time, immediately after the BIOS screen, one is presented with three options instead of the usual two variants of Windows NT. The third option is for using BoardMaster calibration mode. *It should be no longer necessary to use this mode in the future*.

9.2 On MHCU

MHCU is delivered in the basic configuration. No peculiarities are present.

9.3 On spare

The spare computer has been subject to additional configuration, mainly for installation of software required to be delivered only on this machine (see 8.3, 8.4 and 8.5). In particular

- The installation of Visual Basic has automatically installed a new version of the Internet Explorer and updated a number of other DLL libraries. This has also altered the desktop settings, and the look and feel of Control Panel, logout, etc
- the audio drivers have been updated (during an attempt to resolve a conflict for the installation of the National Instrument serial cards).
- For local convenience we have installed printer drivers for our local printers
- For our convenience we have also installed the VNC software (used to "repeat" the screen of a PC on a Unix workstation for producing the screen dumps included in this manual), the Olympus camera software and the Pine mail user agent under a dedicated ifctr account. All this stuff can be kept or removed at discretion of ESO after the commissioning.

9.4 Barcode reader configuration

The configuration of the EEPROM of the barcode readers shall be done only once. It has been done before delivery, and should not be done again unless the EEPROM configuration is lost (or a new barcode reader is acquired). The configuration shall be done on the `spare` computer.

- Make sure the barcode reader you want to configure is connected to the serial port 2 of the `spare` computer, and powered on.
- Start the configuration program as Start ⇒ Programs ⇒ Datalogic ⇒ Winhost
- The program will automatically open the DEVICE CONTROL panel. Click on the button (at right end of top row) to establish the serial connection.
- If the program can recognize the barcode reader configuration, it will open also the DS2100 EDIT CONFIGURATION panel with the current configuration (if this happens everything is fine, and probably no maintenance action is necessary). If it is not so, one can open such panel clicking on the EDIT icon (right end of the screen).
- The correct configurations are saved on a disk file, load them from disk into the DS2100 EDIT CONFIGURATION panel using the button (at left end of top row). You shall choose one of the following two files from the `DS2100` directory:
  - `virmosnw.cfg` for the two barcode readers labelled with fabrication date November 1999
  - `virmosol.cfg` for the barcode reader labelled with fabrication date August 1999

- You shall normally not need to edit the configuration, but just use the saved one.
- To load the configuration into EEPROM, first select EEPROM instead of RAM as destination, using the toggle button at the right end of the bottom row of the commands section in the DEVICE CONTROL panel.
- Then click the SEND button (middle in top row of the commands section in the DEVICE CONTROL panel) and reply OK to the confirmation popup

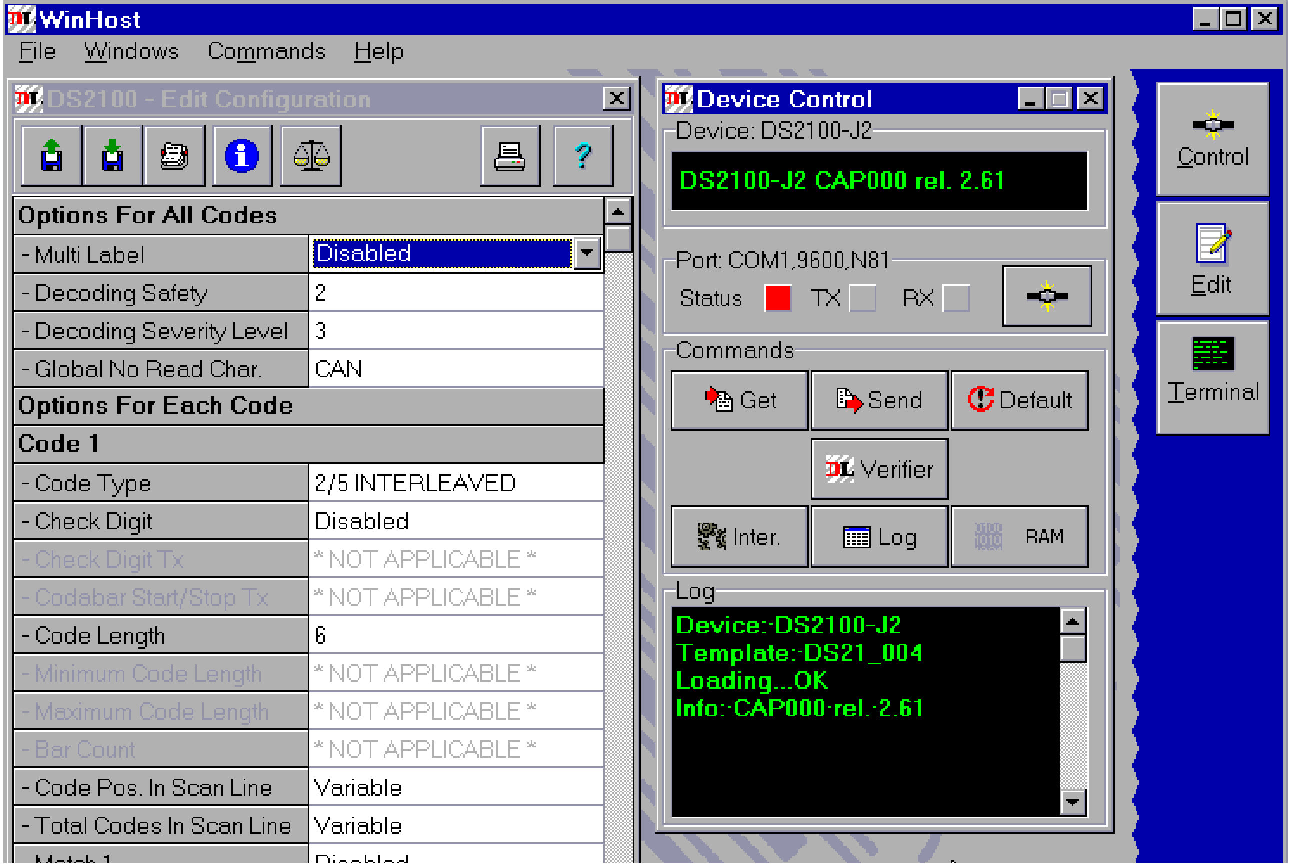
- If wished you may test the reader calling the TERMINAL panel (click on the icon of same name). It has two small start and end buttons in the bottom right corner. A successful test implies :
  - a start echoes a `<STX>` to screen. Wait until the barcode reader times out, and this will be followed by a response containing the "null response string" `nnnnnn`.
  - a further start echoes a `<STX>` to screen. Place a valid barcode in front of the reader and verify you obtain a response containing the correct mask id.
  - repeat the last step once more (you should be able to read a new mask id just issuing an `<STX>` without `<ETX>`
  - give an end, which echoes a `<ETX>` to screen. This terminates the test.

- Once the test is successful, EXIT from Winhost, and install the barcode reader where appropriate

The bar code configuration used for VIRMOS differs from the default one only in a few points. The default configurations are stored in the files `default2.cfg` and `default4.cfg`, the correspondence with the barcode readers is as follows.

Fabrication label	August 1998	November 1999
Device	DS2100-I2	DS2100-J2
template	DS21_002	DS21_004
firmware release	1.16	2.61
default configuration	`default2.cfg`	`default4.cfg`
VIRMOS configuration	`virmosol.cfg`	`virmosnw.cfg`

The fabrication label can be looked on the body of the barcode reader itself.



Options For All Codes	
- Multi Label	Disabled
- Decoding Safety	2
- Decoding Severity Level	3
- Global No Read Char.	CAN
Options For Each Code	
Code 1	
- Code Type	2/5 INTERLEAVED
- Check Digit	Disabled
- Check Digit Tx	* NOT APPLICABLE *
- Codabar Start/Stop Tx	* NOT APPLICABLE *
- Code Length	6
- Minimum Code Length	* NOT APPLICABLE *
- Maximum Code Length	* NOT APPLICABLE *
- Bar Count	* NOT APPLICABLE *
- Code Pos. In Scan Line	Variable
- Total Codes In Scan Line	Variable
- Match 1	Disabled

Device: DS2100-J2

DS2100-J2 CAP000 rel. 2.61

Port: COM1,9600,N81

Status 🟥 TX □ RX □

Log

```
Device:DS2100-J2
Template:DS21_004
Loading...OK
Info:CAP000 rel. 2.61
```

There is a compare button in the DS2100 EDIT CONFIGURATION panel which allows to compare configurations with the same template. The parameters changed for VIRMOS are (in the order they occur) :

- Decoding safety, changed from 1 to **2**
- Code 1 type, changed from default to **2/5 Interleaved**
- Code 1 number of digits (data length) changed from variable to **6** characters
- Data Format, General, data length, also changed from variable to **6** characters
- ibidem, fill character changed from space to a lower case **n**
- Operating mode, mode changed from Online to **Serial Online**
- Operating mode options, serial online timeout changed from disabled to **4s**
- Reading parameter, overflow changed from default to **30.4 µs**

9.5 MHS robot test software

We deliver together with the IFCTR software described in the next section a couple of Visual Basic programs (one developed by Antil and one developed by us) which can be used for testing and exercising the command interface to the MHS robot. Their usage shall be self explanatory : they allow to send individual commands on the serial line. It is not normally intended to use them, nor it is normally possible (since the programs reside in the development environment on `spare`, while the robot is connected to MHCU), hence we do not provide further documentation.

The executable of our program is also temporarily installed on MHCU in `C:\RobotTest\robot_test.exe` to support calibration of IC slots position (see 11.5.6).

We also deliver one further Visual Basic program (to be installed on `spare` only and to be run in the Visual Basic development environment) called `port simulator`, which can be used to simulate the communication protocol and operation of the barcode readers and of the robot in software. This program has been used in Milan for support and testing of new versions of MHS : we connect the 3 cables from the ports of the computer running the MHS to another computer running the port simulator. The same arrangement could in principle work connecting the MHCU with `spare`, and running the port simulator on `spare`. Internal documentation on the port simulator will be provided separately.

10. IFCTR software installation

The MHS, Cut Manager and disk mirroring software developed at IFCTR in the Microsoft VB environment will be delivered installed on all 3 computers. In addition a "kit" will be delivered on CD.

The kits have been created with the VB Package Wizard tool. Essentially they consist of a CAB file and a setup script. The CAB (cabinet) file is logically the Microsoft equivalent of a compressed tar file including the binary executable and all associated dynamic libraries. The setup script will perform a standard WinNT installation, which typically occurs in two stages : first some system libraries, which may be used by more than one application, will be updated once forever (it will **not** be possible to uninstall them), then another set of libraries and the program executable will be installed (these could be uninstalled at wish).

The source directory is `C:\Development\Microsoft Visual Studio\VBPrograms\`*`name`* where *`name`* is one of `Mhs`, `CutManager` or `Mirroring`. There is also a subdirectory `Robot` containing the two undocumented utility programs used for MHS robot testing (see 9.5).

The procedure for generation of a kit, and its installation are described in next section 10.1. These procedures will normally never be necessary, unless the Operating System is installed afresh on a computer.

In case of modifications to the program sources, it is not necessary to do a full install, but it is sufficient to replace the executables already existing on all 3 computers,i.e. files `mhs.exe, cut.exe` and `mirroring.exe` respectively in directory `C:\MhsSW`, `C:\MmuSW` and `C:\Mirror`.

10.1 kit generation procedure

Special actions are required to install MHS (or CutManager or Mirroring) executables generated by Visual Basic, which are recalled here :

- On `spare` there are three steps necessary for the first time ; next times only the first one is required :
  1) Generate the executable : within Visual Basic from the File menu select `make mhs.exe` (or `cut.exe` or `mirroring.exe`)
  2) Generate a distribution kit : Programs ⇒Visual Basic ⇒Tools ⇒Package Wizard : select a project among one of the three source directories quoted in 10 , then select the Package icon and follow instructions, selecting the default (only for the name of the final script you may want to specify the name of the package) with one exception (for the mirroring package only you must add to the list of components besides the `.exe` the library `C:\Winnt\System32\Scrrun.dll` which is not included by default). This will do the following things :
     Under the project location create a `Package` directory, and in this create a `Support` subdirectories, and place there all the libraries (DLL etc.) needed to run the exe, all setup tools, a DDF file listing all the components, a `makecab` BAT file needed to combine all the above in a "cabinet" file
  3) Run the above script and under the `Package` directory create a CAB file (like a tar file, containing all the above in compressed form) and a setup executable and listing
     From the Package Wizard : select the Deploy icon and specify a location on a local folder (currently we use . `C:\Development\Microsoft Visual Studio\VBkits\`*`packagename`*) to make the kit available (this copies the a CAB file, setup executable and listing to a "public" location)

- On `mhcu` and `mmcu` there are the following steps necessary :
  4) Connect the network drive `\\spare\c$,` say this comes out as drive `k:.`
     The first time the operation has to be done from `Administrator`. The other times it can be done by `Operator`.
     If you do it for the first time do it from `Administrator` and check in reconnect at login
     If you do it as `Operator` make sure you connect as `Administrator` with the right password

2a) The first time you need to install the package (e.g. MHS, or Cut Manager or Mirroring). To do this go on the folder where the kit was deployed (which in the example of MHS will appear to you as `k:\Development\Microsoft Visual Studio\VBkits Mhs`) and execute the Setup program.

- Setup will first of all update some system libraries once forever (this **cannot** be uninstalled), currently e.g. they are in the case of MHS (all in `c:\winnt\system32`) :

```
vb6stkit.dll
comcat.dll
asycfilt.dll
olepro32.dll
oleaut32.dl
msvbm60.dll
stdole32.tlb
```

- Then the specific setup will run, this will copy some more system libraries in the same location, set the appropriate registries, plus installing the MHS, Cut Manager or Mirroring executable where you specify (recommended is `C:\MhsSW` or respectively `c:\MmuSW` or `c:\Mirror`). All this stuff can be uninstalled using the Add/Remove Programs tool in the Control Panel The libraries are e.g. for MHS :

```
mscomctl.ocx
comdlg32.ocx
msstdfmt.dll
mscomm32.ocx
msinet.ocx
```

- Finally setup will create an entry for the program in the Start menu. To propagate this copy the directory from`c:\Winnt\Profiles\Administrator\StartMenu\Programs\Mhs` (or whatever) in the same location of `c:\Winnt\Profiles\Operator` or even better use as location `c:\Winnt\Profiles\AllUsers` and remove it from the other places. This arrangement (as well as the desktop icons) can be freely modified at discretion of the users. However we have prepared a common simplified setup for all users, which clearly groups system software, IFCTR VIRMOS software, LPKF software and other software (the latter on `spare` only). .

2b) All following times you just need to copy only the `exe` file from drive `k` to `C:\MhsSW`
*) For CutManager steps are similar except that directory is `C:\MmuSW`
*) Similarly for Mirroring steps are similar except that directory is `C:\Mirror`

11. Programmers' reference

The MHS, Cut Manager and Mirroring software has been developed at IFCTR and the relevant programmer information useful for maintenance is supplied in the next three sections. All other sofware is third party proprietary software which cannot be modified.

