Proposals memo; by B. Parsley

This document attempts to do several things. There is a proposed conceptual
structure for the Tools Group programs. There is a lot of proposing of names for
things. There is a section that proposes solutions for several of the major problem
**s
that we have encountered. I also go through some of the structure piece by piece
and list all the defficiences I have been able to think of. This is an attempt at a
**n
exhaustive list of defficiencies. For many of the pieces I propose some changes to
the current interfaces and implementations.

It should be noted that most of the proposed changes are not due to real
defficiences, but are usually ideas that I or others have had about better ways to
do things. None of the proposed changes should be taken as inferring that
anybody has been guilty of bad design work. A lot of the ideas came from using
the existing software and then thinking of ways it might be changed to make it
easier or smoother to use. There are some proposed deletions of some code that
experience has indicated would rarely be used. There are a bunch of minor
"cleanup" things suggested.

It should also be noted that everything in this document is intended as a proposal
-- to be discussed and debated by the group prior to approval or rejection. I have
not often included some phrase like "I propose" or "I suggest" in the document. I
hope everyone will take all of the ideas stated in here as if there were such a
phrase somewhere in the discussion of the idea.

I have not had time to put all this material into a nice, formal memo form. Please
bear with all the rough edges that are in here. I'll try to smooth it out later. I
**
have not had time to write down all my reasons and justifications for the proposed
changes. I would like to present those verbally.

I suggest that we all spend Wednesday afternoon doing something like the
following: Go through the document and decide which things we will do and
which we won't. For those things that we decide to do, decide when, e.g., for the
Preview Release, as soon as possible after that, "eventually". For those things tha
**t
are to be done soon, we should decide who is to do them.

John Weaver proposes the term "Development Environment" to mean all the software listed below (in large part written by the Tools Group with some major help from others on things like the guts of the Compiler Server; several of the tools will probably be written by others also).

The Development Environment is made up of three pieces: the "Tools Environment", a collection of "tools", and several "servers".

The Tools Environment is mode up of 6 pieces: the module TeInit that constructs the Tools Environment (an image file), the configuration that is the "Tools EXecutive", and 4 configurations that are "packages" of subroutines or "interfaces".
**
The four interfaces are: the User Interface, the Librarian Interface, the Fifo Interface, and the Pup Package (including FTP).

Here is the hierarchical structure of the Development Environment (with abbreviations):

```
  Tools Environment (TE or Te)

    TeInit

    Tools Executive (TEX or Tex)   -- TexDefs

      TexStimLev
      TexProcLev
      TexBackLev
      TexQueue
      TexManipTools
      TexTool

    User Interface (UI or Ui)   -- UiDefs

      UiDisplay
      UiFonts
      UiWindowBasics and UiWindowContents
      UiMenus and UiMenuCS
      UiSelections
      UiCursors
      UiUAS
      UiMisc
      UiContexts

    Librarian Interface (LibI or Libi)   -- LibiDefs

      LibiObjects
      LibiPropLists

    Fifo Interface (FifoI or Fifoi)   -- FifoiDefs

      ???

    Pup Package   -- PupDefs
```

Servers

- Librarian Server
- Fifo Server
- File Store
- Compiler Server
- Xref Server
- Info Server
- Printer Server(s) -- EARS, 3100, etc.
- Source Formatter Server

Tools

- Local Librarian
- Examine
- TypeScript
- Program Editor
- General Editor
- Compiler
- Spooler
- Message
- Chat
- Debugger(s)
- Configuror
- Dynamic Program Analyzer
- Cursor Editor
- various project management tools
- various testing tools

The following things should be noted about the pieces and names above:

The Tools Environment is basically all the code that lives in one of the Tools.image files (but doesn't include the Mesa Runtime System).

The module TeInit has no corresponding Defs file. The files TexDefs and UiDefs have no implementors. The module TexTool includes the Window Manager. The modules UiWindowBasics and UiWindowContents both implement the Defs file UiWindowDefs. The modules UiMenus and UiMenuCS (Menu Command Select) may get combined into just one UiMenus. The modules LibiObjects and LibiPropLists implement LibiDefs and have no other corresponding Defs files. The same situation will probably hold true for the Fifo Interface module(s). There are a whole lot of modules that implement PupDefs. All the other modules in the TE have exactly one Defs file that they implement. The Pup Package was not written by the Tools Group.

I'm not greatly attached to the particular spellings of the prefixes Tex, Ui, Libi **,
and Fifoi, but I do think it very important that each of the files that make up a configuration or interface all start with the same short prefix. I also think that lumping all the modules listed above into the User Interface is a good idea.

I'm not greatly attached to any of the server names. The File Store will not be done by the Tools Group. The Librarian Server, Fifo Server, and the File Store will each have exactly one dedicated machine (but we may want to run both the Librarian and Fifo Servers in the same machine). There may be zero or several instances of the other servers at any particular time. The compiler part of the Compiler Server will not be written by the Tools Group. The Info Server is my name for something that would perform functions like the current Lister and/or Simonyi's CR programs (or maybe this should all be combined into a more general Xref Server). The Source Formatter is the "Procedures and Standards" name for the program that makes Mesa source code files conform to the standards. The above list of servers is probably not exhaustive.

I'm not greatly attached to any of the tool names. There may be no need to have all of the Examine, TypeScript, Program Editor, and General Editor Tools as separate tools since they are so similar. The General Editor (probably Diamond) may never be implemented as a tool. The Compiler Tool should be able to do a single, local compilation and do a "consistent" compilation of some group of modules either locally or via a Compiler Server. The Spooler Tool is my idea for one tool that could set things up for any of the Xref, Info, Printer, or Source Formatter Servers (or any other "data processing" servers there may be). The Message Tool will probably be mostly written by Metcalf's group. The Chat tool may be written by Schwartz. The Debugger Tool(s) assumes the existence of a "Core Debugger". The Configuror assumes the existence of "configurations". The various project management and testing tools will probably not be written by the Tools Group.

I think it is very important for us (the Tools Group) to agree on some such conceptual structure as the above and to agree on the names of things. I think this would be a great aid in our thinking, our communications with each other, and in our documentation. I think an agreement on (what I have called) the Tools Environment's structure and names should be settled on very soon. I think we can leave the names and even the existence some of the tools and servers a little vague for a while longer.

Here are some proposed solutions to problems that currently exist:

How does a PNR or MCR retain control while a PBK is held down?

I propose the following routine that will live in TexProcLev:

```
  IsPbkStillDown: PROCEDURE [TexDefs.PBK] RETURNS [BOOLEAN];
```

Each time the routine is called, it will go through the User Action Queue, looking for the appropriate down PBK item. It will throw away all items (including the looked-for item) until the queue is empty or the item is found (this merely means that no PNRs will be called). The returned BOOLEAN tells (the converse of) whether the item was found. Each call will do the right thing so that subsequent calls on GetProcLevCursorPosition will return the right thing -- that's the hardware coordinates of the cursor if the PBK is still down, or the coordinates in the looked-for queue item if the PBK just went up. In both cases the coordinates returned are a BitmapPlace and have been corrected for the hot spot.

What to do about subwindow boundary crossings with PBKs held down?

I propose the following routine that will live in TexProcLev:

```
  EnumerateDownPBKs: PROCEDURE [UiDefs.PNRsHandle, TexDefs.UpDown];
```

The idea is that cursor PNRs could call this routine if they wanted to warn their tool about down PBKs when the subwindow was being entered or exited. This routine would go through the Processing Level state (NOT the Stimulus Level state) looking for PBKs that were down and call the appropriate PNR if it found one. Note that the PNRs will always be called with the UpDown specified above. This is so, e.g., the cursor PNR can fool its PNRs into thinking
**
the PBKs went down on entry into the subwindow but went up on exit. Note also that the cursor PNR may just want to use the standard PNRs of its subwindow, or it may use another set of PNRs for this sort of thing (and possibly different PNRs for entering and exiting). Note also that if the subwindow isn't interested in some of the PBKs, it is free to use UiMiscDefs.NopPbkPNR in any of the fields of the PNRsObject.

What to do about the cursor "hot spot" problem?

I think the thing to do is to change the cursor package so that any time a StoreCursor or SwapCursor is done, the hardware cursor and mouse coordinates get changed according to the relative differences of the old and new hot spots. Note that interrupts should be turned off when changing the hardware coordinates.

The cursor may jump a little bit, but users should be told that the hot spot will track smoothly. Note that this scheme follows our convention that the user must predict what the situation will be if he/she types ahead. I think it's actually easy for the user. I think he/she need merely always point at things with the hot spot of whatever cursor is currently on the screen.

