XmMol Command Language For Delegate Tasks. Reference Manual.

XmMol ©, P. Tuffery†, 1993-1998.
Version 3.10

†: U436 INSERM, Equipe de Bioinformatique Génomique et moléculaire, Université Paris 7, Tour 53, 1er étage, 2 place Jussieu, 75251, Paris cedex 05.

Reference:
"XmMol: An X11 and Motif program for macromolecular visualization and modelling.", P. Tuffery, J. Mol. Graph., 1995, 13, p67-72.

June 19, 2001

Contents

Introduction

What are delegate tasks of XmMol ?

Delegate tasks are programs that are independant of XmMol, but can dialog with it. They provide an opportunity to enhance XmMol. Their main features are:

- They are standard programs that use a special library that enables information exchange (in both ways) with XmMol.
- The commands accessible to such programs make them able to pilot XmMol (most of the graphic parameters are accessible), to access to the different data fields of the molecules loaded, or to take advantage from XmMol graphic interface (X11 Widgets, mouse pick, or mouse 3D control).Thus, they are a mean to customize XmMol by implementing new features such as the superimposition of molecules, the loading of files in a format different than those built in XmMol. It is also possible to implement external renderers.
- Delegate processes do not lock XmMol while running. For example, if the delegate task is a minimiser, you can, while it is running, change the selection of the color or the atoms displayed, load other files, etc. Simply, sometimes, the coordinates of the file you are minimising will be updated.
- It is possible to make these features part of XmMol menus.
- An important feature is that these programs are not necessarily dependent from XmMol: these programs can be regular programs, that run independently from XmMol if required. The interface with XmMol is only active if these programs are forked by XmMol. (Thus, if you interface a minimiser for instance, you can run it as a numerical program or run it from XmMol for visualisation).

Delegates are called by XmMol as forked child. They must link the XmMolDelegateLib.o library that contains functions that enable communication with XmMol.

The communication is triggered via soft interruptions (SIGUSR1 & SIGUSR2) and uses fifos that are automatically assigned for each task. 1 fifo (ffi0) is common to all delegates and is used to tell XmMol wich delegate and which request, 2 other fifos are specific of each delegate are are used for data transmission (1 fifo Input (ffiX), 1 fifo Output (ffoX)). The communication protocol is the following:

- Delegate writes a request (ffi0), call XmMol and waits for acknowledge.
- XmMol reads the request (ffi0), parse it and acknowledges the delegate.
- Further data transmission then occurs (on the fifos specific of each delegate).

"A priori", many delegates can be run at a time. There is, however, no warranty that, if there are many requests, there will not be conflicts between concurrent calls. However, much can be done even if we accept that, in most cases, only one delegate will be run at a time.

REMARK: External programs that were not build as delegates can also be run from within XmMol as external commands. For such programs, a basic communication is possible through macro-expansion of coordinate filenames (See XmMol's Users Guide).

What programming languages are supported ?

The interface is C written for portability reasons. However, you could choose to write the delegates in other languages. Currently, C and FORTRAN delegates have been written and tested. *A priori*, applications written in any other language having parameter passing conventions identical to C or FORTRAN should also work. For example, Pascal compilers often propose an optional C parameter passing style.

Disclaimer

1 Basic assumptions to write a delegate

- The delegate should include the delegate lib header file:

  #include "XmMolDelegateLib.h" (C) or include 'XmMolDelegateLib.inc' (FORTRAN)

- The delegate should link the XmMolDelegateLib.o or XmMolDelegateLib_.o file.

- The delegate programs MUST perform at least 2 calls:

  XmMolConnect();

  XmMolDisconnect();

  Since a delegate could fail, there is a time out period after which the delegate is killed if it has not send any signal to XmMol. For this reason, the XmMolConnect(); call should be placed on the FIRST line of the program.

  For similar reasons (mostly for memory clean up - almost when using shared memory -, and fifo managment), the XmMolDisconnect(); should be called before the termination of the program.

- In between these two calls, information exchange can be performed via a limited series of basic functions:

  - Commands are sent using calls to XmMolSendCMsg("message"); or XmMolSendF77Msg('message')
  - Data associated with the commands is sent or get using calls to:

    XmMolGetData(&data,&sizeOfData);

    XmMolSendData(&data,&sizeOfData);

    XmMolGet<Int,Char,DtFloat>Data(&data);

    XmMolSend<Int,Char,DtFloat>Data(&data);

    XmMolGetCStringData(&data);

    XmMolGetF77StringData('NULL char terminated string', size)

  - Some specialized functions also exist:

    XmMolCGetPickRec(DtPickRec *p); // To get XmMol mouse left button pick information.

    XmMolSetGrFunc(void (*func)(), char *data); // external renderer

    XmMolCGetFileShmRec(DtXmMolShmRec *p); // Access to one file using shared memory

    XmMolCDtFileShmRec(DtXmMolShmRec *M); // Detach shared memory fields corresponding to one file

Again, the calls to these functions will not affect the program if it is not dialoging with XmMol, so that you can let them permanently in external programs.

2 A basic example

Here is an example of C program (dummyDelegate.c).

```
#include ”XmMolDelegateLib.h”
main(int argc, char *argv[])
char mess[BUFSIZ];
int i;
double pt[3];
int status;
float fstep;
double step = 1.;
char name[BUFSIZ];
int len, j, fileId;

/* Opening connection to XmMol */
status = XmMolConnect();
if (status == 0) printf(”No connection made 0);
else printf(”Connection OK
n”);

/* Sending an XmMol command that will display a message box */
XmMolSendCMsg(”XmXMessage

/* Asks XmMol to load the file 1crn.pdb, PDB format */
XmMolSendCMsg(”XmIOLoadFile 1crn.pdb DcPDB”);
XmMolGetIntData(&fileId);

/* Disconnect (At the end of the program)*/
XmMolDisconnect();
```

then, on HP for example: cc -Aa XmMolDelegateLib.o -o dummyDelegate dummyDelegate.c
(-Aa for ANSI C on HP).

3 The interface library functions (XmMolDelegateLib.o)

This section describes all the functions of the XmMolDelegateLib.o library.

General functions concerning the dialog protocole.

XmMolConnect(); Must be called once AT THE BEGINNING of the program: XmMol has to make shure that the delegate process starts, otherwise it has to flush the information concerning it and to close the fifos opened. Thus, it has a time out period until which the delegate process has to send a signal telling it is OK. If you do not place this call at the beginning of the program, depending of what is performed before it, XmMol might kill it since it will not recognize it as a delegate process.

XmMolDisconnect(); Must be called at the end of the program. Currently, it will not allow the program to continue without dialoging with XmMol.

XmMolSend<C,F77>Msg(char *cmd); All the commands must be called via the XmMolSendCMsg(arg) function, where 'cmd' is one of the instructions described part 4. Note that commands are always specified in the form of character strings. Numerical values must be converted to string. In FORTRAN, a NULL char MUST be added (for example with //char(0))

Intructions to send to or get data from XmMol.

XmMolGetData((char *) data, (int *) size of data to read); A generic function to get data from XmMol, passing an adress to the field and the lenght of it (use sizeof()). FORTRAN programmers simply write: XmMolGetData(data,size), since all arguments are adresses.

XmMolGetCharData(char *data); A shortcut for one character value.

XmMolGetIntData(int *data); A shortcut for one integer value.

XmMolGetFloatData(double *data); A shortcut for one double precision value.

XmMolGetCStringData(char *data);

XmMolGetF77StringData(data,size) A shortcut for a string. Memory is not allocated by this call. READ THE IMPORTANT NOTICE ABOVE. For FORTRAN calls, size returns the length of the string read. The last parameter declared in the XmMolDelegateLib.h is automatically set by FORTRAN. If you want to use XmMolGetF77StringData in C programs, you must add a dummy integer argument.

XmMolSendData((char *) data, (int *) size of data to read); A generic function to send data from XmMol, passing an adress to the field and its length.

XmMolSendCharData(char *data); A shortcut for one character value.

XmMolSendIntData(int *data); A shortcut for one integer value.

XmMolSendFloatData(double *data); A shortcut for one double precision value.

XmMolSendCStringData(char *data); A shortcut for a string. Memory is not allocated by this call. READ THE IMPORTANT NOTICE ABOVE.

XmMolCGetPickRec(DtPickRec *pRec); This will fill pRec with the information corresponding to the atom picked by the next left button click. See XmMolDelegateLib.h for details on DtPickRec. This doesn not allocate the memory space corresponding to the DtPickRec.

XmMolSetGrFunc(void (*func)(), char *data); This installs the function func to be the function called each time a screen refresh is needed when the external renderer mode of XmMol is used. Data is a pointer left for the user to pass some data needed by func. See the XmRndr... functions. OBSOLETE IN VERSION 3.1

XmMolSetCallbackFunc(void (*func)(), char *data); This new function of XmMol3.1 introduces some generalization for delegate callbacks. Refer to XmMolAddUsrCmd.

IMPORTANT NOTICE, 1:

The library is C written. Thus strings use the C convention and are terminated with a '\000' character. However: For STRINGS, the protocol used by XmMol is to send on the pipe one integer that is the length of the following string, followed by the 'lenght' characters. The XmMolGetCStringData() function will not return the integer corresponding to the length, but only 'length' characters. However, if you choose to use XmMolGetData(); you have to perform 2 calls:
XmMolGetIntData(&length); /* integer */ followed by
XmMolGetData(&data,length*sizeof(char)); /* The length chars */

For XmMolSend<C,F77>StringData(), the character array must end with a '\000' character (you can use char(0)). This is used to determine how many characters will be sent. This character will also be transmitted on the pipe.
For example: XmMolSendCStringData("hye") will send on the pipe an integer of value 4, then 4 characters that are 'h','y','e','\000'. DOUBLE CHECK THAT THE COMMAND YOU HAVE USED EXPECTS A STRING!! this might be a problem if you want to communicate with programs that are not c written (fortran, pascal, etc..., although the specific case of fortran is treated).

IMPORTANT NOTICE, 2:

The communication is performed through BLOCKING pipes. So, if a command requires a get data from or send data to XmMol, the programs might deadlock if the exchange is not performed (i.e., if you forget to read or write the data in the correct format).