The software developed at IFCTR uses the Microsoft Visual Basic 6.0 (VB) development environment. The relevant source files are kept in form of *VB projects* which include a `vbp` file, one or more `bas` files and a collection of `frm` and `frx` files. Usually all above files are accessed and managed exclusively through the VB development environment. The `bas` and `frm` files are source files. In particular the latter contain both the description of the visual arrangement of the forms, and the code of the associated routines.

11.1 MHS routine description

There is only one module `Module1` in this package in file `mhsModule1.bas`
In addition there are several forms in this package : the form named *`xyz`* is in file `mhs`*`xyz`*`.frm`

11.1.1 basic module

`Module1` contains public variable and constant declarations (including two boolean flags `debugcrash` and `debugrobot` used to enable additional debugging options if the sources are recompiled into a new executable, or if the program is run in the development environment, in addition a further boolean flag `icalternate` may be used to alter the policy for the filling of IC cabinets in the load phase of IC preparation, see 11.1.6.1) and the following general purpose routines

11.1.1.1 SafeKill

A call to routine `SafeKill(`*`filepath`*`)` deletes the file(s) referred to by *`filepath`* if at least one exists, and silently returns no errors if no files have to be deleted.

11.1.1.2 LogWrite

A call to routine `LogWrite()` will append a normal log entry to the ops-log file. The routine has no arguments. The action and comment texts must be assigned to the public (global) string variables `field2` and `field3`. The routine takes care of formatting the log record according to specifications (see 11.5.7).

11.1.1.3 ErrWrite

A call to routine `ErrWrite()` will append an error or warning log entry to the ops-log file. The routine has no arguments. The `/ERROR` or `/UNFORESEEN` or `/RECOVERY` keyword and associated text must be assigned to the public (global) string variable `field2`. The routine takes care of further formatting the log record according to specifications (see 11.5.7).

11.1.1.4 BLWrite

A call to routine `BLWrite()` will append a blank line to the ops-log file. The routine has no arguments.

11.1.1.5 Crash

A call to routine `Crash()` (with no arguments) is used to simulate a system crash for testing purposes. This version also enters an error record in the ops-log file. Disabled in the operational version. Called from the crash button if this is enabled (which occurs setting `debugcrash` to True)

11.1.1.6 ClearLast

A call to routine `ClearLast()` (with no arguments) clears the internal buffer used to store the "last action" information.

11.1.1.7 WriteLast

A call to routine `WriteLast()` (with no arguments) dumps the internal buffer used to store the "last action" information to the `last_prog.txt` file. Such buffer is a public array of 5 strings `last(5)`

11.1.1.8 DisableMDIForm and EnableMDIForm

These (argumentless) routines are used respectively to disable the "always active" buttons of the master form (i.e. those associated to VIEW ORDERS, VIEW LOG and LAST ACTION) in critical cases or re-enable them once the critical situation is over. The critical cases are those when an external port (barcode or robot) is active and is not desired to interfere with such operations. It is necessary to disable the entire form, since clicking on any part of the form will declare it active and confuse the routines which make reference to the "active form" for displaying messages.

11.1.1.9 ConnectBarCode1 and ConnectBarCode2

A call to these (argumentless) twin routines is used in the initialization stage of any form which will use either barcode readers (number 1mounted to the MHS robot stand, and number 2 mounted on the SC) to establish and test the communication. They will open the appropriate port, and send a predefined (enter programming mode, an arbitrary command which requires an answer) hardcoded command, waiting for a reply within a 3 sec timeout. If communication is established another predefined (exit programming mode) hardcoded command is sent, otherwise the operator is duly notified and an appropriate message logged. A POPUP allows to retry after appropriate corrective actions suggested in the NEXT ACTION box have been taken.

11.1.1.10 ConnectRobot

A call to this (argumentless) routine is used in the initialization stage of any form which will use the MHS robot to establish and test the communication. They will open the port, and send an hardcoded status request command , waiting for a reply within a 3 sec timeout. If communication is not established the operator is duly notified and an appropriate message logged.

11.1.1.11 introduction to MHS robot control

The MHS robot Z-displacement unit is controlled by an APA controller, which talks with our s/w over a serial line. The full details are given in ANTIL documentation [ref. 10, programming annex].. There are a number of binary commands, built by a sequence : `SOH` *`opcode [operands] checksum`* `EOT`. To all commands it is foreseen a valid reply of an `ACK` or an invalid reply of a `NAK`, with the exception of the status command, which returns instead *`SOH statusbyte checksum`* `EOT`. The status byte contains an error code : code 1 indicates the machine has just been switched on and has not yet been homed, code 2 means the machine is ready, code 3 the machine is busy executing a command, codes 4 and 5 correspond to recoverable conditions (set by the photo cell or the emergency stop button), and codes 6 to 10 are in principle hardware unrecoverable errors. Our higher level routines take care of issuing the appropriate commands, verifying their acknowledgment, and eventually the status, and dealing with it.

Basic robot low level commands are as follows. The *speed*, *acceleration* and *home* commands are used typically only at initialization. The typical robot movement is commanded by a *move* command followed by a sequence of *status* commands to verify the movement has ended correctly. There are two error conditions (codes 4 and 5) returned by the status command which have to be cleared by a *reset* command, they are respectively the cases that the photo-cell has revealed an obstacle to the movement, or that the emergency stop button has been pushed. In such cases one must first remove the cause (remove the obstruction, or pull out the emergency button) and only then the reset command will be effective (our software will assist in the right sequence of operations). There is also a *stop movement* command, which is not used by our software.

We have combined the basic low level command in four tiers of routines, such that the software usually deals only with the top tier routines. The four tiers are the following :

1. High level commands : home (see 11.1.1.12), speed or acceleration setting (see 11.1.1.14), or generic movement (see 11.1.1.13). Their generic structure is described immediately after this list.
2. High-level status request (see 11.1.1.15), called by the previous tier, which will try to cope with most error conditions and recover them transparently to the caller.
3. Error recovery routines, called by the previous routine in the attempt to clear the error condition together with the operator (or anyhow to notify the operator).

4. The lower level can be called by all the above to send directly a status request (see 11.1.1.19), or any low level command (see 11.1.1.20), without any additional protocol (but handling communication errors).

The typical protocol implemented in the high level routines is the following :

1. send a high level status request (*status before*), to test if the robot is in condition to proceed processing the *particular* command (this includes both a machine ready status, or a cleared recoverable error, and deals also exit on unrecoverable errors)
2. in affirmative case, proceed to load the command opcode and use the low tier routine to send it
3. in case of a (communication) failure here, exit immediately
4. enter a loop reading the status (with the lowest level request) until this returns any status but the busy one
5. send another (though maybe redundant) high level status reques (*status after*). In case of cleared recoverable errors, it will go back to step 2 and retry execution of the command. In cases of success or unrecoverable failure, it will use the public `status` variable to return control to the caller.

The responsibility to continue operations or to terminate (suspend, abort) the current MHS function belongs to the caller. To do this it has to evaluate the `status` variable, which, at this stage, can have only the values `2` (success, machine ready) or `0` (catch all for any error including communication or undefined errors). Any other (non-zero) values are native robot status codes used only during interim low level calls, and not directly visible in raw form to the software or the operator. The operator is however constantly informed on the last robot status via a message in clear in an appropriate text area.

11.1.1.12 RobotHome

This upper-tier routine is called as `RobotHome(`*`ntime`*`)` whenever it is necessary to command the robot to the home position (i.e. reset the reference zero of the coordinates to the point sensed by the proximity switch). The routine is compliant with the generic protocol described in 11.1.1.11. The peculiarities of this particular call are that it also updates the robot position text area on the screen, and that it skips the entire operation if called with *`ntime`* `=1` and the *status before* indicates that the home command was already sent.
The sense of all this is that the home command must be the *first* command sent after a function requiring the robot is entered, *only if* necessary, i.e. after robot switch on (the robot will retain the home condition if not switched off). However the home command is *unconditionally* sent in the case the IC holder has just been loaded on the Z-displacement unit platform (since the zero under load might be different from the zero of the empty platform).

11.1.1.13 RobotMove

This upper-tier routine is called as `RobotMove` whenever it is necessary to command the robot to any position (typically an IC slot, or the loading/unloading lowermost position). The position to be reached is not passed as an argument, but preloaded by the caller in the public variable `sposta`. The routine is compliant with the generic protocol described in 11.1.1.11. The updating of the robot position text area on the screen is responsibility of the caller.

11.1.1.14 RobotSpeedAccel

This upper-tier routine is called (*once* at start of each function requiring the robot) as `RobotSpeedAccel(`*`opcode`*`)` to assign the initial setting of the speed (*`opcode`* `=2`) and acceleration (*`opcode`* `=3`). The value to be set is not passed as an argument, but preloaded by the caller in the appropriate public variable. The routine is compliant with the generic protocol described in 11.1.1.11. This routine sets the robot position text area on the screen to `UNDEFINED`, since an error here may leave it in an unpredictable position, but it will soon be updated by any forthcoming movement command..

11.1.1.15 RobotAskStatus

This second-tier routine is called as `RobotAskStatus(`*`IgnoreHomeError,statusreply`*`)` by the upper-tier routines. It will call the lower tier routine (see 11.1.1.19) to obtain the decoded and reclassified status (in public variable `status`), and call the appropriate error recovery function (or perform appropriate action) as detailed below.

- `status=1` (no home position). The recovery action is to call `RobotReHome` (see 11.1.1.17) and retry unless the logical variable *`IgnoreHomeError`* is set to `True`. There are in fact cases (regulated by the calling upper-tier routine) in which this status is not an error (when the RobotHome command is called the first time after a switch on this is the normal condition, while the home setting is irrelevant for the RobotSpeedAccel calls).

- status=2 (robot ready). No (recovery) action required ! Normal status !
- status=3 (robot busy). This should not happen, since it should be shielded by the upper-tier routine protocol. However if it occurs recovery is handled internally (offering a choice to the operator to retry or abort).
- status=4 (recoverable error). The recovery action is to call RobotReset (see 11.1.1.16) and retry.
- status=5 (unrecoverable error). The recovery action is to call RobotHardErr (see 11.1.1.18). There is no command retry in this case..

The return value *statusreply* may assume one of three symbolic values (in addition note that the public variable status maintains whatever interim value assigned by intervening calls) :

- replyNotOK if an unrecoverable error occurred (or was declared so by operator action). This includes unpredictable (communication) errors which will leave status=0.
- replyOK if the robot is immediately ready to accept a command
- replyCleared if a recoverable error occurred and was cleared by software assisted operator intervention

11.1.1.16 RobotReset

This third-tier recovery routine handles a POPUP asking the operator whether to try to clear the recoverable error and retry the high level command interrupted by the error, or to abort entirely the calling program. Any action is dealt with by the callers, the routine will just set the status public variable, and *in any case* use the low level SendCommand routine to send a robot reset command (opcode=7). Note that this must be sent *also* in case of abort (otherwise the error will show up as not cleared when resuming).

11.1.1.17 RobotReHome

This third-tier recovery routine handles a POPUP asking the operator whether to try to restore the home position and retry the high level command interrupted by the error, or to abort entirely the calling program. Any action is dealt with by the callers, the routine will just set the status public variable, and *if retry requested* use the low level SendCommand routine to send a robot home command (opcode=5).

11.1.1.18 RobotHardErr

This third-tier recovery routine handles just a POPUP asking the operator to acknowledge the unrecoverable error. Program abort will be dealt with by caller.

11.1.1.19 RobotStatus

This lowest-tier routine sends a status command (opcode=6, value directly hardcoded on the output port), then waits for a status reply (also dealing with a possible communication error with a 3s timeout). It will also decode, interpret and classify the status reply. In particular it will display a message in clear in the robot status text area on the screen. The classification of the status reply sets the status public variable as follows, grouping together different error statuses according to severity. Any illegal or unexpected reply will set status to zero.

- status=1 corresponds to status reply 1 : the robot home position has not yet been set
- status=2 corresponds to status reply 2 : the robot is ready to accept commands
- status=3 corresponds to status reply 3 : the robot is busy processing commands (typically, moving)
- status=4 groups together status replies 4 and 5 (i.e. typical recoverable errors, either the emergency stop button has been pressed, or the photocell detected an obstruction to the movement). It will also group status reply 9 (unlikely to occur, two commands sent simultaneously), which shares with the other the property to need a reset command to be cleared
- status=5 groups together all remaining status replies, corresponding to unrecoverable hardware errors.

This routine is affected by the debugrobot flag which may be used to induce simulated errors under user control if the program is recompiled from sources or run in the development environment.

11.1.1.20 SendCommand

This lowest-tier routine deals with the sending of all operational (i.e. not status) commands. It expects the opcode lodaed in the public variable com_code and the eventual argument in other public variables. The home and reset commands

(opcode=5,7 respectively) are hardcoded directly, while the movement, speed and acceleration, which accept arguments, are coded explicitly including checksum calculation.
The encoded command is then sent on the output port, and the program waits for an ACK reply. In this case the public status variable is set to 2 (successful). Any other case sets it to zero, and is dealt as an error. The operator is notified here, while the abort condition is dealt with by the caller. Recognised errors at this level are :

- a communication error (no response from robot controller within the 3 s timeout)
- a syntax error (a NAK reply, indicating that the controller [thinks to have] received a wrongly formatted command
- an handshaking error (any other illegal reply from the controller)

This routine is affected by the debugrobot flag which may be used to induce simulated errors under user control if the program is recompiled from sources or run in the development environment.

11.1.2 MDIForm1

This module defines the master form used to start the program. Its graphical layout is exemplified in section 6.1.5 of the Operator Manual. In addition it contains the following routines associated to form events.

11.1.2.1 MDIFormLoad

Called when the form is loaded to do necessary initializations.

11.1.2.2 Command1Click(Index)

Associated to the CONVERT, STORE, IC PREPARATION, DISCARD, RECOVERY and EXIT buttons, manages (shows) Forms 1,2,3,5,6 and termination via the EXIT button

11.1.2.3 But2ViewOrders_Click

Associated to the VIEW ORDER button, shows the relevant form

11.1.2.4 But3ViewLog_Click

Associated to the VIEW LOG button, shows the relevant form

11.1.2.5 But4Last_Click

Associated to the LAST ACTION button, shows the relevant form

11.1.3 Form1

This module defines the convert form. Its graphical layout is exemplified in section 6.1.6 of the Operator Manual. In addition it contains the following routines associated to form events, and the additional service routines. Historically convert was called produce (hence this prefix is used in many instances).

11.1.3.1 Form_Activate

Called when the form is activated (loaded in memory and shown, or anyhow passed control : note that the choice between a Form_Load or Form_Activate routine, here and elsewhere, has been done by experience in a way that does not interferes with other events, ant that the two are not interchangeable).

- Verifies presence of an MMJ and of file producecrash.txt.
- If a MMJ exists and no crashed convert session stops after acknowledgment (this means the store phase was active on such MMJ).
- If no MMJ and no crashed session exist log a normal start.
- If MMJ and crashed session exist means convert is resuming (log appropriate entry).
- The file producecrash.txt contains a single numeric flag, which indicates at which stage convert was crashed (or suspended but not aborted : an abort implies the MMT report is sent, hence the program can't be resumed).