About image files:

I think that there are at least 8 different image files that various people will want to have available. (Below, "most Mesa" currently means all Mesa except Keystreams.)

```
    Mesa.image

      all Mesa code segments
      all Mesa symbol segments

    PupBare.image

      all Mesa and Pup Package code segments
      no symbols

    PupPared.image

      all Mesa and Pup Package code segments
      all Mesa symbols, but only Coolie's symbols from the Pup Package

    PupAll.image

      all Mesa and Pup Package code segments
      all Mesa and Pup Package symbol segments

    TeBare.image

      most Mesa, all Pup Package, and all TE code segments
      no symbols

    TePared.image

      most Mesa, all Pup Package, and all TE code segments
      most Mesa, but only Coolie's from the Pup Package, and no TE symbols

    TeFull.image

      most Mesa, all Pup Package, and all TE code segments
      most Mesa, but only Coolie's from the Pup Package, and all TE symbols

    TeAll.image

      most Mesa, all Pup Package, and all TE code segments
      most Mesa, all Pup Package, and all TE symbol segments
```

I propose to write the small amount of code required to produce PupPared.image. I also propose that someone (probably me) generate each of the files listed above. We should keep as many of them as will fit in <Tools> or on IFS. The others may be kept on a disk pack somewhere.

About selections (and in particular how to get more than one tool named TestTool instantiated):

It should be noted that we do NOT currently have a very useful selection mechanism.

I propose a (possibly temporary) solution: Somebody should write a TypeScript Tool. TEX would "define" that tool at startup time. Using a UAS, the TypeScript Tool would allow type-in. New type-in up to a CR or ESC would become the current selection. The tool would also (eventually) have selection button PNRs. This seems the quickest way to get type-in to be selected for use as parameters to commands (at least to the Instantiate-tool command).

Note that the tool would be made up almost exclusively of routines that I propose for inclusion in UiMisc and UiSelection. This tool could serve as a prototype or testbed for those routines before we actually include them in the User Interface.

There may be some problems about bitmaps and turning the display on and off before and after calls on MakeImage.

  I talked to Wick about this a little. Smokey, the three of us should get together
**
  and make sure we do the right thing

Here's an idea for which I can find no better place in this document:

  It would be nice if the Fifo Server could glom onto any Alto on the Ethernet
  that was running DMT and set up a server there. Boggs and Taft know how
  to do this. Note that such a server would have to do all of its file I/O over the
**
  Ethernet. (I'm not sure I'm the originator of this idea.)

I would like to take this opportunity to point out another of our unsolved
problems, but I have no solutions to propose:

  What, if anything, is the Librarian Server going to do about automatically
  compiling, formatting, printing, xrefing, etc. files as they are checked in?

Using my proposed hierarchical structure of the Tools Environment, here are my (intermixed) lists of deficiencies and proposed changes. At various points you will find listings of some of the Defs files for the User Interface. They should help to indicate or explain some of the things I have in mind.

Some of the proposals below have the "keyword" "done" attached to them. That means they've already been done. Some have the keyword "eventually". That means I suggest we wait a (long) while before doing them. Some have the keyword "sometime". That means I suggest we don't try to do them for the Preview Release. I suggest that we try to incorporate all the other proposals (that
**
get accepted) in the Preview Release.

First I would like to talk a bit about the organization of the User Interface definitions files. I am unable to decide which of several schemes I think is the better. I'd like to mull this over with you all. Here are the schemes:

The scheme proposed in this document which basically just breaks the former ToolWindowDefs into UiDefs (types and constants only) and UiWindowDefs (the procedures).

Leave ToolWindowDefs as it is (but rename it UiWindowDefs).

Move the Places, Boxes, etc. definitions to TexDefs.

Treat PNRsObjects as contexts. Note that then the "pnrs" field of SubwindowObjects would be deleted. Then we could move all the PNRs definitions (including the {Cursor/Pbk}PnrTypes) into their own Defs file. We would then have to provide routines like {I/Uni}nstantiatePNRs{With/From}{W/Subw}indow. Below I have proposed that CreateSubwindow take a PNRsHandle parameter. That would have to change if this idea is accepted.

We could make SubwindowObject.contextChain be PRIVATE and of type POINTER. Then we could move all the context definitions into UiContextDefs. Note that no one except the Contexts package is ever supposed to touch this field, so it might not be very dangerous.

TeInit

Change name of file from TexInit to TeInit.

Move UnNew to TexManipTools.

Delete TexInitDefs.

Move code in LibjectLoader to this file.

Load UiDisplay and UiFonts.

Do the right thing about getting the display initialized.

Load the Fifo Interface.

Get new Pup Package.

Eventually, move some of the initialization code to TexProcLev so that this module and its global frame may be released before the MakeImage.

Eventually, pare Mesa runtime further.

Tools Executive (TEX or Tex)   -- TexDefs

TexDefs

No proposed changes.

TexStimLev

Put in the new way of getting to the debugger via a chord. This will (usually) leave the debugger pointed at the right frame of the right process.

Maybe, have the "wiggling bit" problem taken care of by the Stimulus rather than the Processing Level.

Sometime, add abort chord(s).

Sometime, add "clear User Action Queue" chord.

TexProcLev

Query the user as to where the Librarian and Fifo Servers and the File Store are. If they are to be on the local machine, then some NBSs must be done. In any case, the information has to be conveyed to the appropropiate parties.

Change so that user actions occuring while the cursor is in a window's frame go to TexTool (assuming that the frame is not in any of the subwindows of that window).

Do an UnNew on TeInit.

Add IsPbkStillDown (see the discussion about the problem of PNRs retaining control while a PBK is held down).

The Window Manager is going to need an analagous routine to help with the continuous Move commands before the "click" (see below).

Add EnumerateDownPBKs (see the discussion about the problem of subwindow boundary crossings while PBKs are held down).

Eventually, redo the PNR mechanism using coroutines.

TexBackLev

Rename TexBackground{Defs} to be TexBackLev{Defs}.

Eventually, redo background mechanism using coroutines.

TexQueue

No changes proposed.

TexManipTools

Move UnNew procedure from TexInit to here.

Add uas field to ToolInstanceObject.   -- see UiUAS discussion

Add warmStart field to ToolInstanceObject. This is a procedure in each tool to be called for "warm starting" tool instances after something like TexTool's Subsys command.

TexMisc

Move the procedures and their definitions to UiMisc. Move some other definitions to TexDefs. Delete TexMisc{Defs}.   -- done

TexTool -- the Window Manager is discussed below

Smokey suggested an idea about making TexWindow be a real window. I'm not sure, but I currently think it's slightly better the way it is now.

Add the following commands: Subsys, Install, Reset, Checkpoint. See my memo "Wish List for a Mesa System" for details and justifications.

Add a command to turn the Monitor on and off.

Eventually, add the Kill command.

Window Manager -- the code will live in TexTool

I propose the following menu commands:

Flip
Move
Grow
Tiny
Normal
Zoom
Unzoom
Permanent-menu-window/Destroy-menu-window (sometime soon)
Cleanup-screen (sometime soon)

About Flip: If the indicated window is on top, put it on the bottom, else put it on top (or there could be separate Bottom and Ontop commands).

About Permanent-menu-window/Destroy-menu-window:

The idea here is to make a "normal" window containing a Ring Menu Subwindow (see below) and leave it displayed on the screen, i.e., not just while the menu button is held down.

There's a potential problem here if the user uninstantiates the tool containing the MCRs without Destroying the Menu Window. Another potential problem is that the Window Manager has no good way of knowing which windows are Menu Windows.

About Cleanup-screen: This is Titleman's idea about automatically straightening out a chaotic screen full of windows (perhaps by making them all tiny and distributing them as evenly as possible over the bitmap).

Most of the commands above need a (corner of a) window as an operand. I propose the following rules for picking the operand.

If the command is invoked from a window's menu ring, use that window's corner that is nearest to the cursor.

If the command is invoked from TexTool's menu ring while the cursor is in some window's name frame, use that window's nearest (upper left or right) corner.

If the command is invoked from TexTool's "window", use the "nearest" window corner.

Note that if the command is invoked from a menu, the button will be UP when the MCR is called. I suggest that the Flip, Tiny, Normal, Unzoom, and Permanent-menu commands turn themselves into Move (continuous) commands after doing their thing. Then the user will terminate those commands and the Move and Grow commands by "clicking" the menu button (or any PBK) which will "deposit" the window and terminate the command.