For example:
char filename[BUFSIZ];
int fileId;
XmMolSendCMsg("XmXInqFileName %Select a file to input%");
MUST BE FOLLOWED BY:
XmMolGetIntData(&fileId);

XmMolGetCStringData(fileName);
Otherwise, (i) your next read will get this data, (ii) if the buffer is full, the programs might hang.

4 XmMol commands

This section describes all the commands supported by XmMol and their syntax. For the first functions, some examples on how they might be used are given. Other commands use the same scheme. You can also have a look to the different examples shipped with the distribution (XmMolExample0.f, XmMolExample1.c, XmMolLoadPDB.c, XmMolCamera.c, etc).

4.1 New features of XmMol 3.1

Version 3.1 comes with several modifications of the commands. Some commands have been created in order to end the transition form Seg to Frg denomination (refer to the == equivalence in the list below). A MAIN MODIFICATION is that in numerous commands, the file Id (integer) is passed instead of the filename (string). This solves the problems encountered when the same file is loaded several times. Due to this, the type DtPickRec, defined in the XmMolDelegateLib.h has been changed. New commands are introduced:

XmIODelFile

XmFileInqName (returns filename)

XmInqIdOfNthFile (returns file Id)

XmFileInqXmResNumWithName (more robust than XmFileInqXmResNum since in some PDB files, you find several residues having the same numbers and chain ids.

Enhancements have been made in the management of user menus and items. The user may know install its items or menus at any place in the menu bar. Refer to the section XmAddUserMenu.

Delegate callbacks are implemented in a new and more general way. Refer to the section XmAddUserCmd.

Some commands return different values:

XmXInqFile returns FileId + FileName (old was filename only).

XmIONewFile returns a file Id (old was none).

The commands in which the filename argument has been switched into file Id are:

XmInqFile series:
XmShmFileAccessGet, XmInqFileChnNum, XmInqFileSegNum == XmInqFileFrgNum, XmInqFileResNum, XmInqFileAtmNum.

XmFileInq series:
XmFileInqChnLimits, XmFileInqChnId, XmFileInqSegLimits == XmFileInqFrgLimits,, XmFileInqSegId == XmFileInqFrgId, XmFileInqResLimits, XmFileInqXmResNum, XmFileInqResName, XmFileInqResTraceAtmNum, XmFileInqAtmCrd, XmFileInqAtmNam, XmFileInqAtmNum, XmFileInqAtmAcode, XmFileInqAtmIcode, XmFileInqAtmResNam, XmFileInqAtmResNum, XmFileInqAtmChrg, XmFileInqAtmChn, XmFileInqAtmAtn, XmFileInqAtmTpe, XmFileInqAtmCnct, XmFileInqAtmPpty.

XmFileSet series:
XmFileSetAtmCrd, XmFileSetAtmNam, XmFileSetAtmNum, XmFileSetAtmAcode, XmFileSetAtmIcode, XmFileSetAtmResNam, XmFileSetAtmResNum, XmFileSetAtmChrg, XmFileSetAtmChn, XmFileSetAtmAtn, XmFileSetAtmTpe, XmFileSetAtmCnct, XmFileSetSegsAsChains == XmFileSetFrgsAsChains, XmFileSetAtmPpty, XmFileSetResPpty.

XmFilePut series:
XmFilePutAtmCrd, XmFilePutAtmNam, XmFilePutAtmNum, XmFilePutAtmAcode, XmFilePutAtmIcode, XmFilePutAtmResNam, XmFilePutAtmResNum, XmFilePutAtmChrg, XmFilePutAtmChn, XmFilePutAtmAtn, XmFilePutAtmTpe, XmFilePutAtmCnct, XmFilePutAtmPpty, XmFilePutResPpty.

XmDock series:
XmDockAddFileChn, XmDockDelFileChn, XmDockAddFileFrg, XmDockDelFileFrg, XmDockAddFile, XmDockDelFile.

XmBond series:
XmBondSqntlConnect, XmBondScratchConnect, XmBondRestartTables.

XmLbl series:
XmLblAtmLabel, XmLblFileResLabels, XmLblClearFileResLabels, XmLblDstLabel, XmLblLnkLabel.

Xm3D series:
Xm3DSetViewCenter.

XmIO series:
XmIOFileInitTables, XmIOWriteFile, XmIOFlushIn, XmIOPipeCrdIn, XmIOPipeCrdOut, XmIOPipeCrdRangeIn, XmIOPipeCrdRangeOut, XmIOPipeIntTopicRangeIn, XmIOPipeIntTopicRangeOut, XmIOPipeCharTopicRangeOut, XmIOPipeCharTopicRangeIn, XmIOPipeFloatTopicRangeOut, XmIOPipeFloatTopicRangeIn,

XmHide/Show series:
XmUnactivateFile, XmActivateFile, XmHideFile, XmShowFile, XmHideRange, XmShowRange, XmHideRangeH, XmShowRangeH, XmHideRangeBB, XmShowRangeBB, XmHideRangeSC, XmShowRangeSC, XmSetAtmRangeColor.

4.2 New features of XmMol 3.0

This section is left to keep a trace of old evolutions. Some of the commands described might have evolved since version 3.1.
Version 3.0 came with a modification of one command, to allow to implement new items in any menu:

"XmAddUserCmd tpe name %cmd%" has been moved into "XmAddUserCmd tpe menu name %cmd%"

Some new commands appeared (putatively modified since 3.1):

Concerning the XmIO series, new topics DrwAtm (char) and AtmCol (char) are now supported by the commands "XmIOPipeCharTopicRangeIn filename topic atomFrom atomTo" and "XmIOPipeCharTopicRangeOut filename topic atomFrom atomTo". DrwAtm somewhat superseeds XmShow/HideRange.

Some new commands concering colouring were introduced: "XmColorSetBWMode", "XmColorSetColorMode", "XmGetUsedColorNum", "XmGetInstalledColorNum", "XmColorGetNames", "XmColorSetNames", "XmColorSetOneName colId RGBName", "XmSetAtmRangeColor filename from to colId".

4.3 Miscellaneous

- **"XmRefresh"**
  return: none.
  Purpose: To provokes a refresh of the XmMol display. It calls the renderer (thus it will take into account any viewing parameter modification or coordinate, bonding, label, ..., change).

- **"XmInqDId"**
  return: XmMol internal unique Id of delegate. Necessary to know for some callbacks.

- **"XmAddUserMenu fatherName name" fatherName**: the name of the father Menu. For submenus, use a "menu:submenu:subsubmenu" string. An empty fatherName field means the menuBar.
  name: the name of the menu to be created. Will be inserted as the last item of the father menu. For submenus, use a "menu:submenu:subsubmenu" string (No spaces).
  return: none.
  This command installs an sub menu in any XmMol (sub)menu.

- **"XmFreeUserMenu menuName " menuName**: the name of the menu.For submenus, use a "menu:submenu:subsubmenu" string.
  return: none.
  This command deletes a user installed (sub)menu of XmMol.

- **"XmUserItemSetLabel fatherName name label" fatherName**: the name of the Menu containing the item to move.
  name: the name of the menu item to be affected. If empty, means the father menu itself.
  label: the new label of the menu item. Note that the reference to the menu item is unchanged. Only the label is affected.
  return: none.
  This command modifies the label of a user installed (sub)menu.

- **"XmAddUserCmd tpe menu name %cmd%"**

  tpe : must be one of:

  DcCMD for a shell command.

  DcDLG for a delagate.

DcCBK for a delagate callback. This option allows menus to trigger delegate actions instead of XmMol'. In that case, the cmd should be in the form: "dId reason" where dId is the delegate pid of the delegate, and reason the string that will be sent back to the delegate, and that must be managed by the delegate installed callback functions. Built in reasons are:

DcCBM1P, DcCBM2P, DcCBM3P for a delagate callback on mouse button press event. This option allows menus to trigger delegate actions on mouse button press.
Note 1: DcCBM1PL, DcCBM2PL, DcCBM3PL are equivalent to DcCBM1P, DcCBM2P, DcCBM3P, but will prevent XmMol to perform any action due to the corresponding mouse buttons usage.
Note 2: DcCBM1R, DcCBM2R and DcCBM3R for callbacks on button release do are implemented but are stronlgy unadvised, since they are hardly manageable due to the interrupt communication protocol. It is much better than the delegate should check whether the mouse button is released to stop its action triggered on button press for example.

DcCBRB , CBRO, CBRA for callbacks on refresh events. DcCBRB for callbacks just before refresh is performed. DcCBRO replaces the XmMol'refresh call. DcCBRA for callbacks just after refresh. These callbacks allow to totally superseed XmMol renderer.

DcCBGM : A callback due to XmMol window geometry modification.

menu: the name of the menu the command will be inserted in (always at the bottom). For submenus, use a "menu:submenu:subsubmenu" string (No spaces). If the submenu has space in its name, just use the first word. For callbacks, the name is not considered.

name: must be one word (no spaces) that is the title of the new user menu item.

cmd: The command itself. It might contain spaces and MUST BE DELIMITED BY '%'s.

return: none.
Purpose: This command installs an item in the menu bar. When pushed, the corresponding new menu item will, according to the type specified, either send a shell command (via a call to the C system facility) or fork a delegate.

example:
XmMolSendCMsg("XmAddUserCmd DcCMD File myLS %ls -al%");
Will install an item "myLS" in the File menu. When used, this button will execute the shell command "ls -al".

XmMolSendCMsg("XmAddUserCmd DcDLG File:Open PDBinput %loadPDB%");
Will install an item "PDBinput" in the sub menu Open of the File menu. When used, this button will fork the delegate "loadPDB". This will be effective only if the loadPDB

program is accessible and was built as a delegate.

XmMolSendCMsg("XmAddUserCmd DcDLG User Animate %XmMolCamera%");
This will install the delegate "XmMolCamera" as the "Animate" item of the "User" menu. If the menu "User" was not displayed (no item installed, the default), it will now be.

sprintf(buff,"XmAddUserCmd DcCBK Forme Reset %d reset%",pid);
XmMolSendCMsg(buff);
would install in a menu Forme an item that, when selected will callback the delegate and pass him the string "reset".

- **"XmFreeUserCmd fatherName name" fatherName**: the name of the menu in which the command appears. For submenus, use a "menu:submenu:subsubmenu" string.
  name: the name of the command.
  return: none.
  This command deletes a user installed command.