- If the MMJ is null (refers to zero masks) send an MMT report and stop immediately
- In all cases the routine enables or disables buttons as appropriate.
- If resuming from crash also calls `RetrieveSct` and `RefreshList` described below.

11.1.3.2 But6Crash_Click

Used to invoke the `Crash` function from the relevant button (used for testing only).

11.1.3.3 But1Retrieve_Click

Handler for the RETRIEVE button.

- First of all shows the `ManOrders` form.
- If this required a cancellation (termination) calls `ProduceEnd` to terminate
- Otherwise logs the start and records it to the last action buffer
- If no orders were loaded terminates on error, after acknowledgment, via `ProduceEnd`
- Moves MMJ from staging to work area (terminates on error)
- Moves MSF files one by one from staging to work area (on error notifies, and after acknowledgment skips affected file
- Updates the numeric flag in `producecrash.txt` , enables or disables buttons for the next phase, and writes an hint to the "NEXT ACTION" box.

11.1.3.4 But2Translate_Click

Handler for the TRANSLATE button.

- Loops on the list of MSFs
- Opens the MSF and decodes it (non-existence or i/o error on the file causes skip on acknowledgment, the same occurs for *some* syntax errors which are checked)
- Reads the `offsets.txt` file and adds appropriate offsets to all slit coordinates
- Creates the Gerber file and writes the fixed heading to it, followed by the definition of standard apertures, and by an aperture definition statement for each standard slit. This and the next item now handle MSFs defining also round and step-and-repeat slits (used for maintenance masks) besides normal rectangular slits.
- Then writes the Gerber statements to flash the slits
- If there are curved slits call `BEZIER` to handle them
- Open the `holes_i.gbr` and copy the Gerber code for the mask holes (depending on quadrant *`i`*)
- Open the `refpins_i.gbr` and copy the Gerber code for the mask reference pins (depending on quadrant *`i`*)
- Call `BARCODE` to write the Gerber statements for the bar code
- Open the `contour_i.gbr` and copy the Gerber code for the mask contour (depending on quadrant *`i`*)
- Record successful conversion to Gerber into log file and `temp_rep.txt`
- Updates the numeric flag in `producecrash.txt` , enables or disables buttons for the next phase, and writes an hint to the "NEXT ACTION" box.

11.1.3.5 BEZIER

This routine generates the Gerber code for curved slits. Each slit is defined by a sequence of connected points. The points are calculated from the slit parameters in the MSF as Bezier parametric polynomials. More precisely the "long sides" of the slit are two identical Bezier curves translated in Y by the slit height.

The connected points are equispaced in the Bezier parameter `t` (ranging 0.0 to 1.0). The number of steps is at least 20 (of the slit length is less than 10 mm), 40 if the length is between 10 and 20 mm, 80 if it is longer than 20 mm. The length is approximated as the sum of the distances between the initial point and the middle point and between the middle point and the final point, where the middle point is defined as the one for t=0.5 (see below).

The Bezier polynomial is defined as a parametric curve (`t`=0.0-1.0)

- $x(t) = a_x x^3 + b_x x^2 + c_x x + x_0$
- $y(t) = a_y y^3 + b_y y^2 + c_y y + y_0$

The above curve runs in the middle of the slit in field of view coordinates. In practice the upper edge of the slit is described by y(t)+ $d_y/2$ and the lower edge by y(t)-$d_y/2$. Furthermore, both x(t) and y(t) are translated in mask coordinates adding the offsets from the `offsets.txt` file.

Refer e.g. to the Postscript Language Reference Manual [ref. 5] pag. 393 for a description of the usage of Bezier curves.

11.1.3.6 BARCODE

This routines generates the Gerber code for the bar code corresponding to the 6-digit mask identifier *qnnnnn* according to the 2/5 interleaved code type. In this representation each digit 0-9 is represented by a code of 5 bars (two wide and three narrow ones) according to a lookup table. Two digits are interleaved, in the sense that the code for one is represented as white space, and the code for the other as dark space.

11.1.3.7 But3Submit_Click

Handler for the SUBMIT button.

- Loops on the list of Gerber files (exits after acknowledgment if there aren't any - in this case the entire `convert` session will be terminated)
- Loads the relevant information in the `CirCamFiles` screen areas and invokes `CirCaMFiles` to process the next file
- If it required a suspend, handle it as if END button was pressed (via `But4End_Click`)
- Otherwise `Shells` the CircuitCam program (in batch mode passing to it the Gerber file name, or in interactive mode without argument) to do the conversion to LMD and waits (via a POPUP) for an acknowledgment to continue
- Record successful conversion to LMD into log file and `temp_rep.txt` ; in case of unsuccesful conversion a POPUP asks to retry the conversion of skip this mask.
- Enables or disables buttons for the next phase, and writes an hint to the "NEXT ACTION" box.

11.1.3.8 RetrieveSct

This service routines opens the SCT files and loads their content in the appropriate list box. It also keeps a counter of masks in SCT.

11.1.3.9 RefreshList

This service routines opens the temporary report `temp_rep.txt` , which keeps track of which processing has been applied to which mask, and updates accordingly the list boxes with the MSF, Gerber and LMD file names.

11.1.3.10 But4End_Click

Handler for the END button.
Will recognise if the request is for a termination of the convert phase, or a suspension : in the latter case will log a pause entry to the ops-log file, and update the last action buffer.
Real termination (after operator acknowledgment) is handled by the next routine.

11.1.3.11 ProduceEnd

This routine handles also the logging of abort requests (`convert` is assumed to be aborted if terminated before Gerber or LMD conversion). The reason to have separate routine is that the handling of the end condition may be requested not only by the END button handler, but also as a consequence of other contingencies.
If the program is not aborted or suspended it removes the `producecrash.txt` file, calls `MaskSetCopy` to move LMD files to MMCU, and logs a stop entry to ops-log and last action buffer.
In all cases unloads the convert form and passes control back to the MDI FORM..

11.1.3.12 MaskSetCopy

This service routines reads the the temporary report `temp_rep.txt` , copies the LMD files to MMCU (also logging this appropriately to the ops-log file) and updates `temp_rep.txt`. If a mask set is incomplete (i.e. not all LMDs listed in the MMJ order have been produced) it is considered unsuccesful. No LMD files are copied to MMCU (the version on MHCU will be deleted later anyhow with all others) so that it will not be possible to manufacture a partial mask set. An error message is given to operator, and duly logged in log file.

This routine has been rewritten redefining the concept of incomplete mask set (originally defined as a full set of exactly 4 masks) to support operation with less than 4 quadrants. Contextually it has been assumed that an MMJ order will contain exactly one mask set.

11.1.3.13 But5Batch_Click

This routine handles the toggle button between BATCH MODE and INTERACTIVE MODE (the current value is the button label, and the status is recorded in a boolean variable)

11.1.3.14 WriteReport

This service routine is called as `WriteReport(`*`mmj`*`)` where *`mmj`* is the MMJ file name without the `.mmj` extension. It is used in all cases where an MMT report is required when `convert` is aborted. Besides copying the MMT to the staging area for MCS, it gets rid of any intermediate files (which are normally deleted at the end of the `store` phase)

11.1.3.15 MSF parsing routines

This is a set of three routines used to decode coordinate lines in MSFs. The `read4` routine (with 4 arguments) reads a line with the 4 coordinates describing a normal (rectangular) slit. The `readmany` routine (with 13 argu ments) reads a line with the 13 coordinates describing a curved slit. Both routines call a `parse` service routine to extract a single coordinate from the MSF line read in as a string.
The reason to use this arrangement is that the default VB input statements will use free-format reads which, in case of missing coordinates, will wrap to the next file line. Here we use the `Line Input` VB statement, forcing read of exactly one line at a time. The `Val` statement will assign a zero value to missing coordinates, which may be trapped by the caller.

11.1.4 Form2

This module defines the `store` form. Its graphical layout is exemplified in section 6.1.7 of the Operator Manual. In addition it contains the following routines associated to form events, and the additional service routines

11.1.4.1 Form_Activate

Called when form is activated (see 11.1.3.1 for comments on activate vs load).

- Verifies presence of the `producecrash.txt` file. If present means `convert` has not yet terminates, and therefore exits after acknowledgment via `StoreEnd`
- Verifies presence of the `storecrash.txt` file. If present means it's resuming from a crashed session.
- In all cases log *appropriate* entries to the ops-log and last action files.
- In all cases the routine enables or disables buttons as appropriate.

11.1.4.2 But5Crash_Click

Used to invoke the `Crash` function from the relevant button (used for testing only).

11.1.4.3 But1Initialize_Click

Handler for the INITIALIZE button.

- First of all verifies the presence of the `temp_rep.txt` file. If this is missing it means all masks requested in the MMJ have been stored, therefore exits immediately after acknowledgment via `StoreEnd`
- otherwise disables the END button, opens the port to the SC bar code reader and verifies communication with it is correctly established using routine `ConnectBarCode1` (see 11.1.1.9)
- reads the content of the `temp_rep.txt` file, counts the number *`n`* of masks which have a valid LMD and updates their number of the screen in the form *`n`* `of` *`m`*, where *`m`* is the total number in the order
- Loads the content of the SCT in memory and in the appropriate list
- otherwise updates the `storecrash.txt` file, enables or disables buttons for the next phase, and writes an hint to the "NEXT ACTION" box (this is different according to the case there is already at least one mask to store or not. The latter case might occur when resuming from a crash)

11.1.4.4 But2Store_Click

Handler for the NEXT MASK button

- prompts to place a mask on the decoding stand and calls `code_reader`
- if a suspension of `store` was requested by such routine skip the following 9 steps (and then exits calling `TrueSuspend`)
- log the barcode just read
- if the bar code has not a legal format log an error message and skip after acknowledgment
- open `temp_rep.txt` and verify the bar code corresponds to a mask in the present MMJ ; if not log an error message and skip after acknowledgment
- if the bar code points to a mask apparently already stored in the session (i.e. the mask is listed in MMJ and SCT) a POPUP suggests a course of action and requires an acknowledgment
- if the bar code points to a mask apparently already stored in a previous session (i.e. the mask is listed and flagged in the temporary report and SCT) a POPUP suggests a course of action and requires an acknowledgment before terminating via `StoreEnd`
- instructs to store the mask in the appropriate cabinet and wait for acknowledgment
- in case of missing acknowledgment (mask damaged) one may other retry to store it (obviously after manufacturing a good one) or skip storage so far (with the idea of storing it later in the same session).
- Increase tally count of stored masks, and update SCT in memory, on disk and on screen (list box)
- update temporary report and log appropriate entries to ops-log
- formats a different message for the "NEXT ACTION" hint box, and enables or disable buttons appropriately (handling separately the cases whether there is still at least one mask to store or all have been stored)
- updates the `storecrash.txt` file

11.1.4.5 But3Suspend_Click

Handler for the SUSPEND button

- a POPUP asks confirmation for exit
- calls `TrueSuspend` in all cases except if the bar code reader is active in the reading loop
- enables the other buttons in the MDI Form

11.1.4.6 TrueSuspend

This service routine closes the communication port used for the bar code reader (this is a protected close, i.e. is not issued if the port is not open), unloads the store form and logs a pause entry in the ops-log and last action files.

The reason this routine is separate is that the bar code reader loop was initiated by a routine called by a specific button handler, and the termination must be handled gracefully even if the suspension is required by a different button handler. If this were not done mishaps would occur if store is re-entered without quitting MHS.

11.1.4.7 But4End_Click

Handler for the END button.

Real termination (after operator confirmation, handled by a single or double POPUP, according to the fact all masks in the MMJ have been processed or not) is handled by the next routine

11.1.4.8 StoreEnd

The reason to have a separate routine is that the handling of the end condition may be requested not only by the END button handler, but also as a consequence of other contingencies (including the case described in 11.1.4.6).

Deletes the MMJ, MSF, Gerber and LMD files from the work area. Also in case of incomplete mask sets deletes any LMD files left in the MMCU incoming area. Copies the termination report to a newly created directory in the staging area, copies the SCT to the same staging area, deletes temporary files, logs appropriate entries to ops-log, closes the communication port for the bar code reader, and logs a stop entry to ops-log and last action buffer.

Finally unloads the `store` form and passes control back to the MDI FORM.

11.1.4.9 UpdateCrashFile

A trivial service routine which logs the name of the last mask stored to the `storecrash.txt` file.

11.1.4.10 code_reader

Service routine which puts the bar code reader in wait until a string validly formatted as a bar code is read. In case a suspension of the `store` program was requested, also gracefully terminates the bar code readout.

11.1.5 Form3

This module defines the `unload` form. Its graphical layout is exemplified in section 6.1.8.1 of the Operator Manual. In addition it contains the following routines associated to form events, and the additional service routines

11.1.5.1 Form_Load

Called when the form is loaded for simple tasks (the operator has anyhow to use the RETRIEVE button to start the bulk of the operations) .

- Verifies the presence of an MIJ in the MHS work area
- If an MIJ is present assume program is resuming from a crash or suspension and set appropriate flag.
- In all cases the routine enables or disables buttons as appropriate.

11.1.5.2 But4Crash_Click

Used to invoke the `Crash` function from the relevant button (used for testing only).

11.1.5.3 But1Retrieve_Click

Handler for the RETRIEVE button.

- If this is a fresh start (not a resume), first of all shows the `InsOrders` form.
- If this required a cancellation (termination) calls `UnloadEnd` to terminate
- If no MIJ orders were selected also force termination via `UnloadEnd`
- If a fresh order has been chosen, move MIJ from staging to work area
- In all cases logs the start or resume to the ops log and records it to the last action buffer
- Loads the ICT and SCT tables in memory and in the appropriate lists
- *Calls the `ICChange` form (see 11.1.23) to force the operator to check which ICs are in use for each quadrant (original or spare set), which implies loading the appropriate robot position table.*
- Reads the MIJ and create a list of masks with an associated flag for the termination report, initially set to zero
- Perform an analysis comparing the MIJ with the ICT content and create the list of masks to be unloaded (also counts if there are any to be loaded). This is preceded by a check that the masks listed in the MIJ are present either in ICT or in SCT. If they are nowhere an appropriate message is issued which allows to take recovery actions. Also updates the temporary report for any mask already present in the ICs. Establish boolean flags whether there are any masks to be unloaded and loaded. *Slots declared as out of use are duly ignored.*
- If unload is not necessary warn the operator and call `UnloadEnd` to terminate (and start loading)
- otherwise disables the END button, opens the ports to the SC bar code reader and the robot and verifies communication with them is correctly established using routines `ConnectBarCode1` (see 11.1.1.9) and `ConnectRobot` (see 11.1.1.10)
- Execute initial robot settings : home, setting speed and acceleration, move platform to loading position to insert the IC holder, and re-make the home position. All according to operator replies to appropriate questions
- Finally enable the buttons for the next phase

11.1.5.4 But2Unload_Click

Handler for the IC UNLOADING button.