Another possible way to do this would be to have the user select the command in the normal manner. The the Window Manager would wait for the menu button to go down, throwing away all other user actions. When the button went down, the Window Manager would pick a corner and proceed with the command until the button went up. I think this might be a harder way to do things.

I propose that we allocate the mouse button PNRs of TexWindow as follows:

blue -- normal MenuPbkPnr

yellow -- will do the Grow command, with zooming done via running the cursor to any of the corners of the screen

red -- will put the window on top and then turn into the Move command

User Interface (UI or Ui) -- UiDefs

Note that all file names will have to be changed if my proposed names are accepted.

UiDefs

See the attached listing for what this file might look like. I will discuss the
**
Coords, etc. section immediately below. The Windows and Subwindows sections are discussed in the UiWindow section below. The only change in the PNRs section is the order of the parameters for a PbkPnrType -- this one seems more "natural" to me, it is more inkeeping with the sub-standard that the "major" abstraction be the first parameter. The Contexts section is discussed under UiContexts.

If the CARDINALity of Dimension necessitates ANY loopholes, it should be changed to be an INTEGER.

The type Coords may well be superfluous. I put it in for completeness and symmetry.

The types ScreenPlace and ScreenBox are meant to be used by UiDisplay when dealing with bitmaps.

The Coord and Dimension types are defined following the reasoning of the "sub-standards" that predefined types should be used as little as possible. Also note that if Dimension must be changed to INTEGER, there is only one thing to change rather than several.

I think breaking boxes into places and Dimensions will prove to be a useful thing, maybe primarily for clarity of code. There is also the point that under the old scheme, one couldn't do something like the following:

```
    box.place ← WindowPlaceFromBitmapPlace [...];  -- new scheme
    [box.x, box.y] ← WindowPlaceFromBitmapPlace [...]; -- old scheme
```

Note that there are few comments in UiDefs. I think this makes it more readable and nobody has to worry about errors in the comments. Also, there is now documentation for the stuff here, so the need for comments is greatly lessened and there is the potential problem of keeping two versions of the same thing consistent (the comments and the documentation).

UiDisplay

Remove NovaOps from the directory of the implementor.

Use ScreenPlace and/or ScreenBox from UiDefs.

The Defs file has a several definitions that I would guess are copied from Mesa system Defs files. I'm not sure that's a good idea.

UiFonts

Remove NovaOps from the directory of the implementor.

Remove MopCodes from the directory of the Defs file.

Add the following types: FontHeight CharWidth, StringWidth.

Define the type FontHandle either instead of or as a synonym for FAptr.

See the attached Defs file listing for details about most of the above proposed changes,

The Defs file has a several definitions that I would guess are copied from Mesa system Defs files. I'm not sure that's a good idea.

UiWindowBasics and UiWindowContents and UiWindowDefs

See the attached listings of UiDefs and UiWindowDefs for details about a lot of the suggestions discussed below. Note that these haven't been updated yet to reflect the lastest changes in the proposals.

Note that there are few comments in UiWindowDefs. I think this makes it more readable and nobody has to worry about errors in the comments. Also, there is now documentation for the stuff here, so the need for comments is greatly lessened and there is the potential problem of keeping two versions of the same thing consistent (the comments and the documentation).

I added several types to UiWindowDefs to conform to the sub-standard of using predefined types as little as possible and to avoid keywords in calling sequences.

Remove KeyDefs and StreamDefs from the directories.

Fix bug about window.box ← box untrimmed to bitmap.

Fix bug involving MoveWindowContinuous.

Use UiDisplay and UiFonts.

A careful check should be made of the Defs versus the implementors to try to weed out any remaining discrepancies.

Add routines Validate{W/Subw}indow which would raise the ERRORs {W/Subw}indowNotEnlinked when they were called to check parameters to other routines. The errors {W/Subw}indowAlreadyEnlinked should also be raised under the appropriate circumstances.

The Window Manager is going to be the prime, if not the only, user of several routines in UiWindowBasics, e.g., MoveWindow. This is because tools are not generally supposed to do things like Move or Zoom their windows, but rather are to leave that up to the user to do using the Window Manager. I designed the Window Manager and then realized that there was probably little or no need for some window fields, types, and procedures. Here is a list:

Delete window.size and the type WindowSizeType.

Delete the Places from window.{normal/tiny}Box, so they get renamed {normal/tiny}Dims and are of type Dimensions. Note that the Window Manager commands Tiny and Normal will get the Place from the cursor because they turn themselves into a Move command.

The five routines that changed a window's size, e.g., MakeWindowSizeTiny, all got coalesced into one: GrowWindow. Note that the Window Manager command Zoom will get the window's Dimensions from the bitmap's Dimensions. Unzoom, Tiny, and Normal can look into the WindowObject to get the Dimensions.

Note that the tiny and normal parameters to CreateWindow are now Dimensions rather than BitmapBoxes.

In writing a MenuPbkPNR, I discovered that I wanted to specify the INSIDE dimensions of the window to CreateWindow. Since I had no easy way of knowing the dimensions of the window frame, I couldn't specify the OUTSIDE dimensions easily. I would think that a almost all users of CreateWindow would rather specify the inside dimensions too. Also, when it comes time for a tool to divide up it's window into subwindows, I think it would like to know the space it has to work with, i.e., the inside dimensions. But note that it can only use the dimensions of the main subwindow once for this, since the tool will probably shrink that subwindow to make room for others (otherwise it's going to have to deal with overlapping subwindows). On the other hand, all sorts of routines want to know the outside box of a window, e.g., to decide what's visible. I also think that all windows ought to have frames and names and thus name frames. These considerations led to the following suggestions:

Have both a window.inBox and a window.outBox rather than just the present window.box. They are redundant in that each can be calculated from the other, but they ought to serve as "accelerators". It might be sufficient to have just an outBox and an inDims.

Change the meaning of the normal and tiny Dimensions parameters to CreateWindow to refer to inBoxes rather than outBoxes (or we could add a parameter to the calling sequence to say which box the caller wanted or we could add a routine that converted an outBox to an inBox). Note that the boxes parameters of AdjustWindowBoxProc and GrowWindow are similarly affected.

Have CreateWindow NOT create the main subwindow. I think the primary purpose of that subwindow was to provide the information about the inner space of the window to the tool. That's now available via window.inBox. The "specialness" of the main subwindow always seemed a little questionable to me. This way there will be no "special" subwindows.

I think all WindowPlaces ought to be relative to inBox rather than outBox.

CreateWindow takes the window's name as a parameter. It should make its own copy of this string. DrawWindowNameFrame is deleted, but DisplayWindowFrame always does a name frame. DisplayWindow should always call DisplayWindowFrame. This should mean that DisplayWindowFrame will rarely, if ever, be called by anybody but DisplayWindow.

Note that now DisplayWindowFrame will NOT change the box of any subwindow.

If we wanted to retain the possibilities of windows without any frames or with just a border and no name, we could adopt the following (rather hokey) convention: if window.name = NIL, that means no frame at all; if the length of the name string is 0, that means just a border, but no name.

Partly in order to make the creation of subwindows and windows more parallel and partly to help out TexTool, I suggest that the enlinking of the window be removed from CreateWindow. Instead, the two routines {En/De}LinkWindow have been added. For the same reasons, I have moved the place parameters from the Create routines to the Enlink routines for both windows and subwindows. Presumably parallel to the subwindow linking procedures, they will cause the display to be updated appropriately.

I renamed the old type name "AdjustProcType" to be "AdjustWindowBoxProc". I choose that name since it gets called every time someone tries to change (adjust) the bitmap box of the window.

There are a couple of strong reasons why I think that the refresh proc ought to be in the SubwindowObject rather than the WindowObject, but it would take too long to explain here. Note that this idea calls for some changes in the calling sequences to CreateWindow and CreateSubwindow. Also, I changed the name of the refresh proc type to be "RefreshSubwindowProc".

Note that I added the parameter PNRsHandle to CreateSubwindow. Since all subwindows should have a set of PNRs, this seems reasonable. Note, however, that this would blow the scheme of making PNRsObjects into contexts and moving the PNRs definitions to their own Defs file.

I changed the field name "backLink" in SubwindowObject to because someone might think it was a link back up the subwindow chain, i.e., to the previous subwindow on the chain. "window" seems like a better name to me.

I changed the field name "contexts" in SubwindowObject to "contextChain" to conform to the sub-standards.