- **"XmSetDisplayMode mode"**
  mode: one of RIBBON, TRACE or WIREFRAME.
  return: none.
  Purpose: All the "active" files will be rendered in that mode.

- **"XmRedoSplines"**
  return: none.
  Triggers re-computation of splines.

4.4 Some possibilities to activate X11 widgets or to get information about X11 features

These command allow any delegate to use some widgets managed by XmMol. When invoking these commands, the delegate will stop until the X11 widget input is performed.

- **"XmXInqFileName %text%"**
  text (optional): MUST BE DELIMITED BY '%'s. It corresponds to a help string displayed within the widget.
  return: a file Id (integer) and a string (the filename) to the delegate.
  Purpose: This command will display a file selector that allows to select one of the files that have been input by XmMol.
  example:
  char filename[BUFSIZ];
  XmMolSendCMsg("XmXInqFileName %Select a file to input%");
  XmMolGetStringData(fileName); /* This MUST be present. Otherwise you will deadlock your delegate */

- **"XmXInqFileBrowser %selection mask%"**
  selection mask (optional): MUST BE DELIMITED BY '%'s. A mask used to filter the

possible file names. Some problems have been reported when using a "*.*" as mask.
return: a string to the delegate, i.e. the name of the file selected.
Purpose: This command will display a motif file brwoser that allows to select one file name.
example:
char filename[BUFSIZ];
XmMolSendCMsg("XmXInqFileBrowser %*.pdb%");
XmMolGetStringData(fileName); /* This MUST be present. Otherwise you will deadlock your delegate */

- **"XmXInqString %text% defaultValue"**
  text (optional): MUST BE DELIMITED BY '%'s. A help string for the widget displayed.
  defaultValue (optional): A proposal for the string. Numbers must be passed converted to string format.
  return: a string to the delegate. Numbers should be converted from string.
  Purpose: This command will display a dialog box that will allow you to type in a text tath will be returned to the delegate.
  example:
  char filename[BUFSIZ];
  XmMolSendCMsg("XmXInqString %Name of the file to write:% sample.pdb");
  XmMolGetStringData(fileName); /* This MUST be present. Otherwise you will deadlock your delegate */

- **"XmXInqRange tpe val1 val2"**
  tpe (optional): must be one of 'DcInt' or 'DcFloat'.
  val1, val2 (optional): 2 default values, string format.
  return: 2 integers or doubles according to tpe.
  Purpose: This command will manage the display of a widget that asks you a two values. Might be integers or doubles, according to tpe.

  example:

  int from, tto, range[2];
  XmMolSendCMsg("XmXInqRange DcInt 10 100");
  XmMolGetIntData(&from); /* MUST be there to avoid deadlock of delegate */
  XmMolGetIntData(&tto);
  /* XmMolGetData(&range, sizeof(int)*2); is another possibility for the same*/

- **"XmXMessage %txt%"**
  txt: A message written in the widget displayed. MUST BE DELIMITED BY '%'s. The string can contain newline characters.
  return: none.
  Purpose: This command will display a message widget with txt in it. The delegate will be stopped until you press the OK button.

- **"XmXInqYesNo %txt%"**
  txt: The question written in the widget displayed. MUST BE DELIMITED BY '%'s.
  return: an integer value, 1 for Yes, 0 for No.
  Purpose: This command will display a message widget to ask a question. The delegate will be stopped until you press the Yes or No button.

- **"XmXInqPick"**
  return: After you pick with the left button, the content of a DtPickRec is sent to the delegate. The delegate must wait for this information.
  Purpose: This command allows a delegate to get the left button pick information sent to the delegate instead of XmMol. The detail of the structure used to pass the information is within the XmMolDelegateLib.h (DtPickRec structure). This facility is illustrated within the XmMolsuperImp.c example.

  example:
  ```
  DtPickRec pTFrom;
  XmMolSendCMsg("XmXInqPick");
  XmMolCGetPickRec(&pTFrom);
  ```

- **"XmXRefresh" return**: none
  To refresh XmMol display. Forces the delegate to wait the refresh is achieved (Thus different from XmRefresh).

- **"XmXInqCmap"**
  return: the colormap used by XmMol. (sizeof Colormap).
  Purpose: This command will return the Id of the colormap used by XmMol. Might be used for external renderers.
  example:
  ```
  XmMolSendCMsg("XmXInqCmap");
  size = sizeof(Colormap);
  XmMolGetData(&cmapId,&size);
  ```

- **"XmXSetCmap"**
  return: none.
  Purpose: This command will force the colormap of id "Id" to be used by XmMol.
  example:
  ```
  XmMolSendCMsg("XmXSetCmap");
  size = sizeof(Colormap);
  XmMolSendData(&cmapId,&size);
  ```

- **"XmXResetCmap"**
  return: none.
  Purpose: This command will make XmMol use its default colormap. It should be used after a call to XmXSetCmap, to restore XmMol's Colormap.

- **"XmXInqWindowId"**
  return: a string that contains the Id of the XmMol' graphic window.

- **"XmXInqGeom"**
  return: 3 integer values: the current width, height and depth of the graphic window.
  Purpose: This command will return the geometry of the graphic window of XmMol. May be usefull for external renderers.

4.5 Getting general information about files

- **"XmInqCurFileNum"**
  return: one integer, the number of file currently opened by XmMol. Must be followed by a XmMolGetIntData(...);
  Purpose: To get the number of files currently loaded.

- **"XmInqIdOfNthFile num"**
  num: the nth file opened by XmMol (WARNING: starts from 0, and should be less than "XmInqCurFileNum").
  return: one integer, XmMol's internal Id of the Nth file opened. Must be followed by a XmMolGetIntData(...);
  Purpose: To get the Id used by XmMol for the file in the nth position. The Id is also displayed within the file summary box. Each file has a unique Id during a session. Thus, no confusion is possible.

- **"XmInqNameOfNthFile id"**
  id: the nth file opened by XmMol (WARNING: starts from 0, and should be less than "XmInqCurFileNum").
  return: one string, the name of the Nth file opened by XmMol. Must be followed by a XmMolGetStringData(...);
  Purpose: To get the actual name (disk name) of the file.

- **"XmInqFileChnNum fileId"**
  fileId: the XmMol Id of the concerned file. Might have been obtained by "XmInqName-OfNthFile id".
  return: 1 integer, the number of chains of the file 'fileId'. Must be followed by a Xm-MolGetIntData(...);
  Purpose: To get the number of different chains detected in a file. Note that if the atoms of different chains are intertwined, the number returned will be more than the actual number of chain Ids of the file. In fact XmMol also has another chain number that is the effective chain Id number of the file, but that is not accessible to delegate tasks. The delegate can however check that via calls to "XmInqFileNthChnId".

- **"XmInqFileFrgNum fileId"**
  fileId: the XmMol Id of the concerned file. Might have been obtained by "XmInqName-OfNthFile id".
  return: 1 integer, the number of segments of the file 'fileId'. Must be followed by a XmMolGetIntData(...);
  Synonym: XmInqFileSegNum is the old notation of this command and should be avoided.

Purpose: To get the number of different fragments detected in a file. Fragments are detected as 2 consecutive residues not classically connected together (ex: for protein, XmMol searches for a Ci-Ni+1 bond.If the connection is made by different atoms the number returned will be more than the actual number of segments colored by XmMol. In fact XmMol also has another fragment number that is the effective fragment number of non connected fragments, but that is not accessible to delegate tasks. The delegate can however check that via calls to XmInqFileNthFrgId. The number of segments might be great if the file has many H2O.

- **"XmInqFileResNum fileId"**
  fileId: the XmMol Id of the concerned file. Might have been obtained by "XmInqNameOfNthFile id".
  return: 1 integer, the number of residues of the file 'fileId'. Must be followed by a XmMolGetIntData(...);
  Purpose: To get the number of residues of a file.

- **"XmInqFileAtmNum fileId"**
  fileId: the XmMol Id of the concerned file. Might have been obtained by "XmInqNameOfNthFile id".
  return: the number of atoms of the file 'fileId'. Must be followed by a XmMolGetIntData(...);
  Purpose: To get the number of atom within a file.

  example:

```
char buf[BUFSIZ];
char name[BUFSIZ];
int nMols;
int nat;
int nres;

XmMolSendCMsg("XmInqCurFileNum");
XmMolGetIntData(&nMols);
if (nMols)
XmMolSendCMsg("XmInqNameOfNthFile 0");
XmMolGetStringData(name);
sprintf(buf,"XmInqFileAtmNum %s",name);
XmMolSendCMsg(buf);
XmMolGetIntData(&nat);
sprintf(buf,"XmInqFileResNum %s",name);
XmMolSendCMsg(buf);
XmMolGetIntData(&nres);
printf("The first file loaded by XmMol is %s and has %d atoms, %d res0,name, nat, nres);
```

4.6 Getting information about file chains, segments and residues

- **"XmFileInqName fileId"**
  fileId: the XmMol Id of the concerned file. Might have been obtained by "XmInqNameOfNthFile id".
  return: 1 string, the name fo the file with the associated Id. **Purpose**: To get the disk filename of the file associated with the Id fileId.

- **"XmFileInqChnLimits fileId chnnum"**
  fileId: the XmMol Id of the concerned file. Might have been obtained by "XmInqNameOfNthFile id".
  chnnum: the number of the chain concerned. Starts from 0.
  return: 2 integer, the atom numbers (from 0) corresponding to the first and last atom of the chain number 'chnnum' (from 0) of the file 'fileId'. Must be followed by 2 calls to XmMolGetIntData(...); **See Also**: "XmInqFileChnNum" **Purpose**: To get the number of the residues delimiting a chain.

- **"XmFileInqChnId fileId chnnum"**
  fileId: the XmMol Id of the concerned file. Might have been obtained by "XmInqNameOfNthFile id".
  chnnum: the number of the chain concerned. Starts from 0.
  return: 1 integer, the chain Id of the chain number 'chnnum' (from 0) of the file 'fileId'. Must be followed by 1 call to XmMolGetIntData(...);
  Purpose: To get the chain Id of the fragment. See "XmInqFileChnNum".