- Since this is a closed loop, disable the END button
- make a loop quadrant by quadrant, and a loop slot by slot inside the quadrant
    - move the robot Z-displacement unit to place the next slot containing a mask to be unloaded in front of the decoding stand. *Empty and out of use slots are ignored.* Note that the movement always occurs in the *upward* direction, either directly or in two steps. If the previous recorded position is in the opposite direction, the robot is first commanded to a position 1 cm below, and then reaches the final position from below. A (null) movement is commanded even if the last recorded position is the same as the next desired position.

- the operator is instructed to extract the mask from the IC (if this is not possible a dialogue allows to skip the processing of the mask if the slot has been emptied by purpose, e.g. by a failure of the MEU at the telescope, or to force a suspension/abort of `unload`, after acknowledgment, via `UnloadEnd` in any other case; note that since unloading occurs on a list of masks generated by the program, and not explicitly contained in an MIJ order, these events are not flagged as errors in the MIT report, but only logged to the ops-log)
- calls `code_reader` to read the bar code of the extracted mask
- if a suspension of unload was requested by such routine exit via `UnloadEnd`
- if the bar code does not match the expected value a suspension/abort of `unload` is forced, after acknowledgment, via `UnloadEnd`)
- similarly occurs if the extracted mask appears to have the same id of a mask already stored in the SC(T)
- in case of successful verification the operator is instructed to store the mask in the SC. The SCT is updated only after operator confirmation. If a mask is damaged (to be re-made) it is removed from SCT.
- The ICT and SCT tables on disk are updated calling `Update_ict_sct`

- At the end of the loop, if the MIJ does not require to load any mask, the robot Z-displacement unit is moved to the unloading position to allow to remove the IC holder from the platform
- Finaly the END button is re-enabled

11.1.5.5 But3End_Click

Handler for the END button.
Real termination is handled by the next routine. This might occur silently if processing was aborted before loading an MIJ, after operator acknowledgment in case of normal termination, or after operator confirmation in case of premature termination (with the options of suspending `unload` or continuing it cancelling the termination request)

11.1.5.6 UnloadEnd

The reason to have a separate routine is that the handling of the end condition may be requested not only by the END button handler, but also as a consequence of other contingencies.
The routine closes any port (barcode or robot) which is still open, then it might be exit silently if processing was aborted before loading an MIJ.
If termination was forced by a robot error jumping directly in here, it offers the possibilities of suspending or really terminating `unload` in a hard way (sending the MIT report and deleting the MIJ) setting the appropriate flag.
The routine will log the appropriate stop or pause message, unload the `unload` form (sic!) and pass control back to the main (MDI) form.
If the MIJ does not require to load any new mask in the ICs, or if a hard termination was requested on a robot error, the routine deals with the generation and sending of the MIT report, and of the updated SCT and ICT, and with the removal of the MIJ. Otherwise it unloads the `unload` form (sic!) and loads the `load` (Form4) form.

11.1.5.7 Update_ict_sct

Service routine which updates the ICT and SCT files on disk with the content of the tables in memory. This (and other) routine is also capable of handling the possibility that less than 15 slots be used in the ICs and takes appropriate actions if one changes from operation with 15 slots to and from operation with less than 15 slots (altering the configuration file defined in 11.5.6).

11.1.5.8 WriteTempRep

Service routine which dumps to the temporary report file `temp_rep.txt` the current status of all masks (stored or not stored) from memory, in order to make sure the information on disk is up to date.

11.1.5.9 code_reader

Service routine which puts the bar code reader in wait until a string validly formatted as a bar code is read. In case a suspension of the `unload` program was requested do not perform any bar code readout.

11.1.6 Form4

This module defines the `load` form. Its graphical layout is exemplified in section 6.1.8.2 of the Operator Manual. In addition it contains the following routines associated to form events, and the additional service routines

11.1.6.1 Form_Activate

Called when the form is activated, performs the following initialization tasks :

- does some trivial button and flag initialization
- logs a start message (`load` never resumes directly from a crash or suspension but is always invoked after `unload`)
- disables the END button, opens the ports to the SC and IC bar code readers and the robot and verifies communication with them is correctly established using routines `ConnectBarCode1|2` (see 11.1.1.9) and `ConnectRobot` (see 11.1.1.10)
- Loads the ICT and SCT tables in memory and in the appropriate lists
- Reads the MIJ and ticks (on the screen) the masks already present in IC since the unload phase
- If this was not already done in the unload phase, execute initial robot settings : home, setting speed and acceleration, move platform to loading position to insert the IC holder, and re-make the home position. All according to operator replies to appropriate questions
- Reads the existing temporary report and performs a preliminary analysis of the content of the ICT
- A substantial piece of code has been added at this stage to change the IC filling policy. This code is conditionally enabled at compile time if the boolean `icalternate` is set to True in `Module1` (see 11.1.1). With the old policy (`icalternate` set to False) all free IC slots are filled in their sequence order. With the new policy, every other free slot is filled as far as possible, in order to leave free slots around each mask. This is arranged in detail performing a recursive analysis with routine `Analyse` (see 11.1.6.13) and simulating an optimal filling of the slot with routine `Filler` (see 11.1.6.14). Only slots flagged as "to be filled" are then considered for filling (and will be filled in their sequence order from 1 to 15 to optimize robot movements). *Slots declared as out of use are handled identically to empty slots,i.e. are not eligible for filling.*
- Finally enable the buttons for the next phase

11.1.6.2 But5Crash_Click

Used to invoke the `Crash` function from the relevant button (used for testing only).

11.1.6.3 But2Load_Click

Handler for the NEXT MASK button.

- enables and disables appropriate buttons
- make implicitly a loop quadrant by quadrant, and a loop slot by slot inside the quadrant (since the routine is event driven by the button, it has to check at each call where it is in the implicit loop)
  - for each mask to be loaded, verify if the mask already loaded
  - move the robot Z-displacement unit to place the next empty slot where the mask must be loaded in front of the decoding stand. Note that the movement always occurs in one direction as explained in 11.1.5.4.
  - starts a search loop, putting the SC bar code reader in wait until a string validly formatted as a bar code is read. The loop is terminated by a flag set by the MASK FOUND or MASK NOT FOUND buttons. All this is handled by routine `code_reader2`.
-
- If the loop is over, the robot Z-displacement unit is moved to the unloading position to allow to remove the IC holder from the platform
- Finally the END button is re-enabled

11.1.6.4 But3MaskFound_Click

Handler for the MASK FOUND button. This routine just sets a flag `found=yes`. It might invoke directly the actual handler `TrueFound`, however in general such handler is not invoked here but at the end of `But2Load_Click` after a return from `code_reader2`. This prevents mishaps which may occur if a routine started by a button handler is terminated by an event generated in another handler when `load` is suspended and then resumed without exiting MHS.

11.1.6.5 TrueFound

Actual handler for the Mask Found condition.
Sets the flag which interrupts the SC bar code reader loop started by `But2Load_Click`, and logs a message about the mask found, then asks to operator to verify the mask is placed undamaged in the MHS robot stand (and handles logging in case of errors).
It then calls `code_reader` to identify the mask again using the IC bar code reader
It handles and logs any error in this stage, as well in the next when the operator has to confirm insertion of the mask in the IC. Of course it will also log successful insertion.
Finally updates the report and the ICT and SCT tables on disk using calls to `Write_rep` and `Update_ict_sct` and enables buttons for the next mask.

11.1.6.6 But1NotFound_Click

Handler for the MASK NOT FOUND button.
Handles all operator dialogue and logging for this situation, and also updates the ICT and SCT tables on disk calling `Update_ict_sct` and enables buttons for the next mask

11.1.6.7 But4End_Click

Handler for the END button.
Real termination is handled by the next routine. This routine just takes care of asking a confirmation to the operator, and to set all necessary flags in order to inerrupt any eventual pending read on either barcode reader.

11.1.6.8 LoadEnd

The reason to have a separate routine is that the handling of the end condition may be requested not only by the END button handler, but also as a consequence of other contingencies.
The routine closes any port (barcode or robot) which is still open.
If termination was forced by a robot error jumping directly in here, it will offer the operator the possibilities of suspending or really terminating `load` in a hard way (sending the MIT report and deleting the MIJ) setting the appropriate flag.
The routine will log the appropriate stop or pause message, unload the `load` form and pass control back to the main (MDI) form.
In cas a hard termination was requested, the routine deals with the generation and sending of the MIT report, and of the updated SCT and ICT, and with the removal of the MIJ. Additionally it unloads the `load` form (sic!).

11.1.6.9 Update_ict_sct

Service routine which updates the ICT and SCT files on disk with the content of the tables in memory.
This (and other) routine is also capable of handling the possibiity that less than 15 slots be used in the ICs and takes appropriate actions if one changes from operation with 15 slots to and from operation with less than 15 slots (altering the configuration file defined in 11.5.6).

11.1.6.10 WriteRep

Service routine which dumps to disk the temporary report file `temp_rep.txt`

11.1.6.11 code_reader

Service routine which puts the MHS stand bar code reader in wait until a string validly formatted as a bar code is read. In case a break or suspension is requested for any contingency it returns to the caller, which deals with the break.

11.1.6.12 code_reader2

Service routine which puts the SC bar code reader in search until the requested bar code is read, and/or an external condition stops the loop. The condition is the change of a flag `found` from `undef` to either `yes` or `no`. In such case it returns to the caller.

11.1.6.13 *Analyse*

This service routine.is called once for each quadrant to be analysed, and must be called repeatedly after each (simulated) slot filling (change of the `load_ic` array). It assigns weights to all slots in a quadrant according to the following rules :

- A filled slot is assigned a weight –1 (occupied)
- An empty slot surrounded by two filled slots is assigned a weight 0 (less preferred)
- An empty slot with just one adjacent slot free (except for the cases described below) is assigned a weight 1
- An empty slot with both adjacent slots free (or the first and last slot if the adjacent one is free) is assigned a weight of 2 (preferred)
- *Out of use slots are handled exactly as empty slots (therefore they should be best physically emptied)*

11.1.6.14 *Filler and findbest*

The `Filler(`*`quadrant`*`,newslot)` service routine returns the number *`newslot`* of the best slot to be filled for a particular quadrant, and sets a flag "to be filled" in the appropriate array. To do this it calls the auxiliary routine `findbest` which returns the first free slot with the required weight. This routine is called in turn from the highest (2) to the lowest (0) weight until one free slot is returned. Implicitly calls the `Analyse` routine for update.
All this in practice means that, if there are slots surrounded by both sides by free slots they are filled first. Of course this can occur only if the MIJ order requests less than 8 masks per quadrant and the IC is empty. Otherwise, if more masks are requested and in relation with the number of masks already present in the IC, the slots with one adjacent free slot will be used, and finally the remaining slots.

11.1.7 Form5

This module defines the `discard` form. Its graphical layout is exemplified in section 6.1.9 of the Operator Manual. In addition it contains the following routines associated to form events, and the additional service routines

11.1.7.1 *Form_Activate*

Called when form is activated, performs the following initialization tasks :
Verifies presence of an MDJ

- If no MDJ exists log a normal start.
- If an MDJ is there means there was a crashed session and discard is resuming (log appropriate entry, and set a flag).
- In all cases the routine enables or disables buttons as appropriate.

11.1.7.2 *But6Crash_Click*

Used to invoke the `Crash` function from the relevant button (used for testing only).

11.1.7.3 *But1Retrieve_Click*

Handler for the RETRIEVE button.

- First of all disables the END button, opens the port to the SC bar code reader and communication with it is correctly established using routine `ConnectBarCode1` (see 11.1.1.9)
- if the crash flag is not set (i.e. if starting afresh) shows the `DiscOrders` form.
- If the latter required a cancellation (termination) calls `DiscardEnd` to terminate
- If no orders were loaded terminates on error, after acknowledgment, via `DiscardEnd`
- Moves MDJ from staging to work area
- Loads the content of the SCT and ICT in memory and eventually in the appropriate list box (in the case of SCT; the ICT is just used for checking if masks not found in SCT are there)
- Retrieves the content of MDJ as a sorted list of masks (using a VB feature to sort a list box) and initializes the temporary report
- If the MDJ order is null (no masks to be discarded) handles all actions for immediate termination.
- if suitable flag reports previous session was crashed after order retrieval, recover the temporary termination from disk
- otherwise (normal start) dumps temporary report to disk with `StoreReport`, enables or disables buttons for the next phase, and writes an hint to the "NEXT ACTION" box

11.1.7.4 But2Discard_Click

Handler for the DISCARD button (manually operated for each next mask, variously labelled FIRST MASK or NEXT MASK).

- Looks for the next mask to be discarded (the routine keeps an internal tally). If what is thought to be the next mask is already flagged as discarded in the temporary report, writes a message and exits
- Otherwise verify presence of mask to be discarded in SCT and ICT
- If not found in SCT because already in ICT displays an error POPUP and, after acknowledgment, skips the mask
- If not found in SCT displays a warning POPUP, assumes mask was already discarded, deletes the LMD file, updates all tables as in case of successful discard and exits
- Otherwise puts the bar code reader in search mode, and enables the appropriate buttons. When the wished bar code is found, issues a BEEP and disables the MASK NOT FOUND button. Otherwise the operator must use such button to force termination of the search mode and invoke the appropriate handler.

11.1.7.5 But3MaskFound_Click

Handler for the MASK FOUND button, pressed by the operator to acknowledge a mask has been located. *This routine just sets a flag found=yes. It might invoke directly the actual handler `TrueFound`, however in general such handler is not invoked here but at the end of `But2Discard_Click` after a the interruption of the bar code reading loop. This prevents mishaps (see discussion in section 11.1.7.6 immediately following).*

11.1.7.6 But4MaskNotFound_Click

Handler for the MASK NOT FOUND button, pressed by the operator to terminate the bar code search mode if a mask cannot be located. This routine just sets a flag `found=no`. It might invoke directly the actual handler `TrueNotFound`, however in general such handler is not invoked here but at the end of `But2Discard_Click` after a the intrruption of the bar code reading loop. This prevents mishaps which may occur if a routine started by a button handler is terminated by an event generated in another handler when `discard` is suspended and then resumed without exiting MHS.

11.1.7.7 TrueNotFound

Actual handler for the Mask Not Found condition.

- A POPUP requires to try again the search or to cancel it
- If the search is retried, writes appropriate log entries and enables or disables buttons for another search, then exits
- If the search is cancelled, a POPUP asks whether to skip the mask or to suspend discard for offline recovery
- In the case of skip, it writes appropriate log entries and enables or disables buttons for another search, then exits
- In the case of suspend, calls `DiscardEnd` to terminate

11.1.7.8 UpdateSCT

This service routine is called whenever necessary to update the content of the SCT. It locates the record for the current mask to be discarded in the in-memory copy of the SCT, it replaces it with the empty record. It writes the SCT to disk, then re-reads it to update the list boxes. And of course writes the appropriate log entries

11.1.7.9 But4End_Click

Handler for the END button.

Will recognise if the request is for a termination of the discard phase, or an abort (defined as an end before MDJ retrieval) : in the latter case will log an abort entry to the ops-log file, and update the last action buffer.