Note that DestroyWindow and DestroySubwindow have had the suffix "Etc" added since they both destroy contexts also (contexts may now be hung off of windows too -- see UiContexts).

I added "Put" to the names of WindowOnto{Top/Bottom} to conform to the convention that all such procedure names should contain a verb. Similar reasoning led to the adding of "Find" to WindowFromBitmapPlace.

I deleted all the box conversion routines because the place conversion routines ought to serve well enough since the Dimensions don't ever get converted and now you can say box.place ← Convert [box.place] whereas before you couldn't say [box.x, box.y] ← Convert [[box.x, box.y]].

Because I think they will be rarely if ever used, I suggest deleting all the routines that display something in a window rather than a subwindow. DisplayWindowFrame might have trouble, but that's the only likely drawback I can think of to this suggestion.

I picked what I think will be the only commonly used display content routines and left them in (and changed the name of "ReplaceCharacterToPlaceInSubwindow" to be "DisplayCharacterInSubwindow" and added the routine "DisplayStringInSubwindow"). All the other display routines got coalesced into the three Bitblt{Pattern/Array/Character}To{Box/Place}InSubwindow routines. I never could figure out what the old routines did anyway. I'm assuming that these three routines can do everything that the 11 routines that they replace could do.

Note that with the proposed set of procedures and name changes, the rather confusing (at least to me) use of the terms Paint, Draw, Shade, Display, Alter, etc. has been mostly eliminated or least made simpler and I hope consistent and "intuitive".

Maybe each window should have an additional field curDims. This field would be used when a user moves a window over the edge of the bitmap and then back again. Note that when the window goes off the edge, curBox gets trimmed. curDims would be used to remember what size the user really wanted his window to be.

UiMenus and UiMenuCS (Menu Command Specification)

The scheme I propose is (in rough outline):

Each window has a ring of menus associated with it. The number of items in (some) menus may change (they're called "dynamic", "static" if the number of items is always constant). All menu items have "keywords". There are three types of menu items. A "leaf" item has a MCR (Menu Command Routine). A "switch" item has a BOOLEAN (more on these later). A "branch" item has a pointer to a (sub)menu. Menus containing one or more branch items are considered to be the root of a "tree" of menus.

Most of the credit for the ideas about rings and trees of menus goes to Bob.

There are three methods provided by the User Interface for specifying a menu command to be executed (a MCR). One may use the MenuPbkPNR (MPP) which displays menus while a PBK is held down and "selects" menu items. One may use Menu Subwindows (MSWs). The Menu Subwindow mechanism is a lot like the MenuPbkPNR mechanism except that the menus are always displayed (whether a PBK is held down or not) and if the subwindow is too small to display the whole menu, it may be scrolled. A menu command may also be specified via type-in (MCT). In geneneral the user types enough characters to unambiguously specify a keyword.

Now here are some implementation details:

(Almost) all windows would have exactly one ring of menu trees (a window can't have more than one, but it could have zero). The MenuRingObject structure would be a context hanging off the window. A MenuRingObject has two fields: curInst which points to the "current" MenuInstanceObject in the ring. Note that the "ring" is a ring of MenuInstanceObjects. The other field points to the "current" MenuObject (more on this later).

Because dynamic menus are allowed, there must be an extra level of indirection provided since the actual menu items move around as the array changes size. This extra level of indirection is provided by the MenuInstanceObjects. Menu instances are also used because there are expected to be a large number of instances of some menus, e.g., the TexTool and Window Manager menus.

A MenuObject has fields that say whether it's a static or dynamic menu, count how may times this menu has been instantiated (so there won't be any dangling references), give the width of the widest keyword (an accelerator when displaying menus), name it (I think all menus ought to have names), and describe the whereabouts and number of the actual menu items (the array descriptor).

I advocate providing for dynamic menus and menu trees (branch items) mostly because I think they will prove to be very useful. TexTool already uses both. Bob has said something about how nice it might be to select which font you wanted from a list (probably via branch item with a dynamic submenu). Jim has indicated that his Xref tool would like to have different items in its menu depending on circumstances (this may be a case of a "variable" rather than a "dynamic" menu).

Note: Smokey and I agreed yesterday that we would use MenuInstanceObjects so that some sort of dynamic menus could be implemented. But we also agreed that the mechanisms for dynamic menus would be PRIVATE (not part of the "official" or public part of the User Interface).

Add a MenuPbkPNR.

Add a facility for Menu Subwindows.

UiSelections

Sometime soon, add various text selection PBK PNRs. I haven't specified any of these yet.

See the discussion of the problem of selections above (in the "Problems" section) for more information.

Sometime soon, we need to agree on a full-blown selection mechanism.

UiCursors

Make the necessary changes to StoreCursor (and SwapCursor) to implement the fix for the hot spot problem discussed above.

Rename CursorRecord to be CursorObject (as per sub-standards).

Define a CursorArray type. Using that type, make the code in UiCursors shorter by saying things like "hardCursorArrayP↑ ← newCursor.array;" rather than using a FOR loop.

Use the GetUniqueCursorType scheme. If this isn't done, then there must be a constant declared that says which values of a CursorType are pre-defined and which are not.

See the attached Defs file listing for details about most of the above proposed changes.

Define lots of CursorTypes.

Eventually, we will want to keep cursor definitions in a file and probably implement some sort of caching scheme. Also we'll probably want a tool to manipulate the file and define new cursors.

UiUAS

Because User Action Streams are implemented via PORTs, there can be only one User Action Stream per tool instance (the reasons are too hard to explain here). The solution I propose is to put a "uas" field in ToolInstanceObjects and create User Action Streams relative to a tool, not a window or subwindow as before.

Note that now you can have different combinations of PNRs in different subwindows used for the same User Action Stream. The new scheme should use a good deal less heap space. Also the interface is simpler. The Menu Command Type-in implementation will be changed similarly, except that you can have a MCT per window.

See the attached listing of UiUasDefs for the details of the new scheme.

I would like to change the DestroyUasContextProc to just generate an ERROR if it ever gets called. The idea is that a tool should close the User Action Stream for a subwindow before it tries to destroy the subwindow.

Sometime, DestroyUasForTool should release any local frames left around by the coroutines.

UiMisc

See the attached listing of UiMiscDefs for details about most of the stuff discussed below.

Move stuff from TexMisc to here.   -- done

Use UiDisplay and UiFonts.

Delete BlinkScreen (it's done as BlinkDisplay in UiDisplay now).

Add ComputeStringWidth.

A convenience routine that repeatedly calls ComputeCharWidth. Perhaps this would be better put in UiFonts.

Add NopPbkPNR.   -- done

This should probably only be used by TexProcLev when one of the "blank" PBKs goes down (the "wiggling" bit problem). Everybody else should use IgnorePbkPNR so that the user is warned that his type-in has been ignored via the blinking of the screen.

Make yellow be the same "shift key" as blue for Paddle-Button chords.

Since blue will usually be used as the menu button and since currently red and yellow are treated as equivalent shift keys when using the keyset, it makes sense to put the two "shift" buttons on the two least used buttons. Smokey and I will have to learn some new habits, but that's probably the only drawback to this proposal.

Add Prompt, Answer, FileName, and PnrChoices types.

The first three types are defined according to the reasoning in "sub-standards" so that keywords don't have to appear in Defs files and so that predefined types occur as little as possible. PnrChoices is used by both the User Action Stream and Menu Command Type-in stuff.

Add AskUserForParameter and delete AskUserForConfirmation.

This is the first cut at Secondary Parameter Specification. The clients of this program may supply a subwindow in which the dialog will take place or they may say NIL in which case the Mesa window will be used. This routine will type the prompt in the appropriate place and then somehow grab the machine to receive all user actions until a Command-Accept is typed. The typed characters will be echoed and put in the Answer string supplied by the client. At least for now, it's up to the client to convert the Answer string to whatever form it needs.

Add NopRefreshProc.

This is a RefreshSubwindowProc (see UiWindows). It could be used by subwindows with no "content", scroll bar subwindows for instance.

Add TextFileRefreshProc (sometime very soon).

This is a RefreshSubwindowProc (see UiWindows). It could be used by text file and typescript subwindows for instance. Initially it will just do straight text files. Eventually there should be Bravo and/or Diamond file refresh procs (and maybe Markup, Press, SIL, etc.).

Add {Create/Destroy}{Vertical/Horizontal}ScrollBarSubwindow.