- **"XmFileInqFrgLimits fileId frgnum"**
  fileId: the XmMol Id of the concerned file. Might have been obtained by "XmInqNameOfNthFile id".
  frgnum: the number of the fragment concerned. Starts from 0.
  return: 2 integer, the atom numbers (from 0) corresponding to the first and last atom of the segment number 'frgnum' (from 0) of the file 'fileId'. Must be followed by 2 calls to XmMolGetIntData(...);
  See Also: "XmInqFileFrgNum"
  Synonym: XmFileInqSegLimits is the old notation of this command and should be avoided.
  Purpose: To get the indices of the atoms within a fragment.

- **"XmFileInqFrgResLimits fileId frgnum"**
  fileId: the XmMol Id of the concerned file. Might have been obtained by "XmInqNameOfNthFile id".
  frgnum: the number of the fragment concerned. Starts from 0.
  return: 2 integer, the residue numbers (from 0) corresponding to the first and last residue of the fragment number 'frgnum' (from 0) of the file 'fileId'. Must be followed by 2 calls to XmMolGetIntData(...);
  See Also: "XmInqFileFrgNum"
  Purpose: To get the indices of the residues within a fragment.

- **"XmFileInqFrgId fileId frgnum"**
  fileId: the XmMol Id of the concerned file. Might have been obtained by "XmInqNameOfNthFile id".
  frgnum: the number of the fragment concerned. Starts from 0.
  return: the fragment Id of the segment number 'frgnum' (from 0) of the file 'fileId'. Must be followed by 1 call to XmMolGetIntData(...);
  Synonym: XmFileInqSegId is the old notation of this command and should be avoided.
  Purpose: To get the fragment Id of the nth fragment. See "XmInqFileFrgNum". This helps identifying fragments non conventionally linked.

- **"XmFileInqResLimits fileId resnum"**
  fileId: the XmMol Id of the concerned file. Might have been obtained by "XmInqNameOfNthFile id".
  resnum: the number of the residue concerned. Starts from 0.
  return: 1 integer, the atom numbers (from 0) corresponding to the first and last atom of the residue number 'resnum' (from 0) of the file 'fileId'. Must be followed by 2 calls to XmMolGetIntData(...);
  Purpose: To get the number (from 0) of the atoms delimiting a residue.

- **"XmFileInqXmResNum fileId resnum <chain>"**
  fileId: the XmMol Id of the concerned file. Might have been obtained by "XmInqNameOfNthFile id".
  resnum: the number of the residue concerned in the file numbering. Might not starts from 0.
  chain: if present, the chain Id of the residue concerned. If not, it is assigned to a blank character.
  return: 1 integer, the XmMol internal residue number (from 0) of the residue of number 'resnum' (and chain 'chain') in file 'fileId'. Must be followed by a XmMolGetIntData(...);
  Purpose: To get teh internal residue number of a residue. XmMol numbers residues from 0 (since it is C). Residue numbers specified in a file might be different.
  Warning: Some files might have two residues with the same number and the same chain Id, even in the PDB. In that case, the first occurence will be returned.

- **"XmFileInqXmResNumWithName fileId resnum resname <chain>"**
  fileId: the XmMol Id of the concerned file. Might have been obtained by "XmInqNameOfNthFile id".
  resnum: the number of the residue concerned in the file numbering. Might not starts from 0.
  resnum: the name of the residue (3 chars + '\000').
  chain: if present, the chain Id of the residue concerned. If not, it is assigned to a blank character.
  return: 1 integer, the XmMol internal residue number (from 0) of the residue of number 'resnum' (and chain 'chain') in file 'fileId'. Must be followed by a XmMolGetIntData(...);
  Purpose: To get teh internal residue number of a residue. XmMol numbers residues from 0 (since it is C). Residue numbers specified in a file might be different. More robust than XmFileInqXmResNum.

- **"XmFileInqResName fileId resnum <chain>"**
  fileId: the XmMol Id of the concerned file. Might have been obtained by "XmInqNameOfNthFile id".
  resnum: the number of the residue concerned. Starts from 0 (ie number is as in internal XmMol numbering).
  return: 1 string(4) (1 integer + 4 chars), the XmMol Id of the residue number 'resnum' (from 0) of the file 'fileId'. The 4 characters correspond to 3 character that are the name, followed by '\000'. Must be followed by a XmMolGetStringData(...);
  Purpose: To get the name of a residue.

4.7 Getting information about one atom

All these calls must be followed by the appropriate calls to XmMolGetData(...);
All these calls have two arguments:

fileId : an argument which corresponds to the XmMol internal Id of the file. This might have been obtained by invoking "XmInqIdOfNthFile nth". where nth is a number between 1 and the result of a call to "XmXInqCurFileNum".

anum : the number of the atom concerned. Starts from 0, up to "XmInqFileAtmNum" - 1.

- **"XmFileInqAtmCrd fileId anum"**
  return: 3 double, the x, y and z coordinates. **Purpose**: to get the 3D coordinates of the atom. The coordinates sent are not converted in the eye reference but are in the original reference. Thus, they are identical to that of the file loaded, except if some docking or modelling was performed.

- **"XmFileInqAtmNam fileId anum"**
  return: 1 string(5), (1 integer + 5 characters, including 1 ending '\000'), the XmMol Id of the atom. **Purpose**: To get the name of the atom. The name is completed with blanks up to 4 characters if necessary.

- **"XmFileInqAtmNum fileId anum"**
  return: 1 integer, the number of the atom as given by file numbering (might not start at 0). **Purpose**: To get the number of the atom as set in the file loaded. Thus, it might be different of a simple count of the number of atoms within the file.

- **"XmFileInqAtmAcode fileId anum"**
  return: 1 character, the alternate code from PDB files. **Purpose**: To get the alternate code set for the atom in a PDB file.

- **"XmFileInqAtmIcode fileId anum"**
  return: 1 character, the insertion code from PDB files. **Purpose**: To get the insertion code set for the atom in a PDB file.

- **"XmFileInqAtmResNam fileId anum"**
  return: 1 string(4), (1 integer + 3 characters, including 1 ending '\000'), the XmMol Id of the residue of the atom. **Purpose**: To get the name of the residue of the atom. The name is completed with blanks up to 3 characters if necessary.

- **"XmFileInqAtmResNum fileId anum"**
  return: 1 integer, the residue number of the atom as given by file numbering (might not start at 0). **Purpose**: To get the number of the residue of the atom as set in the file loaded. Thus, it might be different of a simple count of the number of atoms within the file.

- **"XmFileInqAtmChrg fileId anum"**
  return: 1 double corresponding to the charge of the atom. **Purpose**: To get the charge associated with the atom. This makes only sense for mac files, or if a delegate process has set this field.

- **"XmFileInqAtmChn fileId anum"**
  return: 1 character, the chain label of the atom. **Purpose**: To get the chain Id of this atom.

- **"XmFileInqAtmAtn fileId anum"**
  return: 1 integer, the atomic number of the atom. **Purpose**: To get the atomic number associated with the atom. (DO NOT USE IF NOT FAMILLIAR WITH THE MAC FORMAT).

- **"XmFileInqAtmTpe fileId anum"**
  return: 1 integer, the atomic class of the atom. **Purpose**: To get the atomic class of teh atom. (DO NOT USE IF NOT FAMILLIAR WITH THE MAC FORMAT)

- **"XmFileInqAtmPpty fileId anum"**
  return: 1 double,the property value associated with the atom. **Purpose**: To get the value of a property associated with the atom. This makes only sense for if a delegate process has set this field.

- **"XmFileInqAtmCnct fileId anum"**
  return: 7 integers (c1 .. c6, nc). The last (nc) is the number of bonds involving this atom. the 6 first are the numbers of the atoms involved in these bonds. c1 .. c6 are integers that start from 1 for the first atom of the file, allowing no gap between the numbers. These numbers might be different from explicit atom numbers of a PDB file! For fields that are not used (less than 6 bonds), a 0 value is set. Double bonds are represented by negative values. **Purpose**: To get the description of the atoms covalently bonded to the atom. (Better to be familliar with the mac format to use it)

WARNING: these numbers are from 1 and not from 0. (kept for compatibility with old programs)

4.8 Modifying file atom fields (text format)

A text way to setup atomic values for the different fields described in the previous sections. Frequent arguments (not detailed again below) are:
fileId: the XmMol id of the file. You can either directly read it from the file summary box of XmMol interface, or get it as the returned value from XmIOLoadFile, XmIONewFile or XmInqIdOfNthFile.
atom: the number of the atom. The number is set as 0 for the first atom of the file.
res: the number of the residue. The number is set as 0 for the first residue of the file.
return: none.

- **"XmFileSetAtmCrd fileId atom x y z"**
  x y z: 3 double values, the new 3D coordinates of the atom.

- **"XmFileSetAtmNam fileId atom atomName"**
  atomName: the new name of the atom. 4 chars will be considered at maximum. If less than 4 are passed, other locations up to 4 are set to ' ' (i.e. the name is left justified).

- **"XmFileSetAtmNum fileId atom atomNumber"**
  atomName: the new atom number of the atom. This corresponds to the atom number that will be written on disk for most output options. This is also what is displayed when you pick the atom. Note that this does not correspond to the same numbering as the "atom" argument, that is a simple indication of the rank of the atom from the beginning of the file.

- **"XmFileSetAtmAcode fileId atom atomalternatecode(1 char)"**
  atomalternatecode: 1 char that strictly corresponds to the alternate code of the PDB file format.

- **"XmFileSetAtmIcode fileId atom atominsertioncode(1 char)"**
  atominsertioncode: 1 char that strictly corresponds to the insertion code of the PDB file format.

- **"XmFileSetAtmResNam fileId atom atomresiduename"**
  atomresiduename: The new residue name of the atom. 3 chararacters will be considered at maximum. If less than 3 are passed, other are set to ' ' (i.e. the name is left justified). Better if you set the same name for all the atoms of the residue.

- **"XmFileSetAtmResNum fileId atom atomresiduenumber"**
  atomresiduenumber: The new residue number of the atom. As for the atom number, this number is used when you pick an atom, or when you write the file on the disk. It is different of a simple count of the residues from the beginning of the file.