Real termination (after operator confirmation, handled by a single or double POPUP, according to the fact all masks in the MDJ have been processed or not) is handled by the next routine.

11.1.7.10 DiscardEnd

The reason to have a separate routine is that the handling of the end condition may be requested not only by the END button handler, but also as a consequence of other contingencies.

Copies the SCT and ICT to the staging area. Also copies the termination report to a newly created directory in the same staging area, deletes all work files, closes the communication port for the bar code reader, and logs a stop entry to ops-log and last action buffer.

In all cases unloads the discard form and passes control back to the MDI FORM.

11.1.7.11 CheckForLast

This service routine is called whenever a mask is discarded. It verifies whether this is the last of the order. It formats a different message for the "NEXT ACTION" hint box, and also eventually disables all buttons other than END.

11.1.7.12 StoreReport

This service routine is called whenever the status of a discarded mask changes, and writes on disk an up to date copy of the termination report (flagging appropriately masks already discarded).

11.1.7.13 TrueFound

Actual handler for the Mask Found condition.

- A POPUP requires acknowledgment of the actual mask discarding
- Takes care of appropriate log entries
- deletes the LMD files
- updates internal tables and reports, and enables or disables buttons for the next phase

11.1.8 Form6

This module defines the RECOVERY FUNCTION MASTER FORM. Its graphical layout is exemplified in section 6.1.10 of the Operator Manual. This form is a plain dispatcher, it contains a trivial `Form_Activate` and `But5End_Click` routine, and other click routines associated to buttons 1,2,3,4,6,7,*9 and 10* which respectively call `ScanSC`, `ScanIC`, `Insert`, `Remove`, `Identify`, `Search`*, and* `TablesView`, `SlotsOutOfUse`
In addition the `But8RobotDown_Click` handler contains the hardcoded commands (reset, home and move to unloading position) necessary to position the Z-displacement unit platform to the lowest position (so that the IC holder can be removed) in case it is left in any other position. The commands are sent to the port directly using the lowest level protocol and no library routine. There is minimal error checking : establishment of the serial communication is checked issuign a status read with timeout only *before* sending the other commands.

11.1.9 CirCamFiles

This module defines the information form used to assist in Gerber-LMD conversion. Its graphical layout is exemplified in in section 6.1.6 of the Operator Manual.. It contains three trivial routines

- `Form_Load` is called when the form is loaded, and just fills in the instrument name.
- `But1Start_Click` associated to top button (variously labelled START, CONTINUE or END under control of the caller form), closes the picture panel and hides the form
- `But2Suspend_Click` associated to bottom (SUSPEND) button, hides the form setting a suspend flag.

11.1.10 DiscOrders

This module defines the DISCARD ORDER SELECTION FORM. Its graphical layout is not shown, but similar to the other selection forms described in 11.1.13 and 11.1.18. Like them, it contains a `Form_Activate` routine, which fills a selection list box with a directory listing of MDJ files present in the staging area, and two trivial `OKButton` and `CancelButton` handlers.

11.1.11 Identify

This module defines the OFF-LINE MASK IDENTIFICATION FORM. Its graphical layout is exemplified in section 6.1.10.6 of the Operator Manual. In addition it contains a trivial `Form_Activate` routine, and the additional click and service routines

11.1.11.1 But1Initialize_Click

This is an ultra- simplified version of the routine of same name described in 11.1.12.1

11.1.11.2 But2Identify_Click

This is a simplified version of the routine described in 11.1.12.2

11.1.11.3 But3End_Click

A trivial routine which closes the connection to the bar code reader search and exits.

11.1.11.4 code_reader

Identical to the routine described in 11.1.12.7

11.1.12 Insert

This module defines the SINGLE MASK INSERTION FORM. Its graphical layout is exemplified in section 6.1.10.3 of the Operator Manual. In addition it contains a trivial Form_Load routine, and the additional click and service routines

11.1.12.1 But1Initialize_Click

This is a local simplified version of the routine of same name described in 11.1.4.3 (connects bar code reader and loads SCT but without crash or report handling)

11.1.12.2 But2Insert_Click

This is a simplified version of the routine described in 11.1.4.4. It reads the bar code via `code_reader`, then checks it is for the expected instrument, checks whether the mask is already recorded as stored in the SCT, and, if checks are passed, requests a confirmation that has been stored. A successful storage is logged to the ops log and recorded in the SCT. Finally the operator dialogue allows to loop back or terminate.

11.1.12.3 But3End_Click

This trivial routine exits immediately in case of a null run, otherwise calls `InsertEnd` to terminate.

11.1.12.4 InsertEnd

This is a trivial simplification of the routine described in 11.1.4.8. It just closes communication to the bar code reader and writes an annotation in the log file.

11.1.12.5 But4ViewLog_Click

Identical to the routine in 11.1.14.4

11.1.12.6 But5Last_Click

Identical to the routine in 11.1.14.5

11.1.12.7 code_reader

This is a local simplified version of the routine of same name described in 11.1.4.10 (without suspension handling)

11.1.13 InsOrders

This module defines the INSERTION ORDER SELECTION FORM. Its graphical layout is exemplified in section 6.1.8.1 of the Operator Manual. In addition it contains a `Form_Activate` routine, which fills a selection list box with a directory listing of MIJ files present in the staging area, and two trivial `OKButton` and `CancelButton` handlers.
This routine handles specially the `null` order, which is always present (does not come from MHS) and is not counted nor selected by default as a "normal" order, unless it is the only one present.

11.1.14 InstrChoice

This module defines the *START* FORM. Its graphical layout is exemplified in section 6.1.1 of the Operator Manual. In addition it contains the following routines.

11.1.14.1 Form_Load

Called at form loading, initializes a few variables and logs the start of MHS *and the starting date* to the ops-log file, *eventually creating a new log* if the day has changed.

11.1.14.2 routines removed

The routines used to toggle VIMOS or NIRMOS instrument have been removed

11.1.14.3 But1OK_Click

Associated to the OK button, logs the selection of the instrument to the ops-log, reads the relevant `configure.txt` file (see section 11.5.6) and calls the `MDIForm`. The form caption now indicates the version of MHS running.
In addition it will also read and the reference load postion, speed and acceleration (`n=0`) for the MHS robot Z-displacement unit.
Reading the `posn.txt` files which contain the reference positions for the 15 slots of the 4 quadrant ICs (n=1..4) is instead deferred to the `ICChange` (see 11.1.23) routine which is called only when the robot is actually used.

11.1.14.4 But2ViewLog_Click

Associated to the VIEW LOG button, invokes `LogView`

11.1.14.5 But3Last_Click

Associated to the LAST ACTION button, invokes `LastProg`

11.1.14.6 But3Cancel_Click

Associated to the EXIT button, terminates MHS (after confirmation via a POPUP) logging a stop action to the ops-log file.

11.1.15 LastProg

This module defines the LAST ACTION form. Its graphical layout is exemplified in section 6.1.4 of the Operator Manual. It contains just a self-explanatory `Form_Load` routine, , and a trivial `But1Close_Click` handler.

11.1.16 ListOrders

This module defines the ORDER LIST form. Its graphical layout is exemplified in section 6.1.3 of the Operator Manual. It contains just a self-explanatory `Form_Activate` routine, and a trivial `But1Close_Click` handler.

11.1.17 LogView

This module defines the LOG VIEW form. Its graphical layout is exemplified in section 6.1.3 of the Operator Manual. It contains just a self-explanatory `Form_Activate` routine, , and a trivial `Command1_Click` handler.

11.1.18 ManOrders

This module defines the MANUFACTURING ORDER SELECTION FORM. Its graphical layout is exemplified in section 6.1.6 of the Operator Manual. In addition it contains a `Form_Activate` routine, which fills a selection list box with a directory listing of MMJ files present in the staging area, and two trivial `OKButton` and `CancelButton` handlers.

11.1.19 Remove

This module defines the SINGLE MASK REMOVAL FORM. Its graphical layout is exemplified in section 6.1.10.4 of the Operator Manual. In addition it contains a trivial Form_Load routine, and the additional click and service routines

11.1.19.1 But1Initialize_Click

This is a local simplified version of the routine of same name described in 11.1.7.3 (connects bar code reader and loads SCT but without order or ICT handling)

11.1.19.2 But2Remove_Click

Manages the search for a named mask (reading the mask id from an input box filled by the user). It checks the requested mask is for the expected instrument, and whether it is already present in the SCT. If checks are passed starts a SC bar code reader search loop. When the mask is found emits an audible beep until receives a signal from the operator (via the `found` flag) to stop the search.

This routine also has an operator dialogue which allows to loop back or terminate

11.1.19.3 But3Found_Click

This is a local simplified version of the routine described in 11.1.7.5 (sets the `found` flag)

11.1.19.4 But5NotFound_Click

A trivial button handler used to stop the search loop. Just sets the `found` flag, writes appropriate messages to the ops log and updates the SCT.

11.1.19.5 But4End_Click

This trivial routine exits immediately in case of a null run, otherwise calls `RemoveEnd` to terminate.

11.1.19.6 RemoveEnd

This is a trivial simplification of the routine described in 11.1.7.9. It just closes communication to the bar code reader and writes an annotation in the log file.

11.1.19.7 But5ViewLog_Click

Identical to the routine in 11.1.14.4

11.1.19.8 But6Last_Click

Identical to the routine in 11.1.14.5

11.1.19.9 UpdateSct

Functionally equivalent to the routine described in 11.1.7.8

11.1.20 ScanIC

This module defines the IC RESCAN FORM. Its graphical layout is exemplified in section 6.1.10.2 of the Operator Manual. In addition it contains a trivial Form_Load routine, and the additional click and service routines

11.1.20.1 But1Initialize_Click

This is a local simplified version of the code described in 11.1.6.1 (connects bar code reader and robot, and does initial robot platform positioning*, and also calls* `ICChange` *(see 11.1.23) to determine which set the ICs belong to*)

11.1.20.2 But7Next_Click

This is an hyper-simplified version of the routine described in 11.1.5.4. Keeps an internal record of the current quadrant and slot, and moves the robot platform to the next position, writes a message to the operator and exits. However if the last slot has been processed, commands the robot platform to the unloading position after acknowledgment.

11.1.20.3 But8Empty_Click

A trivial routine which records in the memory copy of the ICT that the current slot is empty.

11.1.20.4 But9Read_Click

This dedicated routine first calls code_reader to identify the mask in the decoding stand, then verifies the mask belongs to the correct quadrant, and asks to reinsert, writing an appropriate message to the log file and updating the memory copy of the ICT.

11.1.20.5 But3End_Click

A trivial routine allowing to terminate the program after confirmation or to stay in and continue.

11.1.20.6 But5ViewLog_Click

Identical to the routine in 11.1.14.4

11.1.20.7 But6Last_Click

Identical to the routine in 11.1.14.5

11.1.20.8 ScanICEnd

This is a local simplified version of the routine described in 11.1.6.8 (closes connection to bar code reader and robot, and writes ICT to disk if requested)

11.1.20.9 code_reader

This is a simpified version of the routine of same name described in 11.1.6.11

11.1.21 ScanSC

This module defines the SC RESCAN FORM. Its graphical layout is exemplified in section 6.1.10.1 of the Operator Manual. . In addition it contains a trivial Form_Load routine, and the additional click and service routines

11.1.21.1 But5ViewLog_Click

Identical to the routine in 11.1.14.4

11.1.21.2 But6Last_Click

Identical to the routine in 11.1.14.5

11.1.21.3 But1Initialize_Click

This is an ultra-simplified version of the routine of same name described in 11.1.19.1 (it does loads the SCT).

11.1.21.4 But2Scan_Click

This dedicated routine prints instructions to the operator, then reads from him via an Input Box the number of masks expected in a given quadrant. It then loops reading from the SC bar code readers until that many different mask identifiers have been read (and stored in a memory updated SCT). The same identifier may be read many times, but will be always counted as belonging to the same mask.
If the identifier does not belong to the current quadrant, the routine `WrongSC` is called to dispose of it.
The routine handles internally a loop on the four quadrants, and, if all quadrants have been processed, writes the updated SCT from memory to disk if requested.

11.1.21.5 But4End_Click

A trivial routine which exits in case of a null run, or calls `ScanSCEnd` after confirmation to terminate.

11.1.21.6 ScanSCEnd

A trivial termination routine, closes the connection to the bar code reader, writes an annotation to the log file and terminates the program.

11.1.21.7 code_reader

Functionally equivalent to the routine described in 11.1.12.7, but operates on the other (SC) barcode reader.

11.1.21.8 WrongSC

A dedicated handler for the case the scan bar code readout has found a mask misplaced in the wrong quadrant. It starts a new search loop which produces an audible beep until the offending mask has been identified. The loop is terminated by a dedicated button which is made visible only in this routine.

11.1.21.9 But3Found_Click

A trivial routine which sets a flag to terminate the bar code reading loop and handles an acknowledgment

11.1.22 Search

This module defines the SINGLE MASK SEARCH FORM. Its graphical layout is exemplified in section 6.1.10.5 of the Operator Manual. In addition it contains a trivial Form_Activate routine, and the additional click and service routines

11.1.22.1 But1Initialize_Click

This is an ultra- simplified version of the routine of same name described in 11.1.19.1

11.1.22.2 But2Search_Click

This is a simplified version of the routine described in 11.1.19.2

11.1.22.3 But5Found_Click

A trivial routine which sets the `found` flag to terminate the bar code reader search and prints a message.

11.1.22.4 But3NotFound_Click

A trivial routine which sets the `found` flag to interrupt the bar code reader search and prints a message.

11.1.22.5 But4End_Click

A trivial routine which closes the connection to the bar code reader search and exits.

11.1.23 ICChange

This module defines the IC CONFIGURATION FORM. Its graphical layout is exemplified in section 6.1.8.1 of the Operator Manual. It contains a trivial `Form_Load` routine which initializes the form based on the content of the IC configuration stored on disk (file `which.txt`) and two additional service routines

11.1.23.1 But1Modify_Click

A trivial button handler which simply enables the toggle buttons for editing, sets the appropriate flag and changes the label of other button in the form.

11.1.23.2 But2Store_Click

This is the handler for the second button, which is labelled either as NO CHANGE or as STORE-EXIT, according to tact it is called directly to accept the current configursation or in modify mode. In all cases it updates the IC configuration in memory from the on screen form, and reads the appropriate robot position tables (files `posq.txt`), then writes (in modify mode) or rewrites (otherwise) the IC configuration on disk. It logs an appropriate message to the ops log file (a "read IC config" message if called directly, or a "write IC config" if called in modify mode as determined by the appropriate flag. As final action it unloads the form.

11.1.24 SlotsOutOfUse

This module defines the IC SLOT ENABLE/DISABLE FORM. Its graphical layout is exemplified in section 6.1.10.8 of the Operator Manual. It contains a trivial `Form_Load` routine which read the ICTs from disk in the list box on screen, leaving it however disabled for editing, and the additional service routines.