These routines will create a generalized scroll bar subwindow, using some standard conventions about how they work. I have not yet worked out the calling sequence for what I have called the ScrollProc type, but it shouldn't be too hard.

Add {Create/Destroy}TypeScriptSubwindowEtc (sometime very soon).

This will create a subwindow which may be typed into and the typing will be echoed, put in a file, and selected. There will also be a veritcal scroll bar subwindow created (that's the reason for the Etc in the name). The subwindow will use the standard text selection button PNRs.

Add {Create/Destroy}TextFileSubwindowEtc (sometime very soon).

This will create a subwindow in which a text file will be displayed. There will also be a veritcal scroll bar subwindow created (that's the reason for the Etc in the name). The subwindow will use the standard text selection button PNRs.

UiContexts

See the listing of UiDefs for some minor changes.

I'm not sure that the enumeration of context types is right.

Add facility for hanging contexts off of windows (besides subwindows).
Note that this proposal is not reflected in any of the Defs file listings yet.

Librarian Interface (TLI or Tli) -- LibiDefs

I find it difficult to suggest improvements or discern defficiencies in the Librarian Interface because I don't understand it very well. I think my lack of understanding arises primarily from two factors: The documentation (and routines and parameters) are not consistent or clear about naming things. Also the documentation could be better organized and give clearer explanations of things.(I also think there are several misleading or even erroneous statements in the spec). I would suggest a couple of things:

Smokey, I very strongly urge you go through all of the Librarian, very carefully figure out the structure of both the code and the data, decide exactly what the pieces or "abstractions" are, give exactly one name to each thing, and very carefully define what it is exactly that each name refers to (such definitions should include exactly what pieces (if any) a thing is made up of. I urge you to write this all down.

Such a document would be a great help to all of us in understanding the Librarian. Remember that future users of the Librarian Interace are going to have to be able to understand the documentation or you're going to be constantly plagued by questions.

I then suggest that you go through your spec and use all the defined names in the proper manner. This would include changing procedure and type names where necessary. Then you could make any necessary changes in the code.

All of this naming and defining is going to have to be done sooner or later. I suggest that the sooner it gets done, the better the Librarian is going to turn out. The more people understand about your design, the more they can help. I hope that you'll do the naming and defining this week. The changing of the documentation and code could be left for a little while longer.

Smokey, I hope you won't be upset by these suggestions. They are just some advice from me to be taken for what its worth. I really do feel that following these suggestions will lead to a better "Development Environment". The Librarian is so central to our whole project that I think it is particularly important that it gets done as well as we can possibly do it, and I think having more people able to make informed suggestions about it would help.

LibiDefs

Remove TimeDefs from the directory. It's not referenced.

Partly because the two modules of the Interface will now be in the main binding path, but mostly because I don't think it was a good idea to start with, I suggest you delete the interface vectors. Note that those things take up a lot of extra heap space. Note also that their usefulness will be mostly gone when the new binding scheme happens.

LibjectLoader

Move all this code to TeInit.

LibiObjects

Remove SysDefs and InlineDefs from the directory. They're not referenced and SysDefs.xm is taking up (unnecessary) space on my disk.

It's of course necessary to use the Pup interface vector, but I think you ought not to use the Mesa vector.

Write a "real" LibjectContent{File/Stream}.

Implement SnapShots (maybe this only gets done in the Librarian Server?).

LibiPropLists

The four GetProperty~ routines might be coalesced into one.

The Make{Empty/String/Record}Pair routines don't seem very useful to me. Wouldn't constructors work as well?

It's of course necessary to use the Pup interface vector, but I think you ought not to use the Mesa vector.

I would suggest that you seriously consider using "dynamic" property lists (roughly analagous to dynamic menus). Then user programs would never have to worry or guess at what size to allocate. Code for expanding or contracting property lists would only have to be written once. The BundleOfBits routines would go away, as would the PropertyListFull SIGNAL. I would then suggest that probably ALL storage allocation and freeing could be put in the Librarian Interface and no user program would have to worry about it.

Eventually, there should be a way to "register" new property numbers. Perhaps by having something analagous to GetUniqueContextType that would live in the Server.

FIFO Interface (TFI or Tfi) -- FifoiDefs

I have a few more very minor suggested changes. I've noted them on my latest copy of the functional spec.

Pup Package   -- PupDefs

We haven't loaded the FTP part of the package yet.