- **"XmFileSetAtmChrg fileId atom atomCharge"**
  atomCharge: You can use this function to assign the charge field used by the XmMol format. By default, the PDB charge fields of the PDB files are set to 0.

- **"XmFileSetAtmChn fileId atom atomChain(1 char)"**
  atomChain: The new chain descriptor of the atom. Much better to use a letter.

- **"XmFileSetAtmAtn fileId atom atomicNumber"**
  atomicNumber: This corresponds to the atomic number set in the XmMol's internal format. Do not use if not familiar with mac (FLEX) format.

- **"XmFileSetAtmTpe fileId atom atomicClass"**
  atomicClass: The flex atomic class of the atom. Do not use if not familiar with mac format.

- **"XmFileSetAtmCnct fileId atom c1 c2 c3 c4 c5 c6 nc"**
  c1 .. c6: Integers that describe the connectivities of the atom. c1 to c6 are the numbers of the atoms linked as their order of occurrence within the file, BUT NUMBERED FROM 1 (for compatibility with flex). For fields that are not used (less that 6 connectivities), simply set a 0 value. Double bonds are represented by negative values.
  nc: The total number of covalent links of the atom.
  Calls to this command SHOULD ALWAYS BE FOLLOWED BY A CALL TO "XmBondRestartTables fileId" (once the connectivities have all been set).

- **"XmFileSetFrgsAsChains fileId 1stChainLabel(1 char)"**
  Synonym: XmFileSetSegsAsChains **1stChainLabel**: All the fragments are setup as different chains. The labels are incremented from the first value (better to use a letter as Id).

- **"XmFileSetAtmPpty fileId atom atomicpropertyvalue"**
  atomicpropertyvalue: A field to assign a property value to the atom (floating format). This value can be used in conjunction with the color by property value facility.

- **"XmFileSetResPpty fileId res residuepropertyvalue"**
  residuepropertyvalue: A field to assign a property value to the residue (floating format). It sets the same property value for each atom of the residue. Can be used in conjunction with the color by property value facility.

4.9 Modifying file atom fields (binary format)

These functions correspond to the exact pending of the calls described in the section *Getting information about one atom* , except for the commands passing atomic or residue names. Compared to the previous section *Modifying file atom fields (text format)*, this section sets the values up by sending directly the information in the binary format. There is no need to convert in text format. Thus, it should be faster. All these calls must be followed by the correct XmMolSendData(...)s. Frequent arguments (not detailed again below) are:
fileId: the XmMol id of the file. You can either directly read it from the file summary box of XmMol interface, or get it as the returned value from XmIOLoadFile, XmIONewFile or XmInqIdOfNthFile.
anum: the number of the atom. The number is set as 0 for the first atom of the file.
res: the number of the residue. The number is set as 0 for the first residue of the file.
return: none.
REMARK: some of the commands of the XmIO section allow to access some fields for ranges

of atoms. This might be much faster that consecutive simple calls to the functions described in this section. As well, using the sharing memory facilities can also give much better results.

- **"XmFilePutAtmCrd fileId atom"**
  data to send: 3 doubles, corresponding to the x, y and z coordinates.

- **"XmFilePutAtmNam fileId atom"**
  data to send: NOT A STRING, just 5 characters, including ending '\000'). If the name is less than 4 characters long, it must be filled with blanks.
  example:
  char newName[5];
  strcpy(newName,"HG11");
  XmMolSendCMsg("XmFilePutAtmNam toto.mac 10");
  XmMolSendData(newName,sizeof(char)*5);
  DO NOT USE XmMolSendStringData(newName);

- **"XmFilePutAtmNum fileId atom"**
  data to send: 1 integer value.

- **"XmFilePutAtmAcode fileId atom"**
  data to send: 1 character.

- **"XmFilePutAtmIcode fileId atom"**
  data to send: 1 character.

- **"XmFilePutAtmResNam fileId atom"**
  data to send: NOT A STRING, just 4 characters, including ending '\000'. If the name is less than 3 characters long, it must be filled with blanks.
  example:
  char newName[4];
  strcpy(newName,"C "); /* Cytosin */
  XmMolSendCMsg("XmFilePutAtmResNam toto.mac 0"); /* First atom of file */
  XmMolSendData(newName,sizeof(char)*4); /* DO NOT USE XmMolSendStringData(newName);*/

- **"XmFilePutAtmResNum fileId atom"**
  data to send: 1 integer.

- **"XmFilePutAtmChrg fileId atom"**
  data to send: 1 double.

- **"XmFilePutAtmChn fileId atom"**
  data to send: 1 character.

- **"XmFilePutAtmAtn fileId atom"**
  data to send: 1 integer. Do not use if not familiar with the mac format.

- **"XmFilePutAtmTpe fileId atom"**
  data to send: 1 integer. Do not use if not familiar with the mac format.

- **"XmFilePutAtmCnct fileId atom"**
  data to send: 7 integers, see their significance upper. THIS CALL SHOULD ALWAYS BE FOLLOWED BY A CALL TO "XmBondRestartTables fileId" (once the all the connectivities modifications have been performed).

- **"XmFilePutAtmPpty fileId atom"**
  data to send: 1 double.

- **"XmFilePutResPpty fileId res"**
  data to send: 1 double (the same value is assigned for each atoms of the residue).

4.10 Input/Output in various ways

- **"XmIOLoadFile filename fmt"**
  filename: the name of the concerned file. Might have been obtained by "XmInqNameOfNthFile id".
  fmt: must be one of DcXmMol or DcPDB.
  return: an Id for the file loaded.
  XmMol will load the file of name 'filename' of format 'fmt'. Exactly the same as if you do that with the menu.

- **"XmIOLoadCmdFile filename"**
  filename: the name of the concerned command file. 1 command per line. See example in XmMol.doc1 or the XmMol.cmd sample file provided with teh distribution. Might have been obtained by "XmXInqFileBrowser". XmMol will load the file of name 'filename', and run the commands (if possible in this mode). Exactly the same as if you do that with the menu **File:Open:XmMolCmd**.

- **"XmIOWriteFile XmMolFileId diskFileName fmt mode options"**
  XmMolfileId: the XmMol Id of the concerned file, already loaded by XmMol. Might have been obtained by "XmInqNameOfNthFile id".
  diskFilename: the name of the disk file. Might have been obtained by "XmXInqString".
  fmt: must be one of DcXmMol, DcPDB or DcXYZ.
  mode: a flag that indicates what files loaded by XmMol are concerned by output. It must have one of the following value (and not a combination)(0 is not allowed):

  1 : only 1 file is concerned, which name is that given by XmMolFileId.

  2 : all files are concerned. XmMolFileId MUST BE SET, for compatibility, but will not be used.

  4 : only active files are concerned. XmMolFileId MUST BE SET, for compatibility, but will not be used.

  options: a flag that indicates how files selected by the mode flag will be output. A 0 default value is allowed. Possible masks are (they might be combined):

3 : perform split of files by chains. Each chain will be written separately in 1 file. The files will have the generic name diskFileNamechn¡number¿.¡mac, pdb, xyz¿. The 3 mask is incompatible with the 5 mask.

5 : perform split of files by segments. Each segment will be written separately in 1 file. The files will have the generic name diskFileNameseg¡number¿.¡mac, pdb, xyz¿. The 5 mask is incompatible with the 3 mask.

16 : set the output to only displayed atoms

128 : renumber atoms (from 1). This will only be effective if you perform PDB output.

examples:
XmMolSendCMsg("XmIOWriteFile toto.mac tutu.pdb DcPDB 1 0");
will write a tutu.pdb (PDB format) file, that corresponds to the atoms of the currently loaded toto.mac file. All atoms are concerned, no renumbering is performed.
XmMolSendCMsg("XmIOWriteFile toto.mac tutu.pdb DcPDB 2 3");
will write PDB files of name tutuchnxx.pdb, one for per chain of the files currently loaded by XmMol. Systematic renubering of each file from 1 could be obtained replacing 3 by 3+128 = 131. Subsequent restriction to only displayed atoms of each chains would be obtained using the flag 131+16 = 147. Also toto.mac is mentionned, it is not taken into account.

- **"XmIODelFile fileId"**
fileId: the XmMol Id of the file. Might have been obtained by "XmInqIdOfNthFile id", or as a returned value of the XmXInqFile
return: none.
Purpose: XmMol will destroy the file with Id fileId. Exactly the same as if you do that with the delete button of the file summary box.

- **"XmIOFlushIn XmMolFileId diskFilename fmt"**
XmMolFileId: the XmMol Id of the concerned file, already loaded by XmMol. Might have been obtained by "XmInqNameOfNthFile id".
diskFilename: the name of the disk file. Might have been obtained by "XmXInqString".
fmt: must be one of DcXmMol, DcPDB or DcXYZ.
This will overwrite the coordinates of the loaded file 'XmMolfileId' by the coordinates of the disk file 'diskfilename'. No checking of atom numbers. You must make shure before that it is correct.