11.1.24.1 But2Disable_Click and But4Enable_Click

A couple of trivial handlers respectively for the DISABLE SLOTS and ENABLE SLOTS button, which enable the list box on screen for editing, and activates the relevant STORE button.

11.1.24.2 But3ExeDis_Click and But5ExeEna_Click

A couple of similar handlers for the STORE buttons respectively associated to the Disable or Enable function They scan the list box of IC slots looking for those which have been selected for editing (clicking on them), verify they are eligible for toggling (calling the `NotAllowed` error handler if not), and finally calling `Store` to dispose of the changes. Eligible for toggling means that only the transition from/to `empty` to/from `out_of_use` is allowed (it is as well allowed to leave a slot in the `empty` or `out_of_use` status).

11.1.24.3 But1Close_Click

Extremely trivial button handlers which just dismisses the form.

11.1.24.4 Store

Reads the status of the IC slots from the list box on screen into the appropriate tables in memory, and dumps them to the ICT on disk. It also handles the maintenance case that less than 15 slots are in use.

11.1.24.5 NotAllowed

Error handler called in the case one tries to toggle a non eligible slot (one containing a mask). It issues an error message and resets the screen to the previous condition ignoring all edits.

11.1.25 TablesView

This module defines the VIEW TABLE *form used to inspect the content of ICT and SCT and the IC configuration. Its graphical layout is exemplified in section 6.1.10.7 of the Operator Manual. It contains just a self-explanatory* `Form_Load` *routine, and a trivial* `But1Close_Click` *handler.*

11.2 CutManager routine description

There is only one module `Module1` in this package file `Module1.bas`
In addition there are 3 forms in this package

11.2.1 Basic module

`Module1` contains public variable declarations and the following routines

11.2.1.1 SafeKill

This is a local version of the routine described in 11.1.1.1.

11.2.1.2 LogWrite

This is a local version of the routine described in 11.1.1.2.

11.2.1.3 ErrWrite

This is a local version of the routine described in 11.1.1.3.

11.2.1.4 Crash

This is a local version of the routine described in 11.1.1.5.

11.2.2 InstrChoice

This module defines the *START* FORM. Its graphical layout and code are analogous to the equivalent MHS form described in 11.1.14. The only differences are that MHS messages are replaced by Cut Manager messages, and that the `Button4Last_Click` handler is missing here.

11.2.3 FileMove

This module defines the MAIN Cut Manager FORM. Its graphical layout is exemplified in section 6.2 of the Operator Manual. In addition it contains the following routines.

11.2.3.1 Form_Load

This module is called once when the form is loaded. The form can be referred either at program start, or when it is de-iconized after a StencilMaster run. Therefore the handling done is rather complex and requires also the routine described in 11.2.3.6.

At start, if StencilMaster is not running, the user sees a POPUP asking to exit Cut Manager (CANCEL) or to start StencilMaster (OK). BoardMaster was arranged in a way that a single instance of it can exist at a time, so there is no danger : if it is already running Cut Manager will note and log the appropriate message. This is no longer true with the current version of StencilMaster (a second copy crashes with a library error, so there is no practical danger ; however this is considered a feature when StencilMaster is used standalone) therefore the condition is trapped by us in Cut Manager using the `user32` library function `FindWindow` (which relies on the literal content of the title of the StencilMaster window).This prevents accidental startup of a second issue of *StencilMaster when invoked from Cut Manager*. *If StencilMaster is already running the user is notified via another POPUP which shall be acknowledged. Once StencilMaster has been started or verified, if a LMD file is already present in the current directory the user is submitted a POPUP with two options, either CANCEL to remove it (if an accidental leftover e.g. after a crash) and continue with the normal first time initialization, or RETRY to keep it (this assumes one is re-entering Cut Manager after an accidental exit but StencilMaster is already active in background) as if one was already inside Cut Manager.*

11.2.3.2 But1Incoming_Click

Handler for the NEW MASK button. Essentially manages the activation and deactivation of the other buttons and the message in the "NEXT ACTION" hint box according to the fact there is a file selected in the incoming folder list box, or there are no files at all. It also needs to refresh the content of the file list on the screen in case new LMD files were deposited (by the `convert` function running on MHCU in the `Incoming` area of MMCU. Note that the button invoking this function is now not disabled, exactly to allow the operator to repeatedly probe the presence of new LMD files.

11.2.3.3 But2OKIncoming_Click

Handler for the *IMPORT THE SELECTED FILE* button *relevant to new masks* Copies the selected LMD file from the incoming folder both in the current and archive folders, and removes it. The archived file has its proper name (`*qnnnnn*.lmd`) stripped of the leading sequence number added so far. Logs the appropriate ops-log message, and arranges activation and deactivation of the other buttons.

11.2.3.4 But3Archive_Click

Handler for the MASK FROM ARCHIVE button. Essentially manages the activation and deactivation of the other buttons and the message in the "NEXT ACTION" hint box according to the fact there is a file selected in the archive folder list box, or there are no files at all. At each click it also refreshes the content of the file list in the screen, to prevent attempts to access LMD files deleted in the meanwhile as consequence of the execution of MDJ discard orders.

11.2.3.5 But4OKArchive_click

Handler for the *IMPORT THE SELECTED FILE* button *relevant to archive masks*. Copies the selected LMD file from the archive folder (leaving the original there). Logs the appropriate ops-log message, and arranges activation and deactivation of the other buttons.

11.2.3.6 Form_Resize

Called when the entire form is deiconized on return from StencilMaster, updates the activation and deactivation of the other buttons and the message in the "Next Action" hint box according to the presence of a LMD file in the current directory (presence should imply one has reopened the form after a StencilMaster run, absence just after an accidental closure)

11.2.3.7 But5Manufacture_Click

Handler for the MANUFACTURE button. Logs a start message to ops-log and minimizes the Cut Manager window, thus effectively passing control to the underlying StencilMaster session.

11.2.3.8 But7Done_Click

Handler for the MASK DONE button. Logs an end manufacturing message, deletes the LMD file in the current directory and arranges activation and deactivation of the other buttons

11.2.3.9 But8Remanufacture_Click

Handler for the REMANUFACTURE button. Logs an error message and arranges activation and deactivation of the other buttons to allow a repetition of the manufacturing.

11.2.3.10 But9LogView_Click

Associated to the VIEW LOG button, shows the relevant form via a call to `LogView`

11.2.3.11 But6End_Click

Handler for the END button. A POPUP asks confirmation for exit, writes appropriate message to the ops-log and gives instructions for StencilMaster termination.

11.2.4 LogView

This module (similar to the one described in 11.1.17) defines the LOG VIEW FORM used by Cut Manager. Its graphical layout is similar to the equivalent MHS form described in section 6.1.3 of the Operator Manual. It contains just a self-explanatory `Form_Activate` routine, and a trivial `But1Close_Click` handler.

11.3 Mirroring routine description

There a single source file `MirrorForm.frm` in this package. Although nominally the program includes an (invisible) form, this simple program is intended to be run from the DOS command line (or start menu shortcut) manually, or typically via the `AT` command.

11.3.1 Form_Load

The main form will do the following automatically at its initialization :

- rename the old log file `mirror.log` into `mirror.bak`
- open the new log file `mirror.log` and log a time stamp
- open the appropriate configuration file (*`hostname`*`.conf`, with *`hostname`* being selected automatically as either `mmcu` or `mhcu`) and read the names of the master and slave drives (or directories) to be mirrored
- arrange for the remote share connection
- it will then invoke the mirroring routine between the top master and slave directories
- at the end will disconnect the remote share

The arrangement of the remote share is slightly tricky. In the case the program is run manually by the operator, the remote disk is usually already mounted at login, thus the program will notice and use it. In the case the program is run by another user (or it is run via `AT` while operator is logged), either the disk is not mounted, or is not accessible with such shared drive name. In all such cases (as well as when the disk is not mounted by the operator) the program will mount it on drive `Z` and then disconnect it when finished using the `net use` command.

11.3.2 Mirror

The mirroring routine is recursive, and, called as `Call Mirror(`*`master_path, slave_path`*`)`, takes as argument a master and a slave directory path. It will do the following operations :

- build a directory listing of the master (source) directory
  - for each element which is a directory will call recursively itself to mirror it
  - for each element which is a normal file, will test if the file exists already on the slave (destination) directory
    - if it does not exist will copy it to destination
    - if it does exist will check the date, and copy it only if the source file is newer
- build a directory listing of the slave (destination) directory
- for each element check whether it exists on the source directory. If not, it's an old file or directory and will be deleted. Note that the program will only delete files, or *terminal* directories (i.e. containing files but not subdirectories) *with* their content. It will not delete directories with subdirectories (this is a feature, since this case will not happen in our situation).

Except for the case of directories (where an error is silently ignored), any other unforeseen error will terminate the program logging a generic error message. A time stamp will always be logged at termination.

11.3.3 Auxiliary routines

There are also two trivial routines, the logical functions `existFile` and `existFileOrDir`, used to test for the existence of a plain file, or of any file (i.e. a plain file or a directory) with a given name.

11.4 Directory arrangement

Each of the three computers has a 6 GB disk partitioned in 3 2-GB partitions, identified as drives `C`:, `D`: and `E`:. These drives are used as follows :

- `C:` (a FAT filesystem) contains the operating system and the installed software on all machines
- `D:` (an NTFS filesystem) contains the MHS staging and work area
- `E:` (an NTFS filesystem) contains the Cut Manager work area

The master version of the `D:` disk is the one on MHCU. The MMCU accesses `\\mhcu\d:` as a share and periodically mirrors its content to the local `D`: disk as described in section 6.3 of the Operator Manual. The `D:` disk on `spare` is potentially empty.

The master version of the `E`: disk is the one on MMCU. The MHCU accesses `\\mmcu\e:` as a share and periodically mirrors its content to the local `E:` disk as described in in section 6.3 of the Operator Manual. The `E:` disk on `spare` is potentially empty.

The above will allow data recovery in case of interchange of one of the three computers in case of failure as described in section 12

The following sections describe the usage and role of the various standard directories.

11.4.1 Staging area

The directory `D:\staging` is the staging area used by MCS to transfer information to/from MHCU. It is divided into four directories, each one of which is the virtual ftp root directory of four dedicated ftp accounts. Each account has full access to its own directory only (and to no other file on any computer). The operator account has full access to all directories :

- `D:\staging\vimos` is reserved to the `vimos` account for use by the VIMOS MCS
- ~~`D:\staging\nirmos` is reserved to the `nirmos` account for use by the NIRMOS MCS~~
- `D:\staging\contingency` is reserved to the `contingency` account for use by foreign instruments (see section 7.10 of the Operator Manual)
- `D:\staging\fors2` is a similar contingency directory reserved for FORS2

The first two directories have a common layout. The layout and usage of the foreign contingency directories is outside the scope of the present document. The arrangement of the `Vimos` ~~and `Nirmos`~~ directories is :

- subdirectory `orders`, with three subdirectories
  - subdirectory `mmj` contains one subdirectory for each MMJ delivered by MCS
  - subdirectory `mij` contains one subdirectory for each MIJ delivered by MCS
  - subdirectory `mdj` contains one subdirectory for each MDJ delivered by MCS
- subdirectory `reports`, with three subdirectories
  - subdirectory `mmt` contains one subdirectory for each MMT generated by MHS
  - subdirectory mit contains one subdirectory for each MIT generated by MHS
  - subdirectory `mdt` contains one subdirectory for each MDT generated by MHS
- subdirectory `tables`, with three subdirectories
  - subdirectory `sct`, contains 4 files `sct1.txt` to `sct4.txt`
  - subdirectory `ict`, contains 4 files `ict1.txt` to `ict4.txt`
  - subdirectory `mob`, for convenience of the MCS

The individual M*x*J (MMJ, MIJ, MDJ) directories in turn contain at least one job file of the relevant type (`mmj`, `mij`, `mdj`) and, in case of MMJ, also all relevant MSF files.
The individual M*x*T directories in turn contain a single termination report file of the relevant type (`mmt`, `mit`,`mdt`).
The naming of the files is described in 11.5.1
The `mij` directory always contains also a `null` directory, with the `null.mij` insertion job file used for maintenance to empty the IC. This directory and ists content are never deleted.
The `icti.txt` and `scti.txt` files are the latest copy of the ICT and SCT tables, and are updated any time that MHS deposits a new termination report.

11.4.2 MHS work area

The directory `D:\mhs` is the work area used by MHS. It is divided in four directories : `vimos`, ~~*nirmos*~~ plus `fors2` and `contingency`. The first two directories have a common layout. The layout and usage of the foreign contingency directories is outside the scope of the present document. The arrangement of the `Vimos` ~~*and Nirmos*~~ directories is :

- one file `configure.txt` (used by the MHS program to load configuration information and described in 11.5.6)
- directory `archive`, containing support files for special masks
- directory `calib`, containing file `offsets.txt` which are the x and y offsets in mm of the optical axis with reference to the mask centres for the four quadrants (these offsets are used to convert the coordinates relative to the optical axis in the MSFs to the ones used in Gerber files).
- directory `contours`, containing 4 `contour_q.gbr` Gerber template files used to include the Gerber commands for the contour of the mask for quadrant *q* (coordinates in these files are relevant to the mask geometric centre)
- directory `curr` is used to contain the single Gerber file being processed by CircuitCam (taken from `gbr`) and the LMD file produced (then moved to `lmd`)
- directory `gbr` where MHS convert places the Gerber files it generates
- directory `holes`, containing 4 `holes_q.gbr` Gerber template files used used to include the Gerber commands for the fixed holes of the mask for quadrant *q* (coordinates in these files are relevant to the mask geometric centre)
- directory `refpins`, containing 4 `refpins_q.gbr` Gerber template files used used to include the Gerber commands for the reference pins of the mask for quadrant *q* (coordinates in these files are relevant to the mask geometric centre; the two reference pins are along the contour but are cut as separate apertures for easier maintenance of their position)
- directory `lmd` where MHS convert places provisionally the LMD files it generates before moving them to MMCU
- diredtory `msf` where MHS convert copies MSF files from the staging area
- directory `orders` where MHS copies job order directories from the staging area (this has the same arrangement of the directory of same name in the staging area (see 11.4.1) except that MSFs are moved to the `msf` directory)
- directory `reports` where MHS generates termination reports (and temporary reports and files) before moving them to the staging area (this has the same arrangement as the directory of same name in the staging area; see 11.4.1)
- directory `robot` contains the *`pos0`*`.txt` *and `which.txt`* files *(i.e. the general robot setup parameters, and the IC configuration file), and two subdirectories `A` and `B`, each one with an independent set of `posq.txt` files* with the robot reference positions *of the slots in the ICs of the two sets (A=original, B=spare)*
- directory `tables`, which contains the master versions of ICT and SCT (this has the same arrangement as the directory of same name in the staging area (see 11.4.1)

In addition there are MHS service files which are stored in the root directory `D:\`, namely :

The daily log files `mhcu_yyyy-mm-dd.ops-log` (described in 11.5.7)
The `last_prog.txt` file used to store the status of the last action undertaken by MHS across runs.