There are dangling references in the Mesa interface vector. There'll be more as we throw away more pieces of the Mesa Runtime System. It's a crock! Don't use it!

```
-- File:  UiDefs.mesa;  Last modified by:  Parsley,  24 July 1977

DIRECTORY  TexDefs:  FROM "TexDefs";


UiDefs:  DEFINITIONS = BEGIN

--  Coords, Dimensions, Places, and Boxes

  Coord:  TYPE = INTEGER;
  Coords:  TYPE = RECORD [x, y:  Coord];

  Dimension:  TYPE = CARDINAL;
  Dimensions:  TYPE = RECORD [w, h:  Dimension];

  ScreenPlace:  TYPE = RECORD [x, y:  Coord];
  BitmapPlace:  TYPE = RECORD [x, y:  Coord];
  WindowPlace:  TYPE = RECORD [x, y:  Coord];
  SubwindowPlace:  TYPE = RECORD [x, y:  Coord];

  ScreenBox:  TYPE = RECORD [place: ScreenPlace,  dims: Dimensions];
  BitmapBox:  TYPE = RECORD [place: BitmapPlace,  dims: Dimensions];
  WindowBox:  TYPE = RECORD [place: WindowPlace,  dims: Dimensions];
  SubwindowBox:  TYPE = RECORD [place: SubwindowPlace,  dims: Dimensions];

--  Windows

  WindowObject:  TYPE = RECORD [
    link:  PRIVATE WindowHandle,
    subwindowChain:  SubwindowHandle,
    inBox:  BitmapBox,
    outBox:  BitmapBox,
    adjustProc:  AdjustWindowBoxProc,
    name:  WindowName,
    tinyDims:  Dimensions,
    normalDims:  Dimensions ];
  WindowHandle:  TYPE = POINTER TO WindowObject;

  AdjustWindowBoxProc:  TYPE = PROCEDURE
    [WindowHandle, BitmapBox] RETURNS [BitmapBox];

  WindowName:  TYPE = STRING;

--  Subwindows

  SubwindowObject:  TYPE = RECORD [
    link:  SubwindowHandle,
    window:  WindowHandle,
    box:  WindowBox,
    pnrs:  PNRsHandle,
    refreshProc:  RefreshSubwindowProc,
    entered:  PRIVATE BOOLEAN,
    contextChain:  PRIVATE ContextHandle ];
  SubwindowHandle:  TYPE = POINTER TO SubwindowObject;

  RefreshSubwindowProc:  TYPE = PROCEDURE
    [SubwindowHandle, SubwindowBox, RefreshSequencer] RETURNS [SubwindowBox];
  RefreshSequencer:  TYPE = RECORD [first, last:  BOOLEAN];

--  PNRs

  PNRsObject:  TYPE = RECORD [
    cursorPNR:  CursorPnrType,
    keysetPNR:  PbkPnrType,
    keyboardPNR:  PbkPnrType,
    redButtonPNR, yellowButtonPNR, blueButtonPNR:  PbkPnrType ];
  PNRsHandle:  TYPE = POINTER TO PNRsObject;

  CursorPnrType:  TYPE = PROCEDURE [SubwindowHandle, TexDefs.EnterExit];

  PbkPnrType:  TYPE = PROCEDURE
    [TexDefs.PBK,  TexDefs.UpDown,  SubwindowHandle,  SubwindowPlace];

--  Contexts

  ContextObject:  TYPE = RECORD [
    link:  PRIVATE ContextHandle,
    type:  ContextType,
    own:  OwnContextData,
    destroyProc:  DestroyContextProcType ];
  ContextHandle:  TYPE = POINTER TO ContextObject;

  ContextType:  TYPE = {menu, mct, uas, selection, textFile, typeIn};

  OwnContextData:  TYPE = UNSPECIFIED;

  DestroyContextProcType:  TYPE = PROCEDURE [ContextHandle, SubwindowHandle];

END.
```

```
-- File: UiCursorDefs.mesa: Last modified by: Parsley   24 July 1977

DIRECTORY
  TexDefs: FROM "TexDefs",
  UiDefs: FROM "UiDefs";


UiCursorDefs: DEFINITIONS = BEGIN

--  Types and Constants

  CursorObject: TYPE = RECORD [
    info: CursorInfo,
    array: CursorArray ];
  CursorHandle: TYPE = POINTER TO CursorObject;

  CursorInfo: TYPE = RECORD [
    type: CursorType,
    hotSpot: CursorHotSpot ];
  CursorInfoHandle: TYPE = POINTER TO CursorInfo;

  CursorArray: TYPE = ARRAY [0..16) OF WORD;
  CursorArrayP: TYPE = POINTER TO CursorArray;

  CursorType: TYPE = {testPointer, ...};

  CursorHotSpot: TYPE = RECORD [x, y: [0..16)];

  hardCursorArrayP: CursorArrayP = PRIVATE LOOPHOLE[431B];

--  Procedural Interface

  GetCursorFromType: PROCEDURE [CursorType] RETURNS [CursorHandle];

  StoreCursor: PROCEDURE [CursorHandle];
  FetchCursor: PROCEDURE [CursorHandle];
  SwapCursors: PROCEDURE [old, new: CursorHandle];

  GetCurrentCursorInfo: PROCEDURE RETURNS [CursorInfoHandle];

  GetUniqueCursorType: PROCEDURE RETURNS [CursorType];

  BitmapPlaceFromCurrentCursorAndXY: PROCEDURE [TexDefs.CursorXY]
    RETURNS [UiDefs.BitmapPlace];

--  Signals and Errors

END.
```

```
-- File:  UiFontDefs.mesa;  Last modified by:  Parsley.  24 July 1977

DIRECTORY  SegmentDefs:  FROM "SegmentDefs";

UiFontDefs: DEFINITIONS =
  BEGIN

-- Font Records

  FontHeight, CharWidth, StringWidth:  TYPE = CARDINAL;

  FontHandle:  TYPE = POINTER TO FontArray;

  FHptr: TYPE = POINTER TO FontHeader;
  Fptr: TYPE = POINTER TO FONT;
  FCDptr: TYPE = POINTER TO FCD;
  FAptr: TYPE = POINTER TO FontArray;
  FontArray: TYPE = ARRAY [0..256) OF FCDptr;

  FONT: TYPE = MACHINE DEPENDENT  RECORD
    [
    FHeader: FontHeader,
    FCDptrs: FontArray,         -- array of self-relative pointers to
      -- FCD's.  Indexed by char value.
      -- font pointer points hear!
    ExtFCDptrs: FontArray        --  array of self-relative pointers to
      -- FCD's for extentions.  As large an
      -- array as needed.
    ];

  FontHeader: TYPE = MACHINE DEPENDENT RECORD
    [
    maxHeight: CARDINAL, -- height of tallest char in font (scan lines)
    variableWidth: [0..1], -- IF TRUE, proportionally spaced font
    blank: [0..177B], -- not used
    maxWidth: [0..377B] -- width of widest char in font (raster units).
    ];

  FCD: TYPE = MACHINE DEPENDENT  RECORD
    [
    widthORext: [0..77777B], -- width or extention index
    hasNoExtension: BOOLEAN, -- TRUE=> no ext.;prevfield=width
    height: [0..377B],    -- # scan lines to skip for char
    displacement: [0..377B]     -- displacement back to char bitmap
    ];

-- Font Procedures

  GetSystemFont: PUBLIC PROCEDURE
    RETURNS [FAptr, CARDINAL];
  GetFont: PUBLIC PROCEDURE [filename: STRING]
    RETURNS [SegmentDefs.FileSegmentHandle];
  LoadFont: PUBLIC PROCEDURE [segment: SegmentDefs.FileSegmentHandle]
    RETURNS [p: Fptr];
  ComputeCharWidth: PUBLIC PROCEDURE
    [char: CHARACTER, font: POINTER]
    RETURNS [CARDINAL];

END. of ToolFontDefs
```

```
-- File: UiMenuDefs.mesa; Last modified by: Parsley, 21 July 1977

DIRECTORY
  UiDefs: FROM "UiDefs",
  UiFontDefs: FROM "UiFontDefs";


UiMenuDefs: DEFINITIONS = BEGIN

-- Types and Constants

  MenuRingObject: PRIVATE TYPE = RECORD [
    curInst: MenuInstanceHandle,
    curMenu: MenuHandle];
  MenuRingHandle: PRIVATE TYPE = POINTER TO MenuRingObject;

  MenuInstanceObject: PRIVATE TYPE = RECORD [
    link: MenuInstanceHandle,
    menu: MenuHandle ];
  MenuInstanceHandle: PRIVATE TYPE = POINTER TO MenuInstanceObject;

  MenuObject: PRIVATE TYPE = RECORD [
    static: BOOLEAN,
    nInstances: [0..177B],
    widestKeyword: [0..377B],
    name: MenuName,
    items: MenuItemArrayD ];
  MenuHandle: TYPE = POINTER TO MenuObject;

  MenuName: TYPE = STRING;

  MenuItemArrayD: TYPE = DESCRIPTOR FOR ARRAY OF MenuItemObject;
  MenuItemArrayDHandle: TYPE = POINTER TO MenuItemArrayD;

  MenuItemObject: TYPE = RECORD [
    keyword: MenuKeyword,
    variant: SELECT COMPUTED MenuItemType FROM
      leaf => [mcrProc: McrType],
      branch => [subMenu: MenuHandle],
      switch => [onOff: BOOLEAN],
      ENDCASE ];
  MenuItemHandle: TYPE = POINTER TO MenuItemObject;
  MenuItemIndex: TYPE = CARDINAL;