- **"XmIOPipeCrdRangeIn fileId atomFrom atomTo"**
fileId: the XmMol Id of the concerned file. Might have been obtained by "XmInqNameOfNthFile id".
atomFrom: the number of the starting atom (from 0).
atomTo: the number of the ending atom (included)(from 0).
This will overwrite the coordinates of the atoms number atomFrom to atomTo of the loaded file 'fileId' by the coordinates that are send to the pipe. The coordinates are flushed within XmMol, so the delegate must make the correct call(s) to XmMolSendData(...);

example:
double atmCrd[10][3]; /* Supposed initialized */
...
XmMolSendCMsg("XmIOPipeCrdRangeIn toto.mac 0 10");
for(i=0; i¡10; ++i) XmMolSendData(atmCrd[i], sizeof(double)*3);

- **"XmIOPipeCrdRangeOut fileId atomFrom atomTo"**
  The exact reverse of XmIOPipeCrdRangeIn. The coordinates are flushed within the delegate. So the delegate must make the correct call(s) to XmMolGetData(...);
- **"XmIOPipeCrdIn fileId"**
  This will overwrite the coordinates of the loaded file 'fileId' by the coordinates that are send to the pipe. The same as "XmIOPipeCrdRangeIn fileId atomFrom atomTo" with atmFrom = 0 and atomTo = fileId.atomnumber - 1. The coordinates are flushed within XmMol, so the delegate must make the correct call(s) to XmMolSendData(...);
- **"XmIOPipeCrdOut fileId"**
  The exact reverse of XmIOPipeCrdIn. The coordinates are flushed within the delegate, so the delegate must make the correct call(s) to XmMolGetData(...);
- **"XmIOPipeIntTopicRangeIn fileId topic atomFrom atomTo"**
- **"XmIOPipeIntTopicRangeOut fileId topic atomFrom atomTo"**
- **"XmIOPipeCharTopicRangeIn fileId topic atomFrom atomTo"**
- **"XmIOPipeCharTopicRangeOut fileId topic atomFrom atomTo"**
- **"XmIOPipeFloatTopicRangeIn fileId topic atomFrom atomTo"**
- **"XmIOPipeFloatTopicRangeOut fileId topic atomFrom atomTo"**
  A series of commands that allow grouped access to some fields of each file. Atom numbers are from 0. 'fileId' is as given elsewhere. Allowed topics may vary according to the command:
  for Int:

    AtmNum (1 integers for each index)

    ResNum (1 integers for each index)

    Atn (1 integers for each index)

    Tpe (1 integers for each index)

    s2 (1 integers for each index) (secondary structure type)

Cnct (7 integers for each index) SHOULD ALWAYS BE FOLLOWED BY A CALL TO "XmBondRestartTables fileId" (once the connectivities have all been set).

for Char:

ACode (1 char for each index)

ICode (1 char for each index)

Chn (1 char for each index)

AtmNam (4 char + '\000' for each index (= 5 char))

ResNam (3 char + '\000' for each index (= 4 char))

DrwAtm (1 char for each index, 1 for drawn, 0 for hidden)

AtmCol (1 char for each index, color num per atom)

for Float:

Chrg (1 doubel for each index)

Ppty (1 doubel for each index)

Crd (3 double for each index, the same as "XmIOPipeCrdRange")

- **"XmIONewFile filename nat nres"**
filename: the name of the concerned file. Might have been obtained by "XmXInqString".
nat: number of atoms of the file.
nres: number of residues of the file.
return: a file Id (integer).
purpose: This function is left to give an opportunity to input any format files within XmMol. It only ensures to allocate a new file space corresponding to nat and nres. All fields are empty and you should fill them using "XmIOPipeXXXTopicRangeIn" or XmFilePutxxx calls. ALL THE FIELDS must be HAND SET (all atom fields and the connectivity matrix). AFTER THIS HAS BEEN DONE, YOU MUST CALL ONCE THE "XmFileSetupTables" FUNCTION, SO THAT THE FILE IS CORRECTLY INTERFACED WITH XMMOL. NO WARRANTY IF YOU DO NOT DO THINGS THIS WAY, OR IF YOU INPUT OTHER FILES BEFORE HAVING COMPLETED ALL THIS PROTOCOL.
example: refer to the XmMolLoadPDB.c delegate provided with the distribution.

- **"XmIOFileInitTables fileId"**
fileId: the XmMol Id of the concerned file. Might have been obtained by "XmXInqString".
purpose: This function will perform anything so that the display of the file is correctly performed. IT MUST BE CALLED ONCE FOR EACH FILE CREATED BY "XmNewFile". DO NOT USE WITH OTHER FILES. **example**: refer to the XmMolLoadPDB.c delegate provided with the distribution.

4.11 About rendering and 3D transformations

A series of functions to perform the corresponding operations of the Render menu.
The way the 3D transormation matrices are used is as follows:
{x',y',z',dpth} = E2CG x Tr x St x Rt x Sc x CG2E x W2E (x Dck) {x, y, z, 1}.
where:
{x, y, z,1} are the 3D coordinates of the point in the space (world reference, homogeneous coordinates).
{x', y', z',dpth} are the 3D coordinates of the transformed point in the space (eye reference, homogeneous coordinates).
Dck: A putative 4x4 docking matrix, in the world reference.
W2E: The 4x4 transition matrix from world reference to eye reference.
CG2E: The 4x4 transition matrix to superimpose the centre of move of the object with the origin of the eye reference.
Sc: A 4x4 scaling matrix. (scaling on the diagonal, all other locations null.)
Rt: A 4x4 rotation matrix.
St: A 4x4 additional rotation matrix for stereo.
Tr: A 4x4 translation matrix. (translation values on the last column).
E2CG: The inverse of CG2E, to make move as if relative to the center of move.

The final plan projection is then on the form:
{xp,yp,0,0} = PP x {x', y', z',dpth}
where PP corresponds to the plane projection, and takes the perspective paramters into account.
extra arguments to pass: none.
return: none.

- **"Xm3DClippingOff"**
- **"Xm3DClippingOn"**
- **"Xm3DStereoOff"**
- **"Xm3DStereoOn"**
- **"Xm3DDCueOff"**
- **"Xm3DDCueOn"**
- **"Xm3DSingleBuffer"**
- **"Xm3DDoubleBuffer"**

A series of functions to perform 3D eye transformations.
extra arguments: should be obvious for the different commands. **return**: none.

- **"Xm3DRotReset"**
  Purpose: Makes the rotation matrix the identity matrix.

- **"Xm3DXRot x"**
  x: Rotation value around the eye reference X axis. Value is in radians.

- **"Xm3DYRot y"**

- **"Xm3DZRot z"**

- **"Xm3DScaleReset"**
  Purpose: Makes the scaling matrix the the identity matrix.

- **"Xm3DScale x"**
  x: The scaling (floating format) value to concatenate with the current scaling matrix. (Not a good idea to assign x to 0).

- **"Xm3DTrnsReset"**
  Purpose: Makes the translation matrix the identity matrix.

- **"Xm3DTrns x y z"**
  Purpose: Makes a translation by x y z (floating format).

- **"Xm3DBClip dz"**
  Purpose: Delta of positionning (floating format) of the Back Clip plane. The clip planes are setup as planes perpendicular to the eye reference Z (from screen toward user's eye) axis.

- **"Xm3DFClip dz"**
  Purpose: Delta of positionning of the Front Clip plane.

- **"Xm3DBDCue dz"**
  Purpose: Delta of positionning of the Back Depth Cueing plane. The Depth Cueing planes are setup as planes perpendicular to the eye reference Z (from screen toward user's eye) axis.

- **"Xm3DFDCue dz"**
  Purpose: Delta of positionning of the Front Depth Cueing plane.

- **"Xm3DEyeLookAt xe ye ze xa ya za"**
  xe ye ze: the world reference 3D coordinates of the eye.
  xa ya za: the world reference 3D coordinates of the point at which the eye looks.

- **"Xm3DSetFocal fov"**
  (Around 100 should be good).

- **"Xm3DSetViewCenter fileId ResNum AtName <chain>"**
  return: none
  Purpose: This will center the view on one atom.
  (WARNING: Resnum is currently in the file numbering, which might be different from a simple count from 0!!!). If no chain descriptor is precised, a blank chain is considered.

- **"Xm3DInqBC"**
  return: 3 double values, ie the coordinates of the view transformation center. (World coordinates).

- **"Xm3DInqRM"**
  return: a 4 x 4 DtFloat (double) matrix, the current rotation matrix R.

- **"Xm3DInqTM"**
  return: a 4 x 4 DtFloat (double) matrix, the current translation matrix T.

- **"Xm3DInqSM"**
  return: a 4 x 4 DtFloat (double) matrix, the current scaling matrix S.

- **"Xm3DInqWEM"**
  return: a 4 x 4 DtFloat (double) matrix, the current World to Eye reference transformation matrix WE.

- **"Xm3DInqEWM"**
  return: a 4 x 4 DtFloat (double) matrix, the current Eye to World reference transformation matrix EW.

- **"Xm3DInqGWEM"**
  return: a 4 x 4 DtFloat (double) matrix, the current GWE global World- ¿Eye transformation matrix. This matrix is equivalent to CGtoE x WE.

- **"Xm3DInqEtoCGM"**
  return: a 4 x 4 DtFloat (double) matrix, the current global Eye to transformation center matrix EtoCG.

- **"Xm3DInqCGtoEM"**
  return: a 4 x 4 DtFloat (double) matrix, the current transformation center to Eye matrix CGtoE.

- **"Xm3DInqGVM"**
  return: a 4 x 4 DtFloat (double) matrix, the current global viewing transformation matrix. This matrix is such as:
  X' = (EtoCG x T x R x S x GWE) X
  where X is the world coordinates of atoms, X' are the viewing coordinates of atoms. X' projected on XY plane are drawn by XmMol's renderer.

4.12 Selecting the atoms displayed

A series of commands to select the visible atoms. Frequent arguments (not detailed again below) are:
fileId: the XmMol id of the file. You can either directly read it from the file summary box of XmMol interface, or get it as the returned value from XmIOLoadFile, XmIONewFile or XmInqIdOfNthFile.

return: none.

REMARK: The commands of the XmIO section XmIOPipeCharTopicRange<In,Out> also allow a direct access to the DrwAtm field.

- **"XmUnactivateFile fileId"**
  return: none
  Purpose: The same as to toggle off the 'visible' button of the correponding file. When this has been performed, the corresponding file does not respond to any request. In particular, performing show mask commands will not be taken into account.

- **"XmActivateFile fileId"**
  return: none
  Purpose: The same as to toggle on the 'visible' button of the correponding file.

- **"XmHideRange fileId atomFrom atomTo"**
  atomFrom: the number of the starting atom (from 0).
  atomTo: the number of the ending atom (included)(from 0).
  return: none
  Purpose: To hide a series of atoms of the file.

- **"XmHideFile fileId"**
  return: none
  Purpose: A shortcut for XmHideRange fileId 0 nAtomsOfFile-1.

- **"XmShowRange fileId atomFrom atomTo"**
  atomFrom: the number of the starting atom (from 0).
  atomTo: the number of the ending atom (included)(from 0).
  return: none
  Purpose: To Show a series of atoms of the file. SEE ALSO XmIOPipeCharTopicRangeOut AND XmIOPipeCharTopicRangeIn TOPICS.

- **"XmShowFile fileId"**
  return: none
  Purpose: A shortcut for XmShowRange fileId 0 nAtomsOfFile-1.