11.4.3 Cut Manager work area

The root directory `E:\` is the work area used by Cut Manager on MMCU. It is divided in four directories : `vimos`, ~~*nirmos*,~~ `fors2` and `contingency`. The first two directories have a common layout. The layout and usage of the remaining two contingency directories is outside the scope of the present document. The arrangement of the `Vimos` ~~*and* *Nirmos*~~ directories is :

- directory `incoming`, containing the LMD files moved there by MHS `convert`. The name of these files is *m_qnnnnn*`.lmd` where the suffix *m* is a sequence number, and *qnnnn*n the mask id.
- directory `curren`t, containing the single LMD file intended to be currently processed by StencilMaster
- directory `archive`, containing the LMD files of all already manufactured masks until discarded. The name of these files are just in the form *qnnnnn*`.lmd`
- directory `permanent`, containing LMD files for maintenance masks described in section 7.7 of the Operator Manual
- directory `special masks`, containing LMD and JOB files for special purposes

In addition there are Cut Manager service files which are stored in the root directory `E:\` , namely the daily log files `mmcu_`*yyyy-mm-dd*`.ops-log` (described in in 11.5.7)

11.4.4 Mirroring work area

The `mirroring` program has a small work area in `C:\Mirror`. This area contains the executable, the two log files `mirror.log` (current) and `mirror.bak` (last), the startup batch `mirrorsetup.bat` and the configuration files `mmcu.conf` and `mhcu.conf`. Both files must be present on all machines. The program will select the appropriate one.

The configuration files will contain four lines as follows.

Line	Content	Example
1	source directory	`d:\`*[dir\]*
2	destination hostname	`mmcu`
3	destination share name	`dclone`
4	destination directory	`\`*[dir\]*

The name of the destination directory will be built by the program as e.g. `g:\dir` or `z:\dir` where drive `g` or `z` will respectively be already mounted as, or programmatically mounted as share `\\mmcu\dclone` . Note that lower case and final backslashes are **mandatory**. Typically one is mirroring an entire drive, therefore the part in italics is absent (the square brackets are not part of the text and shall **not** be typed but just there to remind a directory path is optional).

11.4.5 data area for roughness meter

The data area for the roughness meter software is currently set at shortcut property level on `C:\Roughness`

11.4.6 Summary of directories where s/w is installed

We summarize here a list of all directories relevant to non-system software

- `C:\Barcode\Winhost` contains the bar code configuration utility (installed on `spare` only)
- `C:\niserial` contains the drivers for the National Instrument additional serial ports (installed on `spare` and `mhcu`)
- `C:\ProgramFiles\TaylorHobson` contains roughness meter software (Mountains) (installed on `spare` only)
- `C:\ProgramFiles\Ftsp` contains roughness meter software (ftsp front end) (installed on `spare` only)
- `C:\ProgramFiles\Orl` contains the VNC software used to make screen dumps
- `C:\Roughness` is the roughness meter data area (on `spare` only)
- `C:\LPKF31` LPKF CircuitCam software and spare data area

- `C:\LPKF30\Bmaster` LPKF BoardMaster software (original version, on MMCU only) and spare data area
- `C:\LPKF30\Smaster` LPKF StencilMaster software (current version) and spare data area
- `C:\MhsSW` IFCTR MHS executable
- `C:\MmuSW` IFCTR Cut Manager executable
- `C:\Mirror` IFCTR Mirroring executable and service files
- `C:\Robot_test` executable for robot testing (on MHCU only)
- `C:\Development` contains the Visual Basic (VB) development environment, inclusive of the source files for MHS, Cut Manager, Mirroring and robot test s/w (inclusive of port simulator and installed on `spare` only). In particular the directories
- `C:\Development\Microsoft Visual Studio\VBkits Mhs, Cutmanager` and `Mirroring` contains the CAB kits for MHS, CutManager and Mirroring (on `spare` only) built as described in 10.1
- `C:\UserManual` HTML (and eventual source) version of the present manual (on `spare` only)

11.5 File layout and purpose

This section describe the purpose, content and layout and naming convention of the various files used for mask handling, or makes reference to appropriate documents.

11.5.1 Job and Termination report files

- Job order files (MMJ, MIJ, MDJ) are generated by MCS in response to an order file received from OHS, and are ASCII files with a list of mask identifiers. These files have extensions `mmj`, `mij` and `mdj`.
- Termination report files (MMT,MIT. MDT) are generated by MHS and made available to MCS, with a one-to-one correspondence between a job and a termination report (the only exception is the `null.mij` file, which corresponds to a request generated autonomously within MHS when the ICs must be completely emptied). They are ASCII files with a list of mask identifiers, and an associated error code. These files have extensions `mmt`, `mit` and `mdt`.
- The names of the files *yyyy-mm-dd*T*hh_mm_ss*.000-*nnn*.mx*j* correspond to the ESO ICD naming convention (ref. [1]), i.e. ISO date-and-time and sequence number, except that the `hh:mm:ss` string is expressed as `hh_mm_ss` are required by the Windows NT operating system.
- Full detail on the file layout is contained in. MCS User Manual [ref. 3]),

11.5.2 Machine Slit Files

- Machine slit files are ASCII files containing the description of a mask, i.e. an identification header, the position and size of a number of standard (rectangular slits), and eventually the position and parameters of a number of curved slits.
- The names of the files are of the form *qnnnnn*`.msf` where `qnnnnn` is a mask identifier. The digit *q* assumes the values 1,2,3,4 for the four VIMOS quadrants ~~and 5,6,7, 8 for the four NIRMOS quadrants~~. The 5 digits *nnnnn* are a sequential number with leading zeros (from `00000` to `99999`). The mask identifier appears on the masks in form of a barcode.
- Sequential numbers are assigned by MCS (with the first 100 mask numbers being reserved e.g. for permanent masks) and recycle at 99999 (since no more than 100 mask sets can be stored in SC prior to discarding, no ambiguity should arise).
- Full detail on the file layout is contained in. MCS User Manual [ref. 3]).
- In order to support the production of maintenance masks MHS supports an internal extension to the MSF layout defined in such document, which is recalled here :
  - The "number of rectangular slits" in the header includes also the case of round apertures and step-and-repeat patterns.The latter are a Gerber facility to define in a compact way a regular raster pattern of apertures.
  - For each round slit add 1 to the number of rectangular slits
  - For a step-and-repeat pattern add 1 to the number of rectangular slits (note that there can be only one step-and-repeat pattern in a file because of limitations in CircuitCAM, if there are more raster patterns in a MSF each slit must be defined individually).
  - Rectangular slits are defined by a normal entry with a quadruple X_0 Y_0 [Delta]X [Delta]Y
  - Round apertures are conventionally defined by an entry with a quadruple X_0 Y_0 radius 0.0 (in mm)

- A step and repeat pattern is defined by a couple of entries : the second is a normal round or rectangular slit centred in the starting point (lower leftt) of the pattern, while the first quadruple has the peculiar form `dx dy -nx -ny` with `dx` and `dy` being the step sizes of the pattern in mm, while `nx` and `ny` are the number of steps on X and Y (they are flagged as negative values for convenience of the program)

11.5.3 standard Gerber files

Standard Gerber files *qnnnnn*`.gbr` are produced by MHS convert from each MSF files, and inherit the same mask identifier (except that a numeric prefix could be temporarily added during manufacturing). Gerber format files are ASCII files used for description of mechanical parts. The full specification of the Gerber language is outside of the scope of the present document, and can be found in Gerber Documentation [ref.4]
We present here an annotated sample excerpt of a typical Gerber file for a VIMOS mask:

`%IN_VIMOS_MASK_305x305_1_100204.msf*%`	heading (all this is fixed format but name)
`%ASAXBY*%`	axis order
`%FSLAX33Y33*%`	format for X Y coordinates 3.3 digits (no decimal dot)
`%MOMM*%`	positions are in mm
`%SFA1.0B1.0*%`	scale factor
`%IOA0B0*%`	default direction
`%IR0*%`	no rotation
`G04__________DEFINE_APERTURES*`	comment : start aperture definitions
`%ADD10C,0.0*%`	define circular apertures for holes : 10 null hole
`%ADD13R,0.8X3.0*%`	define rectangular apertures for bar code: 13 is thin line
`%ADD14R,1.6X3.0*%`	14 is thick line
`%ADD15R,6.0X3.0*%`	15 is leader/trailer (quiet zone)
`%ADD20R,2.38X2.38*%`	define aperture 20 as square hole (reference object)
`%ADD21R,2.38X2.38*%`	another reference object
`%ADD22R,2.737X0.406*%`	define aperture 22 as a slit
`%ADD23R,2.737X0.406*%`	another slit (all slits defined even if same size)
`...`	and so on, and so forth
`G04________SLIT_CUTTING*`	comment : start standard slit “flashing”
`%LNSLITS*%`	associate to layer named `SLITS`
`G54D20*`	object to be flashed is aperture number 20
`X144160Y135530D03*`	flash it at this X,Y position
`G54D21*`	another aperture
`X153326Y162189D03*`	
`...`	and so on, and so forth
`G54D29*`	
`X28237Y97705D03*`	
`G04________CURVED_SLIT_CUTTING*`	comment : start curved slits if any
`G54D10*`	define pen as zero-size hole (aperture number 10)
`X163892Y163602D02*`	move to starting point
`X164110Y163249D01*`	draw to next point
`X164322Y162923D01*`	
`...`	and so on, and so forth
`X163892Y164182D01*`	
`X163892Y163602D01*`	
`G04________REFERENCE_PINS_APERTURES__Q1`	comment, start reference pin apertures along contour
`%AMREFV*1,1,2,0,0*4,1,4,-0.707,0.....`	define a macro named REFV for one peculiar aperture (because of orientation depends on couple of quadrants : quadrants 1=2 but different from quadrants 3=4)
`%ADD11REFV*%`	define aperture 11 as macro REFV
`G54D11*`	object to be flashed is aperture 11
`X-138500Y-150710D03*`	flash it at this position (depends on quadrant)
`%ADD12R,10.0X4.74*%*`	define aperture 12 as a rectangular hole
`G54D12*`	object to be flashed is aperture 12
`X102500Y-152500D03`	flash it at this position (depends on quadrant)

```
G04______________BAR_CODE*             comment : start barcode section
G54D15*                                leading quiet zone
X81000Y301500D03*                      flash it at this position (depends on quadrant)
G91*                                   here onwards all coordinates are relative movements
X3400D02*                              always move
X800D02*                               a move followed by no aperture is a “dark slit”
G54D13*                                select thin slit
D03*                                   cut thin “white” slit at current position
X800D02*                               move
...                                    lot of code generated by the program according to mask id
G54D15*                                trailing quiet zone
D03*                                   flash it to terminate bar code
G90*                                   here onwards back to absolute coordinates
G04________MANINA HOLES__Q1            comment : holes for mask movement (manina = little hand)
%LNBARCODE*%                           optional low speed  layer historically named BARCODE
%ADD16C,6.7*%                          define aperture 16 as round hole 6.7 mm
G54D16*                                object to be flashed is aperture 16
X-146250Y92490D03*                     flash it at this position (depends on quadrant)
Y112490D03*                            flash two more displaced only in y direction
Y132490D03*
G04________QUADRANT_ID__Q1*            comment : start quadrant ids
...                                    code to generate a pattern of 1-4 holes as quadrant id
G04________FIXING_HOLES__Q1            comment : holes for mask fixing once in position
%ADD17R,3.0X4.5*%                      define 17 as a rectangular aperture
G54D17*                                object to be flashed is aperture 17
X73500Y147480D03*                      flash it at this position (depends on quadrant)
X-37500D03*                            flash one more displaced only in x direction
G04_CONTOUR_version description  *     comment : start contour section
%LNCONTOUR*%                           associate to layer named  CONTOUR
G90*
G54D10*                                all this done with zero-radius pen
X-2000Y150000D02*                      lot of code generated by the program copying this from a
...                                    template file (different for each quadrant and instrument)
M02*                                   end of file
```

The Gerber language allows to group the cutting or drawing of particular objects into *layers* (with a symbolic name defined by a `%LN` statement). This is essential for use with LPKF CircuitCam and StencilMaster. One of the functions of CircuitCam is to associate each input layer (in the Gerber file) with an *output layer* (i.e. a virtual *tool*, corresponding to a givern power, frequency of the laser beam and a given moving speed) and a production *phase*.

In our case (as described in sections 8.1 and 8.2) the layer `SLITS` (used not only for slits but also for the barcode and reference pin apertures) is associated with a precision tool, the layer `BARCODE` (the name is historical, it might be used nowadays despite its name for the positioning and fixing holes) with a faster tool if desired. Both layers are in the *“Cutting Side” phase* (i.e. are processed together by StencilMaster). The layer `CONTOUR` is instead associated to a faster tool in the *“Outline” phase* (i.e. must be processed separately by StencilMaster, since once the contour is cut, the piece will fall off).

11.5.4 special Gerber files

Special Gerber files (with extension `.gbr`) can be used for various maintenance purposes. These includes full Gerber files with the layout of maintenance masks (see section 7.8 of the Operator Manual) and quality samples (see section 7.9 of the Operator Manual), and also includes partial Gerber files described here. These files are not functional, but are used by MHS `convert` as templates included in each mask Gerber file as appropriate. They include (see 11.4.2) :

- mask contours (files `contour_`*`q`*`.gbr`)
- mask fixed holes (files `holes_`*`q`*`.gbr`)
- mask reference pin apertures (files `refpins_`*`q`*`.gbr`)

11.5.5 LMD files

The LMD format is the LPKF proprietary format used to store the information used by StencilMaster to manufacture the masks. Documentation on their layout is reserved by LPKF. Standard LMD files (with names *`qnnnnn`*`.lmd`) are produced by LPKF CircuitCam starting from a standard Gerber file.

An LMD file describes a single piece to be cut. LPKF job files (extension `.job`) can combine the manufacturing of more than one piece replicated at different positions on the same sheet of material, but are currently not used.

11.5.6 working files

Storage Cabinet Tables (SCT) and Instrument Cabinet Tables (ICT) are working files whose master copy resides in the MHS work area of MHCU. A copy of these files is however made available to MCS in the staging area, therefore the layout of such files is also described in MCS User Manual [ref. 3]).