  MenuItemType: TYPE = {leaf, branch, switch};

  MenuKeyword: TYPE = STRING;
  menuKeywordBranch: CHARACTER = '↑;
  menuKeywordSwitch: CHARACTER = '?;

  McrType: TYPE = PROCEDURE
    [UiDefs.SubwindowHandle, MenuHandle, MenuItemIndex];

-- Procedural Interface

  CreateStaticMenu: PROCEDURE [MenuItemArrayD, MenuName]
    RETURNS [MenuHandle];
  DestroyStaticMenu: PROCEDURE [MenuHandle];

  CreateDynamicMenu: PROCEDURE [MenuName] RETURNS [MenuHandle];
  DestroyDynamicMenu: PROCEDURE [MenuHandle];

  AddItemToDynamicMenu: PROCEDURE [MenuHandle, MenuItemObject];
  DeleteItemFromDynamicMenu: PROCEDURE [MenuHandle, MenuItemHandle];
  DeleteIndexFromDynamicMenu: PROCEDURE [MenuHandle, MenuItemIndex];
  DeleteKeywordFromDynamicMenu:  PROCEDURE [MenuHandle, MenuKeyword];

  InstantiateMenuWithWindow:  PROCEDURE
    [MenuHandle, UiDefs.WindowHandle];
  UninstantiateMenuFromWindow:  PROCEDURE
    [MenuHandle, UiDefs.WindowHandle].

  CreateMenuKeyword:  PRIVATE PROCEDURE [rawKey: MenuKeyword]
    RETURNS [newKey: MenuKeyword];
  DestroyMenuKeyword:  PRIVATE PROCEDURE [MenuKeyword];
  AddUndefItemToDynamicMenu:  PRIVATE PROCEDURE
    [MenuHandle, MenuKeyword] RETURNS [MenuItemHandle];
  DeleteUndefItemFromDynamicMenu: PRIVATE PROCEDURE
    [MenuHandle, MenuItemIndex];
  FixMenuKeywords:  PRIVATE PROCEDURE
    [menu: MenuHandle, newIndex: MenuItemIndex];
  DestroyMenuRingContext:  PRIVATE UiDefs.DestroyContextProcType;

  GetMenuFont:  PRIVATE PROCEDURE
    RETURNS [UiFontDefs.FontHandle, UiFontDefs.FontHeight];
  ChangeMenuFont:  PRIVATE PROCEDURE
    [UiFontDefs.FontHandle, UiFontDefs.FontHeight];

  CreateMenuRingForWindow:  PRIVATE PROCEDURE [UiDefs.WindowHandle];
  DestroyMenuRingFromWindow:  PRIVATE PROCEDURE [UiDefs.WindowHandle];

-- Signals and Errors

  MenuItemNotFound:  ERROR;
  MenuInstanceNotFound:  ERROR;
  MenuKeywordsConflict:  ERROR;
  MenuStatDynError:  ERROR;
  MenuInUse:  ERROR;

END.
```

```
-- File: "UiMenuCsDef[illegible]  -- Last modified by: Pressley, 21 July 1977
--  MenuCS stands for [illegible]
    MPP stands for Menu [illegible] MPP
    MSW stands for M[illegible]
--  MCT stands for Menu [illegible]

DIRECTORY
  TexDefs: FROM "TexDefs",
  UiDefs: FROM "UiDefs",
  UiMenuDefs: FROM "UiMenuDefs",
  UiMiscDefs: FROM "UiMiscDefs",
  IODefs:  FROM "IODefs";

DEFINITIONS FROM UiMenuDefs;


UiMenuCsDefs:  DEFINITIONS = BEGIN

-- Types and Constants

  -- Types and Constants used by more than one of MPP, MSW, MCT

    MenuMatrixDimension:  PRIVATE TYPE = [0..377B);
    menuMargin:  PRIVATE CARDINAL = 2;

  -- MPP's Types and Constants

    noFlip:  PRIVATE CARDINAL = 16;
      -- if cursor is this close to a Menu Window, it won't flip

  -- MSW's Types and Constants

    MswObject:  PRIVATE TYPE = RECORD [  -- pointed to by context.own
      scrollMenuVerticalSw, scrollMenuHorizontalSw:  UiDefs.SubwindowHandle,
      nHorItems, nVerItems:  MenuMatrixDimension,
      horOffset, verOffset:  MenuMatrixDimension,
      variant:  SELECT MswType FROM
        ring => [scrollRingSw:  UiDefs.SubwindowHandle],
        tree => [rootMenu, curMenu:  MenuHandle],
        ENDCASE ]
    MswHandle:  PRIVATE TYPE = POINTER TO MswObject;

    MswType:  PRIVATE TYPE = {tree, ring}

  -- MCT's Types and Constants

    MctObject:  PRIVATE TYPE = RECORD [  -- pointed to by mainSw.context.own
      feedBack:  UiDefs.SubwindowHandle,
      pbs:  TexDefs.PaddlesButtons,
      partial:  STRING ];
    MctHandle:  PRIVATE TYPE = POINTER TO MctObject;

    mctPartialInitLength:  PRIVATE CARDINAL = 4;
    mctUserCA:  PRIVATE CHARACTER = IODefs.ESC;

-- Procedural Interface

  -- Procedures used by more than one of MPP, MSW, MCT

    RefreshMSW:  PRIVATE UiDefs.RefreshSubwindowProc;
    PaintMSW:  PRIVATE PROCEDURE [sw: UiDefs.SubwindowHandle,
      nHorItems, nVerItems, horOffset, verOffset:  MenuMatrixDimension];

  -- MPP's Procedural Interface
```

```
    [illegible]: [illegible];

  -- MSW's Procedural Interface

    [illegible]: PROCEDURE [illegible];
      -- if [illegible] NIL, then it will [illegible] else a [illegible] MSW
    DestroyMSW: PROCEDURE [illegible];

  -- MCT's Procedural Interface

    CreateMctForWindow:  PROCEDURE
      [UiDefs.WindowHandle, UiDefs.SubwindowHandle];
      -- subwindow is used for feedback:  if NIL, use "system" window
    DestroyMctFromWindow:  PROCEDURE [UiDefs.WindowHandle];

    OpenMctForWindow:  PROCEDURE
      [UiDefs.WindowHandle, UiMiscDefs.PnrChoices];
    OpenMctForSubwindow:  PROCEDURE
      [UiDefs.SubwindowHandle, UiMiscDefs.PnrChoices];
    CloseMctForWindow:  PROCEDURE [UiDefs.WindowHandle];
    CloseMctForSubwindow:  PROCEDURE [UiDefs.SubwindowHandle];

-- Signals and Errors

  SubwindowAlreadyIsMsw:  SIGNAL;
  MswNotLargeEnough:  SIGNAL;
  MswTreeMenuNotInRing:  ERROR;
  IllegalMsw:  ERROR;

  WindowAlreadyHasMCT:  ERROR;
  IllegalMctFeedBackSubwindow:  ERROR;
  WindowHasNoMCT:  ERROR;
  InvalidPNRsForMCT:  ERROR;

END.
```

```
 -- File: UiMiscDefs.Mesa  Last modified by: [illegible], 28 Jul, 1977 --

DIRECTORY
  TexDefs: FROM "TexDefs",
  UiDefs: FROM "UiDefs",
  UiFontDefs: FROM "UiFontDefs",
  StreamDefs: FROM "StreamDefs";


UiMiscDefs:  DEFINITIONS = BEGIN

-- Types and Constants

  KeyCode:  TYPE = [0..177B];

  KeyDescription:  TYPE = RECORD [
    useLock:  BOOLEAN,
    shiftCode:    KeyCode,
    normalCode:  KeyCode ];

  KeysDescriptionTable:  TYPE = ARRAY TexDefs.Key OF KeyDescription;

  PaddlesTransString:  TYPE = STRING;
  ButtonsTransTable:   TYPE = ARRAY [0..8) OF CHARACTER;
  ButtonsShiftTable:   TYPE = ARRAY [0..8) OF CARDINAL;

  Prompt, Answer, FileName:  TYPE = STRING

  PnrChoices:  TYPE = RECORD [keyboard, keyset, red,yellow,blue:  BOOLEAN];

-- Procedural Interface

  IgnorePbkPNR:  UiDefs.PbkPnrType;
  NopPbkPNR:  UiDefs.PbkPnrType;
  NopCursorPNR:  UiDefs.CursorPnrType;

  NopRefreshProc:  UiDefs.RefreshSubwindowProc;
  TextFileRefreshProc:  UiDefs.RefreshSubwindowProc;

  CreateVerticalScrollBarSubwindow:  PROCEDURE
    [UiDefs.SubwindowHandle, ScrollProc] RETURNS [UiDefs.SubwindowHandle];
  CreateHorizontalScrollBarSubwindow:  PROCEDURE
    [UiDefs.SubwindowHandle, ScrollProc] RETURNS [UiDefs.SubwindowHandle];
  DestroyScrollBarSubwindow:  PROCEDURE [UiDefs.SubwindowHandle];
  ScrollProc:  TYPE = PROCEDURE [UiDefs.SubwindowHandle--??--];

  CreateTextFileSubwindowEtc:  PROCEDURE
    [UiDefs.WindowBox, StreamDefs.DiskHandle]
    RETURNS [UiDefs.SubwindowHandle];
  DestroyTextFileSubwindowEtc:  PROCEDURE [UiDefs.SubwindowHandle];

  CreateTypeInSubwindowEtc:  PROCEDURE
    [UiDefs.WindowBox, FileName]
    RETURNS [UiDefs.SubwindowHandle];
  DestroyTypeInSubwindowEtc:  PROCEDURE [UiDefs.SubwindowHandle];

  PostMessage:   PROCEDURE [STRING];
  PostMessage2:  PROCEDURE [STRING, STRING];
  PostMsgCont:  PROCEDURE [STRING];

  AskUserForParameter:  PROCEDURE [Prompt, Answer, UiDefs.SubwindowHandle];

  SetStateOfPBKs:  PROCEDURE [TexDefs.PBKsP, TexDefs.PBK, TexDefs.DownUp];