- **"XmHideRangeH fileId atomFrom atomTo"**
  atomFrom: the number of the starting atom (from 0).
  atomTo: the number of the ending atom (included)(from 0).
  return: none
  Purpose: To hide hydrogens encountered between the atoms from and to.

- **"XmShowRangeH fileId atomFrom atomTo"**
  atomFrom: the number of the starting atom (from 0).
  atomTo: the number of the ending atom (included)(from 0).
  return: none
  Purpose: To show hydrogens encountered between the atoms from and to.

- **"XmHideRangeBB fileId atomFrom atomTo"**
  atomFrom: the number of the starting atom (from 0).
  atomTo: the number of the ending atom (included)(from 0).
  return: none
  Purpose: To hide backbone atoms encountered between the atoms from and to.

- **"XmShowRangeBB fileId atomFrom atomTo"**
  atomFrom: the number of the starting atom (from 0).
  atomTo: the number of the ending atom (included)(from 0).
  return: none
  Purpose: To show backbone atoms encountered between the atoms from and to.

- **"XmHideRangeSC fileId atomFrom atomTo"**
  atomFrom: the number of the starting atom (from 0).
  atomTo: the number of the ending atom (included)(from 0).
  return: none
  Purpose: To hide non backbone atoms encountered between the atoms from and to.

- **"XmShowRangeSC fileId atomFrom atomTo"**
  atomFrom: the number of the starting atom (from 0).
  atomTo: the number of the ending atom (included)(from 0).
  return: none
  Purpose: To show non backbone atoms encountered between the atoms from and to.

- **"XmSetShowHalfs"**
  return: none
  Purpose: To make half bond drawn from visible to unvisible atom.

- **"XmUnsetShowHalfs"**
  return: none
  Purpose: To make half bond not drawn from visible to unvisible atom (the default).

4.13 Selecting the colouring patterns

- **"XmColorSetBWMode"**
  return: none
  Purpose: To install a black and white display mode.

- **"XmColorSetColorMode"**
  return: none
  Purpose: To install a color display mode.

- **"XmGetUsedColorNum"**
  return: one integer.
  Purpose: To get the number of colors possibly applied to one atom. This may be different from XmGetInstalledColorNum since there exists dashing in Black & white for example.

- **"XmGetInstalledColorNum"**
  return: one integer.
  Purpose: To get the true number of colors currently in use by XmMol. For Black & White, only 1 is returned for example.

- **"XmColorGetNames"**
  return: n (accessed by XmGetInstalledColorNumber) strings: the names of entries of RGB.txt.
  Purpose: To get the names of the colors currently in use by XmMol (according to the X11/RGB.txt file). For Black & White, only "White" will be returned.

- **"XmColorSetNames"**
  return: none
  Purpose: To set the names of the colors to be used by XmMol (according to the X11/RGB.txt file). For Black & White, you must send 1 name, for compatibility. You must send n (accessed by XmGetInstalledColorNumber) strings to XmMol (the names of entries of RGB.txt). The colors will be installed automatically. YOU *MUST* SEND THE EXACT (XmGetInstalledColorNumber) NUMBER OF COLOR NAMES.

- **"XmColorSetOneName colId RGBName"**
  return: none
  Purpose: To set the name of one color to be used by XmMol (according to the X11/RGB.txt file). If the colId is less than 0 or more or equal than "XmGetInstalledColorNum", nothing will happen. The RGBName *MUST* be one that your XServer knows about. Have a look to the X11/RGB.txt file.

- **"XmSetAtmRangeColor fileId from to colId"**
  fileId: name of file concerned. from, tto: the number of delimiting
  atoms: (from 0)(included).
  colId: the number of the color to be applied, given as the index (from 0) of the name of the color as accessed by XmGetColorNames or previously setup by XmSetColorNames. By default, 0 corresponds to white. If the color is more than XmGetUsedColorNumber or less than 0, nothing is done. SEE ALSO XmIOPipeCharTopicRangeOut AND XmIOPipeCharTopicRangeIn TOPICS.
  return: none

- **"XmSetResRangeColor fileId from to colId"**
  fileId: name of file concerned. from, tto: the number of delimiting residues (from 0)(included).
  colId: the number of the color to be applied, given as the index (from 0) of the name of the color as accessed by XmGetColorNames or previously setup by XmSetColorNames. By default, 0 corresponds to white. If the color is more than XmGetUsedColorNumber or less than 0, nothing is done. SEE ALSO XmIOPipeCharTopicRangeOut AND XmIOPipeCharTopicRangeIn TOPICS.
  return: none

- **"XmColorBy colorScheme"**
  colorScheme: one of HalfBond, File, Chain, Segment, Residue, BB/SC, PropertyVal, PropertyClass.
  example: XmMolSendCMsg("XmColorBy BB/SC");
  return: none
  Purpose:Exactly the same as to use the color menu.

4.14 Labels

A series of functions relative to labels. **return**: None.

- **"XmLblLongLabels"**
- **"XmLblShortLabels"**
- **"XmLblClearLabels"**
- **"XmLblHideLabels"**
- **"XmLblShowLabels"**
- **"XmLblClearDistances"**
- **"XmLblHideDistances"**
- **"XmLblShowDistances"**
- **"XmLblAtmLabel fileId ResNum AtName <chain>"**
  Purpose: To setup a label for atom of name AtName of the file fileId and of residue number resNum (file numbering) and of chain 'Chain'. If no chain is specified, a blank chain is considered as the default.
- **"XmLblSetResLabels"**
  for all files
- **"XmLblClearResLabels"**
  for all files
- **"XmLblFileResLabels fileId"**
  for the file 'fileId'
- **"XmLblClearFileResLabels fileId"**
  for the file 'fileId'
- **"XmLblDstLabel <fileId ResNum AtName <chain> >1,2"**
  Purpose: Display distance between 2 atoms determined for each by fileId, residue number (file numbering) and atom name and chain. If no chain is specified, a blank chain is considered by default.

- **"XmLblLnkLabel <fileId ResNum AtName <chain> >1,2"**
  Purpose: Draw a line between 2 atoms determined for each by fileId, residue number (file numbering) and atom name and chain. If no chain is specified, a blank chain is considered by default.

- **"XmLblHideHBonds"**
  Purpose: This command concerns H bonds that are detected for the XmMol's formatted files that include explicit hydrogen locations. (refer to the user's guide corresponding section).

- **"XmLblShowHBonds"**

4.15 Accessing the bonding matrix

- **"XmBondScratchOn"**
  Purpose: Enables scratchBonding, based on atom names and distances (for PDB files only). This will be performed automatically at the load of the files.

- **"XmBondScratchOff"**
  Purpose: Disables scratchBonding.

- **"XmBondSqntlConnect fileId"**
  fileId: the XmMol Id of the concerned file. Might have been obtained by "XmInqNameOfNthFile id" for example.
  Purpose: Recompute a connectivity matrix based atom sequentiallity. Consecutive atoms are bonded, except if their chain Id is different. SHOULD ALWAYS BE FOLLOWED BY A CALL TO "XmBondRestartTables fileId".

- **"XmBondScratchConnect fileId"**
  fileId: the XmMol Id of the concerned file. Might have been obtained by "XmInqNameOfNthFile id" for example.
  Purpose: Recompute a connectivity matrix based upon atom names and distances. SHOULD ALWAYS BE FOLLOWED BY A CALL TO "XmBondRestartTables fileId".

- **"XmBondRestartTables fileId"**
  fileId: the XmMol Id of the concerned file. Might have been obtained by "XmInqNameOfNthFile id" for example.
  Purpose: Restart the file analysis after its connectvity matrix has been modified. SHOULD ALWAYS BE CALLED AFTER ANY MODIFICATION OF THE CONNECTIVITY MATRIX DIRECTLY. XmBondxxxConnect functions call it automatically. This is also automatically performed if XmBondScratch is On when PDB files are loaded. Mac files a priori already have it done.

4.16 Docking

This section describescommands that allow some docking in the way XmMol proposes it interactively. Please refer to the User's manual to understand the way they should be used.
return: None.

- **"XmDockStart"**
  Purpose: Starts docking display mode for XmMol
- **"XmDockEnd"**
  Purpose: End docking display mode for XmMol.
  WARNING: If you did not use the XmDockKeep command, the coordinates are unchanged. This should always be called to end a docking process.
- **"XmDockSave"**
  Purpose: Transform the coordinates of the docked region permanently (not reversible).
- **"XmDockForget"**
  Purpose: Re-initialize the docking matrix to identity. Coordinates are identical to those after the last XmDockKeep command.
- **"XmDockAddFileFrg fileId chnId"**
  fileId: the XmMol Id of the concerned file. Might have been obtained by "XmInqNameOfNthFile id" for example.
  chnId: the Id of the chain that will be added to the list of docked regions.
  Purpose: To add a chain to the docking list. Refer to the commands "XmInqFileChnNum fileId" and "XmFileInqChnId" to get informations about chains of XmMol, and the way they can be managed by a delegate.
- **"XmDockDelFileFrg fileId chnId"**
  fileId: the XmMol Id of the concerned file. Might have been obtained by "XmInqNameOfNthFile id" for example.
  chnId: the Id of the chain that will be removed to the list of docked regions.
  Purpose: To remove a chain to the docking list. Refer to the commands "XmInqFileChnNum fileId" and "XmFileInqChnId" to get informations about chains of XmMol, and the way they can be managed by a delegate.
- **"XmDockAddFileFrg fileId frgNum"**
  fileId: the XmMol Id of the concerned file. Might have been obtained by "XmInqNameOfNthFile id" for example.
  frgNum: the number (from 0) of the fragment that will be added to the list of docked regions.
  Purpose: The basic unit for docking is the segment. see the commands "XmInqFileSegNum fileId" and "XmFileInqFrgId" to get informations about segments of XmMol, and the way they can be managed by a delegate.
- **"XmDockDelFileFrg fileId segNum"**
  fileId: the XmMol Id of the concerned file. Might have been obtained by "XmInqNameOfNthFile id" for example.
  frgNum: the number (from 0) of the fragment that will be removed to the list of docked regions.
  Purpose: The basic unit for docking is the fragment. see the commands "XmInqFileFrgNum fileId" and "XmFileInqFrgId" to get informations about fragments of XmMol, and the way they can be managed by a delegate.