- There are four `scti.txt` and four `icti.txt` files (with *`i`*`=1..4` being the quadrant number) per instrument. They are all plain ASCII files, and list the line number and one mask identifier per line.
- In the case of the ICT, each file is exactly 15 line long, and may also contain the `empty` *or `out_of_use`* mask identifier. This is because slots in the ICs are numbered, and the system keeps track of which mask is in which slot. If at any time ICs will be operated with less than 15 slots, the unused slots (at the end of the list) shall be flagged with the `unused` mask identifier. The program is capable of reverting from/to the `empty` to/from the `unused` identifier if the number of slots in the `configure.txt` file is edited and MHS restarted.
- In the case of the SCT instead slots in the SCs are not numbered, and the system does not keep track of empty slots, nor of the location of a given mask, but just of its presence. For convenience however the files have the same format, i.e. 100 lines containing a line number and a mask identifier or the `empty` keyword (without this having a relationship with the order of actual SC slots)

There are also working files used internally by the software, namely

- configuration file `configure.txt` (in `D:\Mhs\`*`instrument`*) with three lines : the first one is the name of the MMCU drive where LMD files have to be transferred by the convert MHS function ; the second line is the full path of the CircuitCam software, the third line is the number of slots in the ICs (currently 15 i.e. all). These values (in particulary the latter) are read at MHS initialization and all MHS modules are parametric with respect to the number of IC slots.
- `last_prog` file (in `D:\`) used internally to keep track of the last instrument, MHS function and order used during the last MHS run, and of the time the last function was started and stopped (or resumed or suspended).
- the temporary report file `temp_rep.txt` used by the programs to build a running version of the MxT reports (and formatted as an MxT, eventually without header at the convenience of the program)
- the `producecrash.txt` and `storecrash.txt` used for checkpointing the last executed phase of respectively convert and store (so that the program can restart correctly after a crash or suspension)
- the alignment file `offsets.txt` containing the offsets between optical axis and mask centre
- *the IC configuration file `which.txt` (in `D:\Mhs\Vimos\Robot`) containing on one line four flags indicating to which cabinet set the four ICs currently in use belong. The flag is 0 or 1 for the original (A) or spare (B) set.*
- robot position reference files `pos`*`i`*`.txt` files (with *`i`*`=1..4` being the quadrant number) containing the Z positions in mm of the 15 slots of each IC when mounted into the IC holder and onto the MHS robot platform (if less than 15 slots in the IC will be used, the relevant values are "don't care"). *There are two sets of such files, located respectively in directories `D:\Mhs\Vimos\Robot\A` and `D:\Mhs\Vimos\Robot\B` for the two IC sets (A=original, B=spare).* Plus a file `pos0.txt` *(located in `D:\Mhs\Vimos\Robot`)* containing the coordinate in mm of the unloading position, and the default speed and acceleration (between 0 and 100). The latter file shall never be changed, while the quadrant `pos`*`i`*`.txt` files must contain the "calibrated" slot positions of a specific cabinet. The cabinet calibration procedure has to be repeated if the IC is physically replaced *or dismounted and remounted (and in any case the ICs in the spare set have to be calibrated independently of those in the original set)*, and is the following :
  - make a backup copy of the `pos`*`i`*`.txt` file
  - open the `pos`*`i`*`.txt` file with an editor (e.g. notepad editor)
  - start the `C:\Robot_test\robot_test` program
  - press the Zero Axis button to home the robot
  - *press the Speed Setup button and then return to accept the default value proposed in the popup*

- *press the ACCEL SETUP button and then return to accept the default value proposed in the popup (this and the former step ensure the robot is initialized correctly as in 11.1.1.14).*
- for each slot repeatedly use the MOVE button to input a position to move the slot in front of the mask stand (positions are in mm consistently with the content of the `posi.txt` file and are displayed on the screen)
- test the mask insertion in the IC : the mask should slide in the right slot remaining flat without bending upwards or downwards. If this does not occur, extract the mask, use the MOVE button to adjust the position and retry.
- remember that, if you intend to move to a position which is lower than the current you must first move to a position 10 mm below the current and then move again up to the desired one (to emulate the behaviour of the robot which always reaches slot positions from below)
- When satisfied, record the current position in the `posi.txt` file, and proceed to the next slot

11.5.7 log files

There are two kind of log files (plus the short-lived mirroring log files described in 11.4.4).

- Ops-log files are produced by MHS and Cut Manager, and keep track of the operations done by operator as described further below.
- Proprietary software log files are produced by LPKF software outside of our control.

For what concerns LPKF software CircuitCam has no log file while BoardMaster and StencilMaster keep their own ever-growing log files (which however records only the stop and start of manufacturing phases, but does not identify the LMD file being manufactured) respectively in `C:\Lpfk30\Bmaster\Bmaster.log` and in `C:\Lpfk30\Smaster\Smaster.LOG`. This file should periodically be purged.

Ops-log files are mantained by MHS and Cut Manager in the directories described in 11.4.2 and 11.4.3 respectively. The name and layout of these files is as far as possible compliant with the rules in chapter 5 of the ESO DICD [ref.2] .

Namely the names of the file is of the form `host_yyyy-mm-dd.ops-log` where `host` is one of `mhcu` or `mmcu`, and the date is the date of file creation (the host name and date are separated by an underscore and not by a dot for reasons mandated by Windows NT). A new log file is opened automatically the first time MHS or Cut Manager are called in a given day after midnight. This is at variance with in the ESO DICD , where log files are changed at noon, but this is justified since MMU will be mainly operated during normal working hours, unlike a telescope operated at night time. The purging of old daily log files is outside of the scope of the present document.

The daily logs are ASCII files (with a maximum line length 97 characters, this is longer than the 72 characters recommended by the ESO DICB, but still consistent with it, since inclusion of log information in FITS headers is not applicable to the MMU) which can also be accessed by the VIEW LOG button (see section 6.1.2 of the Operator Manual). The typical record is divided into four columns :

- the time of the day
- the action verb in the form given by Table 15 of the DICB, or `/UNFORESEEN` or `/RECOVERY` records, mimicking typical VLT ops-log.
- a comment section giving more information on the particular action (e.g. mask identifiers or quadrant numbers are listed here)
- a subsystem identifier in the form `[hostI]`, where `host` is either `mhcu` or `mmcu`, and the instrument code `I` (when applicable) is `V` for VIMOS ~~and N for NIRMOS~~.

With the exception of the first `DATE` record, or a few comments inserted for clarity and of the `/UNFORESEEN` or `/RECOVERY` records, all other records are action records using the following verbs.

Verb	subsystem & argument	usage
`-START`	`MHS`	startup of MHS program
`-START`	`CUTMGR`	startup of Cut Manager program
`-START`	`instrument`	instrument selection performed (both MHS and Cut Manager)
`-START`	`BOARDMASTER`	Cut Manager is starting StencilMaster for the first time

Verb	subsystem & argument	usage
-START	MANUF	Cut Manager passes control to StencilMaster
-START	MHS (sub)program	start of MHS function
-STOP	MHS	termination of MHS program
-STOP	MHS (sub)program	termination of MHS function
-STOP	CUTMGR	termination of Cut Manager program
-STOP	MANUF	Cut Manager notified of terminated manufacture
-ABORT	MHS program	forced premature program termination
-PAUSE	MHS program	suspend requested by operator
-RESUME	MHS program	resuming after suspend or crash
-RESUME	BOARDMASTER	Cut Manager attempts to start already running StencilMaster
-OPEN	BARCODE number	communication with bar code port established
-OPEN	ROBOT PORT	communication with robot port established
-CLOSE	BARCODE number	communication with bar code port closed
-CLOSE	ROBOT PORT	communication with robot port closed
-READ	BARCODE	successful read of barcode mask identifier
-READ	*IC CONFIG*	*retrieval of IC configuration (file which.txt)*
-READ	ICT	retrieval of content of ICT
-READ	SCT	retrieval of content of SCT
-READ	MxJ	job order file copied from staging area to work area
-READ	MSF	MSF copied from staging area to work area
-WRITE	GERBER	created Gerber file
-WRITE	LMD	created or copied LMD file
-WRITE	*IC CONFIG*	*IC configuration on disk (file which.txt) updated*
-WRITE	ICT	updated ICT content or made full copy to staging area
-WRITE	SCT	updated SCT content or made full copy to staging area
-WRITE	MxT	copied Termination report to staging area
-MOVE	MASK	mask extracted/inserted from/to IC or SC
-MOVE	LMD FILE	(Cut Manager) retrieved LMD file to current directory

Note that only actions managed through MHS or Cut Manager are logged. Manual actions (using directly CircuitCam or StencilMaster) are **not** logged. Robot movements are not logged, as well as the switch on of the bar code readers in reading mode. Only the result of a search or insertion|removal are logged (an exception is that comments are inserted for clarity concerning the search of masks to be discarded).

11.6 FTP protocol with MCS

All transactions for data exchange with MCS are initiated on the IWS side, and therefore for all details we refer to MCS User Manual [ref. 3].

- The protocol used is ftp (MCS puts job orders and related files, and gets termination reports; the ascii transfer takes care of the different record termination conventions used by Unix and Windows NT). There are dedicated ftp accounts for use exclusively by ftp, one reserved to the VIMOS IWS, ~~one to the NIRMOS IWS,~~ one for the conveniency of FORS2 and a fourth one for any other purpose (as described in section 7.3). These accounts access only the MHCU staging area described in section 11.4.1.
- For what concerns job orders, the procedure used by the IWS is first to create an order subdirectory, then to populate it with any related files (namely, MSFs for a MMJ), and finally to put the job order file itself. Until the job order file (with the right name corresponding to its subdirectory) appears, MHS will ignore the job (considering it "currently being transmitted". MCS has the authority to remove any order file insofar MHS has not yet started processing it. MCS will use temporary file names during transfer and MHS will duly ignore them (prefixed with temp_) until renamed.
- The MCS will keep account of the orders sent. As soon as an order disappears from the staging area (because MHS has moved it to its work area) MCS considers it "under execution" until a termination report is written.

- MHS will create in the staging area termination report subdirectories, update the copy of SCT and ICT and finally move the termination report file in the appropriate directory. At this time a poll from MCS will find the report, and take care to retrieve it with all associated files, and to dispose of them. MHS will never remove a termination report: this task is reserved exclusively to MCS.

12. Extraordinary maintenance

This section lists the interventions to be undertaken for recovery of severe hardware failures, which may imply replacement of spare parts (LRUs).

12.1 Spare parts available for LPKF cutting machine

Extraordinary maintenance of the LPKF machine inclusive of replacements (exclusive of parts in the standard spare kit, and of additional parts) is covered by the LPKF maintenance contract. Spare part lists are given in the LPKF StencilLaser documentation (section 11.5 of [ref. 8]) and in associated hardware documents.

The standard spare kit with additions includes (as delivered before installation and excluding additional items left by LPKF after installation or eventually acquired by ESO)

- 4 laser lamps
- 2 spare lamp contacts
- laser focusing unit and protective "lens"
- 1 pair of goggles (must be weared when laser cover is removed)
- paper for cleaning optical parts
- 1 steel nozzle (plus an additional 1)
- drill points for nozzle cleaning
- 3 felt rings (plus 1 additional felt and 1 brass ring) for exhauster
- 1 O-ring
- assorted tools
- particle filter cartridge for primary cooling circuit filter (filter unit inside control rack)
- additional spare filter unit (blue case) for chiller (contains one spare cartridge)
- 1 additional filter cartridge for chiller filter
- additional complete set of "rapid" taps for chiller
- additional 2 380 V industrial plugs

For routine intervention and replacement of pieces in the standard spare kit see Operator Manual 9.2

In addition the consumables listed in see Operator Manual 9.2 are supplied or shall be procured in situ

12.2 Spare parts for SC

A 10 m spare connection cable for the SC barcode (serial communication and power supply) has been delivered during the visit of G.Conti in July 2001. This cable (although long) can be used also as a spare for the IC barcode.

12.3 Spare parts for IC robot

No spare parts were originally supplied, a recommended spare part list is given in the Antil documentation [ref. 10]. Quotations for spare parts are available but budget does not allow to supply any at present. The policy will be reviewed jointly by ESO and the VIRMOS Consortium.

The following spare parts were delivered in occasion of the repair intervention during the visit of G.Conti in July 2001.

- proximity switch M12 NPN with relevant connector
- photoelectric emitter XUM H07301
- photoelectric breaker (receiver) XUM H073534
- 10 fuses 10.3 x 38 10A
- 10 fuses 10 x 20 2A

12.4 Spare parts common to IC robot and SC

The only spare parts common to SC and IC robot are the barcode readers (and eventually the serial cable described in 12.2). Three barcode readers will be delivered, one installed on the IC robot mask stand, and one on the VIMOS SC. The third one is a *spare*.
All barcode readers will be delivered pre-configured in EEPROM, so that they can be interchanged freely in a way transparent to the MHS program, irrespective of the fact two of them are of a later firmware release than the first one . In the rare eventuality one of the barcode readers has to be reconfigured, refer to section 9.4. The same section also specifiies the details of the differences between the two releases.

12.5 Spare parts for computers

It is intended that the spare computer will be kept in semi-cold redundancy. It will normally be kept off, except for the times quality checks with the roughness meter are necessary. Except for this it could be used as a supply of spare parts in case of hardware failures in MMCU or MHCU (the individual spare parts will act as LRUs). The spare computer is also used to host the Visual Basic development environment and the barcode EEPROM configuration software.
Alternately one could replace the entire spare system to a failed MMCU or MHCU, i.e. the entire system is a LRU. In this latter case one need to replace the hardware (replace cards in the backplane), reconfigure it and reinstall the s/w.
The idea is to simplify things arranging such that all LPKF, IFCTR and NT (i.e. Internet Peer Web server) software is installed on the system disk of all three machines. This will leave as only actions :

- The physical replacement of cards (and their eventual port configuration : if the cards were installed in the past and then removed, it is likely that the basic port configuration is already present, and the rest will be autoconfigured during the "hardware detect" phase of the boot)
- The network configuration (change network name and IP address. This is preferred to just notify the users of the other computers, including the IWSs, so that they repoint their references to the new machines , and the other MMU computers so that they update their shares. Requires no action on the other computers.
- The startup of the ftp server on MHCU

In addition we have implemented mirroring of particular data areas across different computers, so that when a new one comes up, it can be repointed to a safe copy of the data disk, namely we are cloning D and E across MHCU and MMCU, using an homegrown program scheduled periodically . Therefore the rules to handle a failure at computer level are :

- In cases of failure of *video, keyboard or mouse* of an operational computer (MMCU or MHCU) the relevant part can be unplugged from the spare computer and plugged to replace the failed part while this is being repaired..
- In case of failure of the *additional National Instrument serial card* in MHCU one could use the identical card installed in the spare computer
- A failure of the *built-in COM ports* located on the motherboard are considered as a CPU failure as described below.
- The case of failure of *barcode readers* is described above in 12.4.
- The case of failure of *LPKF supplied cards* requires LPKF staff intervention
- In case of failure of a *computer disk or CPU*, it is easier to replace the entire CPU cabinet. Two procedures are therefore possible :
  - **replace MHCU with spare**
    - swap computer with NI cards already in place
    - replug cables
    - unplug roughness meter hardware key
    - assume Ccam already instelled
    - plug in CircuitCam hardware key
    - reconfigure network
    - update D disk from MMCU Dclone

- **replace MMCU with spare**
  - leave NI cards in or remove them and keep them aside to insert in repaired spare computer
  - move LPKF cards in new computer
  - configure ports (or verify configuration)
  - assume StencilMaster already installed
  - reconfigure network
  - update E disk from MHCU Eclone

Note that all three machines are configured in such a way to be potentially replaced to each other (although we do not describe a specific procedure to replace MMCU with MHCU or viceversa). Note also that each machine running StencilMaster will have its independent set of log files and calibration files in the `C:\LPKF30\Smaster32` directory. It is recommended to run a calibration of the laser machine after replacement, in order to update the calibration tables.