  SetStateOfPaddlesButtons: PROCEDURE
    [TexDefs.PaddlesButtonsP, TexDefs.PBK, TexDefs.DownUp];
  [illegible] BK. [illegible]]
    [illegible]

  [illegible] FontHandle]
    [illegible]

  [illegible] CHARACTER];
  [illegible] CHARACTER];
  ForceStringUpper: PROCEDURE [to: STRING, from: STRING];
  ForceStringLower: PROCEDURE [to: STRING, from: STRING];

  TranslatePaddlesIntoChar:
    PROCEDURE [TexDefs.Paddles, TexDefs.Buttons] RETURNS [CHARACTER];
  TranslateButtonsIntoChar:
    PROCEDURE [TexDefs.Buttons] RETURNS [CHARACTER];
  TranslateKeyIntoChar:
    PROCEDURE [TexDefs.Key, TexDefs.KeysP] RETURNS [CHARACTER];

-- Signals and Errors

END.
```

```
-- [illegible] modified by: Parsl [illegible]
 - [illegible]

DIRECTORY
  [illegible] TexDefs',
  [illegible] FROM "[illegible]",
  [illegible] FROM "UiDefs",
  UiMiscDefs: FROM "UiMiscDefs";


UiUasDefs: DEFINITIONS = BEGIN

-- Types and Constants

  UasObject: PRIVATE TYPE = RECORD [  -- is pointed to by ToolInstanceObject.uas
    pbs: TexDefs.PaddlesButtons,
    getUasCharP: GetUasCharPortP,
    putUasCharP: PutUasCharPortP,
    portBlock: UasPortBlock ];
  UasHandle: PRIVATE TYPE = POINTER TO UasObject;

  GetUasCharPortType:  TYPE = PORT RETURNS [CHARACTER];
  GetUasCharPortP:     TYPE = POINTER TO GetUasCharPortType;
  PutUasCharPortType:  PRIVATE TYPE = PORT [CHARACTER];
  PutUasCharPortP:     PRIVATE TYPE = POINTER TO PutUasCharPortType;
  UasPortBlock:        PRIVATE TYPE = ARRAY [0..8] OF UNSPECIFIED;
    --  two ports get stored here;
    --  the extra words allow the beginning addresses to end in 2B

  Tool:  TYPE = TexManipToolsDefs.ToolInstanceHandle;

-- Procedural Interface

  CreateUasForTool:  PROCEDURE [Tool];
  DestroyUasFromTool:  PROCEDURE [Tool];

  OpenUasForWindow:  PROCEDURE
    [Tool, UiDefs.WindowHandle, UiMiscDefs.PnrChoices]
    RETURNS [GetUasCharPortP];
  OpenUasForSubwindow:  PROCEDURE
    [Tool, UiDefs.SubwindowHandle, UiMiscDefs.PnrChoices]
    RETURNS [GetUasCharPortP];
  CloseUasForWindow:  PROCEDURE [UiDefs.WindowHandle];
  CloseUasForSubwindow:  PROCEDURE [UiDefs.SubwindowHandle];

-- Signals and Errors

  ToolAlreadyHasUAS:  ERROR;
  ToolHasNoUAS:  ERROR;
  InvalidPNRsForUAS:  ERROR;

END.
```

```
-- File: UiWindowDefs.mesa; Last modified by:  Parsley,  24 July 1977

DIRECTORY
  UiDefs: FROM "UiDefs",
  [illegible]: FROM "[illegible]",
  [illegible]: FROM "[illegible]";

DEFINITIONS FROM UiDefs;


UiWindowDefs:  DEFINITIONS = BEGIN

--  Types and Constants

  -- Most of the definitions of UiDefs would normally occur here

  IntegerArrayD:  TYPE = DESCRIPTOR FOR ARRAY OF INTEGER;
  Roundedness:  TYPE = [0..4];
  SubwindowPlaceArrayD:  TYPE = DESCRIPTOR FOR ARRAY OF SubwindowPlace;
  BitbltPattern:  TYPE = ARRAY [0..4) OF WORD;
  BitbltArrayP:  TYPE = POINTER;
  WordsPerLine:  TYPE = CARDINAL;

--  Procedural Interface

  -- Basic procedures;  deal with [sub]windows as a whole, not contents of

    CreateWindowEtc:  PROCEDURE [
      name:  WindowName,
      normal, tiny:  Dimensions,
      adjustProc:  AdjustWindowBoxProc]
      RETURNS [WindowHandle, SubwindowHandle];
    DestroyWindowEtc:  PROCEDURE [WindowHandle];

    NopAdjustWindowBoxProc:  AdjustWindowBoxProc;

    EnlinkWindow:  PROCEDURE [WindowHandle, BitmapPlace];
    DelinkWindow:  PROCEDURE [WindowHandle];

    CreateSubwindow:  PROCEDURE [Dimensions, PNRsHandle, RefreshSubwindowProc]
      RETURNS [SubwindowHandle];
    DestroySubwindowEtc:  PROCEDURE [SubwindowHandle];

    NopRefreshSubwindowProc:  RefreshSubwindowProc;

    EnlinkSubwindow:  PROCEDURE
      [WindowHandle, SubwindowHandle, WindowPlace];
    DelinkSubwindow:  PROCEDURE [SubwindowHandle];

    GrowWindow:  PROCEDURE [WindowHandle, BitmapBox];

    MoveWindow:  PROCEDURE [WindowHandle, BitmapPlace];

    MoveWindowContinuous:  PROCEDURE [WindowHandle, MoveContinuousProc];
      MoveContinuousProc:  TYPE = PROCEDURE
        [WindowHandle] RETURNS [BOOLEAN, BitmapPlace];

    PutWindowOntoTop:  PROCEDURE [WindowHandle];
    PutWindowOntoBottom:  PROCEDURE [WindowHandle];

  -- Conversion procedures

    FindWindowFromBitmapPlace:  PROCEDURE [BitmapPlace]
      RETURNS [WindowHandle];
  [illegible]Window[illegible]: PROCEDURE [WindowHandle, Window[illegible]]
    RETURNS [BOOLEAN];
  [illegible]InSubwindow[illegible]: PROCEDURE [SubwindowHandle, SubwindowBox]
    RETURNS [BOOLEAN];

  [illegible]PlaceInSubwindow: PROCEDURE [[illegible]WindowHandle, WindowPlace]
    RETURNS [BOOLEAN];

  BitmapPlaceFromWindowPlace:  PROCEDURE
    [WindowHandle, WindowPlace] RETURNS [BitmapPlace];
  BitmapPlaceFromSubwindowPlace:  PROCEDURE
    [SubwindowHandle, SubwindowPlace] RETURNS [BitmapPlace];

  WindowPlaceFromBitmapPlace:  PROCEDURE
    [WindowHandle, BitmapPlace] RETURNS [WindowPlace];
  WindowPlaceFromSubwindowPlace:  PROCEDURE
    [SubwindowHandle, SubwindowPlace] RETURNS [WindowPlace];

  SubwindowPlaceFromWindowPlace:  PROCEDURE
    [SubwindowHandle, WindowPlace] RETURNS [SubwindowPlace];

-- Miscellaneous display procedures

  DisplayWindow:  PROCEDURE [WindowHandle];

  DisplayWindowFrame:  PROCEDURE [WindowHandle];

  MoveBoxInSubwindow:  PROCEDURE
    [SubwindowHandle, SubwindowBox, SubwindowPlace];

  TrimSubwindowBox:  PROCEDURE [SubwindowHandle, SubwindowBox]
    RETURNS [SubwindowBox];

-- Content procedures;  display bits inside of subwindows

  DisplayCharacterInSubwindow:  PROCEDURE
    [SubwindowHandle, SubwindowPlace, CHARACTER, UiFontDefs.FontHandle]
    RETURNS [SubwindowPlace];
  DisplayStringInSubwindow:  PROCEDURE
    [SubwindowHandle, SubwindowPlace, STRING, UiFontDefs.FontHandle]
    RETURNS [SubwindowPlace];

  WhitenBoxInSubwindow:  PROCEDURE [SubwindowHandle, SubwindowBox];
  BlackenBoxInSubwindow:  PROCEDURE [SubwindowHandle, SubwindowBox];
  InvertBoxInSubwindow:  PROCEDURE [SubwindowHandle, SubwindowBox];

  BitbltPatternToBoxInSubwindow:  PROCEDURE
    [SubwindowHandle, SubwindowBox, BitbltPattern,
    UiDisplayDefs.BbSourceType, UiDisplayDefs.BbOperation];

  BitbltArrayToBoxInSubwindow:  PROCEDURE
    [SubwindowHandle, SubwindowBox, BitbltArrayP, WordsPerLine,
    UiDisplayDefs.BbSourceType, UiDisplayDefs.BbOperation];

  BitbltCharacterToPlaceInSubwindow:  PROCEDURE
    [SubwindowHandle, SubwindowPlace, CHARACTER, UiFontDefs.FontHandle,
    UiDisplayDefs.BbSourceType, UiDisplayDefs.BbOperation]
    RETURNS [SubwindowPlace];

-- Graphic content procedures;  draw bits inside of subwindows

  DrawDiagonalOfSubwindowBox:  PROCEDURE
    [SubwindowHandle, SubwindowBox];

  DrawRectilinearCurveInSubwindow:  PROCEDURE
```

```
      [illegible]

    [illegible]
      [illegible]

  -- Private Procedures

    DoSubwindowToZInWindow: PRIVATE PROCEDURE
      [WindowHandle, Window[illegible]];
    GetUiWindowBasicsBbPtr: PRIVATE PROCEDURE RETURNS
      [UiDisplayDefs.BbPtr];

--  Signals and Errors

  WindowNotEnlinked:  ERROR;
  WindowAlreadyEnlinked:  ERROR;
  SubwindowNotEnlinked:  ERROR;
  SubwindowAlreadyEnlinked:  ERROR;

END.
```