- **"XmDockAddFile fileId"**
  fileId: the XmMol Id of the concerned file. Might have been obtained by "XmInqName-OfNthFile id" for example.
  Purpose: A shortcut to avoid iterative calls to XmDockAddFileSeg.

- **"XmDockDelFile fileId"**
  fileId: the XmMol Id of the concerned file. Might have been obtained by "XmInqName-OfNthFile id".
  Purpose: A shortcut to avoid iterative calls to XmDockDelFileSeg.

- **"XmDockTranslate x y z"**
  x, y, z: the value of the translation performed.

- **"XmDockRotate x y z angle"**
  x, y, z: the value of the direction of rotation. The center of transformation is the barycenter of the moving atoms, except if you used the XmDockCenter command.
  angle: value (radians) of the angle of rotation around X.

- **"XmDockCenter x y z"**
  x, y, z: the value of the center of transformation for docking rotations. **Purpose**: WARNING: Normal use of docking function is to perform rotations around the barycenter of the selected atoms. This is automatically inited when selecting moving atoms. However, you are left the opportunity to make your own transformations BUT, doing this, you should remember that your value will be overwritten each time that you will change the selection of moving atoms. As well, you will have to manage the consequences of each change of the center on the concatenation of the elementary moves. The matrices are concatenated as P' = Tr x Rt x P where P is a vertical homogeneous 4 coordinates vector, Rt a 4x4 homogeneous coordinate rotation matrix, Tr a 4x4 homogeneous coordinate translation matrix. Each elementary rotation is preconcatenated to the current rotation matrix Rt.

- **"XmDockSetRotateMatrix"**
  MUST be followed by a call to XmMolSendData passing a 4x4 rotation matrix.
  Purpose: A direct way to pass a rotation matrix used for docking. This matrix replaces the current docking rotation matrix. It is not concatenated to the previous docking rotation matrix.

- **"XmDockSetMatrix"**
  MUST be followed by a call to XmMolSendData passing a 4x4 transformation matrix.
  Purpose: A direct way to install a docking matrix (translation + rotation). This matrix replaces the current docking matrix. It is not concatenated to the previous transformation matrix.

4.17 Using shared memory

The sharing memory facilites supported by XmMol (2.8) allow to share an XImage data, and some fields of each file loaded by XmMol. Better if you get some understanding about System

V IPC's BEFORE trying to use them. On dirty exit of XmMol or delegates, some shared memory segments may remain. Think of the ipcs and ipcrm shell level commands to check if everything is OK.

- **"XmShmXImageGet width height depth format offset padding bytes_per_line"**
  See the XCreateImage documentation for arguments. Depth should be always 8 since Pseudo-Color, and format should be ZPixmap. returns: the ipc key of the image data space allocated by XmMol. XmMol will allocate the data necessary for, and create an XImage. The data is allocated using shmget and attached to XmMol. the size of the data is :
  ImgWidth*ImgHeight*sizeof(unsigned char) + 32; // if ImgDepth ¡= 8
  ImgWidth*ImgHeight*sizeof(int) + 32; // if ImgDepth ¿ 8
  If an image data was already attached, it is first detached. If it was allocated by XmMol, is is also destroyed, so that successive call will not allow shared memory segments not used. USING THIS FUNCTION ALSO ENSURES THAT YOU WILL NOT HAVE SOME SHARED MEMORY SEGMENTS LEFT ON XmMol EXIT. TO THIS RESPECT, IT IS BETTER TO USE IT THAN XmShmXImageAt, IF YOU ARE NOT FAMILIAR WITH IPC's. Remember than you can check the currently shared memory segments remaining with the shell command ipcs, and remove them with ipcrm -m Id.

- **"XmShmXImageAt IPCId width height depth format offset padding bytes_per_line"**
  See the XCreateImage documentation for arguments other than IPCId.

  IPCId: the IPC key to attach the image data to. If an image data was already attached, it is first detached. **return**: none XmMol will attach the data which IPC key is IPCId, and create an XImage using this data.

- **"XmShmXImageDt"**
  return: none XmMol will detach the data associated to a previously created XImage, and destroy the XImage. Should only be called after XmShmXImageAt has been called once.

- **"XmShmFileAccessGet fileId"**
  NOT IMPLEMENTED UNTIL VERSION 2.8 **return**: keys for all fields of a DtXmMolShmRec, that gives a direct access to some of the fields of the File Rec of the file of name "fileId". To be shure to get all fields in a proper order, use the built-in function: XmMolCGetFileShmRec(DtXmMolShmRec *M). When you finish needing M, DO NOT USE free() but use shmdt() for each pointer of M. BETTER, use XmMolCDtFileShmRec(DtXmMolShmRec *M). IF YOU FORGET THIS, YOU MAY HAVE AN INCREASING NUMBER OF SHARED MEMORY SEGMENTS NOT RELEASED ON XmMol EXIT.

  Example:

  DtXmMolShmRec M;
  int nRes, i;

  /* /usr/local/toto being loaded by XmMol */

```
XmMolSendCMsg("XmShmFileAccessGet /usr/local/toto");
XmMolCGetFileShmRec(&M);
XmMolSendCMsg("XmInqFileResNum /usr/local/toto");
XmMolGetIntData(&nRes);
for (i=0;i¡nRes;++i)
printf("Atom %d coordinates: %.2lf %.2lf %.2lf0,
M-¿corm[i][0],M-¿corm[i][2],M-¿corm[i][2]);
```

4.18 External renderers

External renderers can be implemented as delegates of XmMol. The basic idea is that external renderers can either be automatically called by XmMol on each screen refresh (In such case, all facilities are given to the delegate to ask XmMol why it wants to refresh), but delegates could also simply use XmMol as an ouptut, keeping their own refresh control. In all cases, the drawing will request a shared XImage data (see the XmShmXImage¡At,Dt,Get¿ functions). The renderer fills the image and XmMol flushes it onto the screen. The transfer of all rendering computation ot a delegate supposes 2 phases:

(1) install the function that will be considered as the new rendered. This is made by implementing callback functions to trigger the renderer:
XmMolSetCallbackFunc(void (*func)(), char *data);

(2) activate them. This is made by a call to:
sprintf(buff,"XmAddUserCmd DcCBRO refresh any %%%d DcCBRO%%",pid);
or
sprintf(buff,"XmAddUserCmd DcCBM2P any %d %%%d DcCBM2P%%",XmMolId,pid);
XmMolSendCMsg(buff);
XmMolSenCMsg("XmRndrInqCntrl"); is obsolete.

- **"XmRndrInqCntrl"**
  is obsolete in version 3.1

- **"XmRndrInqWhy"**
  obsolete in version 3.1

- **"XmRndrInqMouse"**
  Obsolete in version 3.1. Use XmInqMousePos instead. This call will return the x and y position of the mouse as well as its state (as in X11).

An example of how a "rendering" function should be organized:

```
void XmMolRefreshScreen(char *reason, char *data)
{
  extern int MainWide, MainHigh;
```

```
int why, mouseX, mouseY, mouseState, what, sze;
int isze = sizeof(int);
double whatMods[3];
static int tran[3] = {0,0,0};
static int curX = 0, curY = 0;
int OK, rval;
int cWdth, cHght, cDpth;

fprintf(stderr,"Delegate ¡- XmMol to refresh for reason: %s\n",reason);

if (!strcmp(reason, "DcCBM2P")) {
  fprintf(stderr,"RasMol called on DcCBM2PL\n");
  XmMolSendCMsg("XmInqMousePos");
  XmMolGetData((char *)&mouseX, &isze);
  XmMolGetData((char *)&mouseY, &isze);
  XmMolGetData((char *)&mouseState, &isze);
  curX = mouseX;
  curY = mouseY;
  do {
    XmMolSendCMsg("XmInqMousePos");
    XmMolGetData((char *)&mouseX, &isze);
    XmMolGetData((char *)&mouseY, &isze);
    XmMolGetData((char *)&mouseState, &isze);

    fprintf(stderr,"RasMol mouse refresh: posX %d posY %d buttonMasks %d\n",mouseX,mouseY,mouseState);
    if (!mouseState) break;
    OK = 0;
    rval = mouseY - curY;
    rval = 9 * rval / 20;
    if (rval) {
      sprintf(CurLine,"ROTATE X %d",rval);
      ExecuteRotationCommandAndRefresh();
      OK = 1;
    }
    rval = mouseX - curX;
    rval = 9 * rval / 20;
    if (rval) {
      sprintf(CurLine,"ROTATE Y %d",rval);
      ExecuteTranslationCommandAndRefresh();
      OK = 1;
    }
    if (OK) RefreshScreen();
    else {
      fprintf(stderr," Delegate refresh: Dummy refresh\n");
```

```
      }
      curX = mouseX;
      curY = mouseY;
    } while (mouseState);
  } else if (!strcmp(reason, "DcCBRO")) {
    sprintf(CurLine,"TRANSLATE X 0");
    ExecuteTranslationCommandAndRefresh();
    fprintf(stderr,"Delegate simple refresh()\n");
    RefreshScreen();
  }
}
```

Such a function could be implemented in a program using the following way:

```
void main()
{
  int XmMolId = -1;
  int sze;
  size_t IPCId;
  unsigned short *ptr = NULL;
  char XmMolCmd[BUFSIZ];

  XmMolId = XmMolConnect(); /* Open connection to XmMol */

  /* Ask XmMol to allocate a shared image data 100x100 (8 planes) */
  sprintf(XmMolCmd,"XmShmXImageGet %d %d %d %d %d %d %d",
            100, 100, 8, ZPixmap, 0, 8, 0);
  XmMolSendCMsg(XmMolCmd);
  sze = sizeof(key_t);
  XmMolGetData((char *) &IPCId, &sze); /* Get the IPC key to the memory. */
  imagePtr = shmat(IPCId, 0, 0); /* Attach image space to the process.*/

  XmMolSetCallbackFunc(XmMolRefreshScreen,NULL);
  /* renderer MUST fill the image data pointed by imagePtr and call XmRefresh.*/

  sprintf(buff,"XmAddUserCmd DcCBM2P any %d %%%d DcCBM2P%%",XmMolId,pid);
  XmMolSendCMsg(buff);


  XmMolDisconnect(); /* From here, XmMol's renderer works again.*/
}
```