Ultimate

Goran Mitrovic

COLLABORATORS			
	TITLE : Ultimate		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Goran Mitrovic	February 12, 2023	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

Ultimate

1.1 Ultimate Patch System Manual

```
                Ultimate Patch System 1.0
by Goran Mitrovic
released on 14 October 1995.

 TABLE OF CONTENTS
```

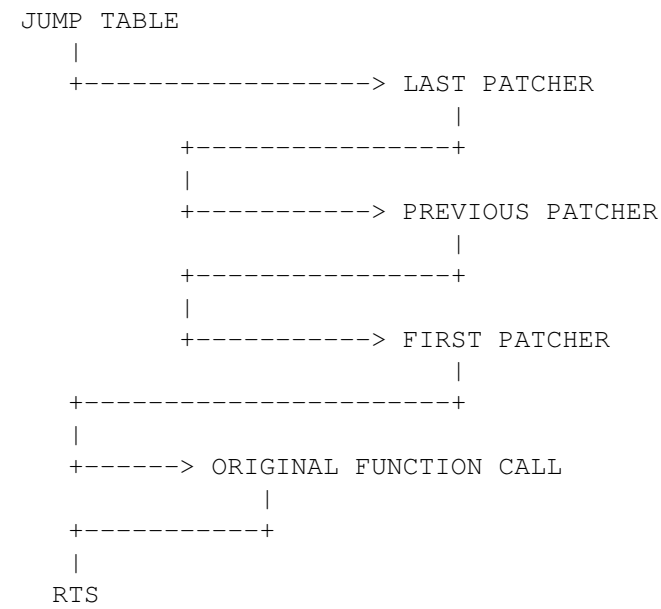

1.2 Copyright information

The Ultimate Patch System (the binary and documentation) is Copyright © 1995 Goran Mitrovic. All Rights reserved.

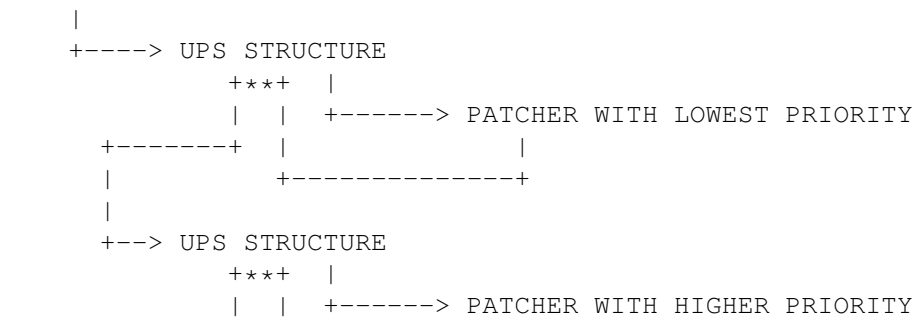
Ultimate Patch System comes with NO WARRANTIES. The author is not responsible for any loss or damage arising from the use of Ultimate Patch System; the user takes all such responsibility.

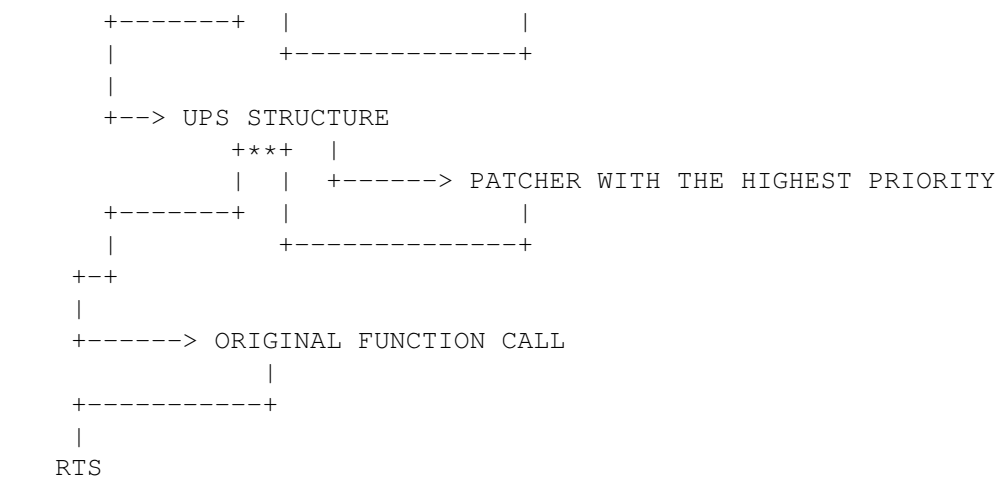
1.3 Introduction

Library functions can be patched. On the old way, server part of program have to copy patch in allocated memory, and few other things... When few patches are installed on the same function call, it looks like this:



Now, if 'first patcher' wants to be removed, it can't be done because after executing 'previous patcher' he is called, so if he is hard removed, system might crash! Here comes the Ultimate Patch System. It allows very easy creation of patches. They could be one-file patchers, or patchers made via messages from application to Ultimate Patch System. There's quite a lot of other possibilities(look 'developers' section). And, coder doesn't have to worry about the removal of his patch - it is always possible.





There. Now, when, for example, the 'higher patcher' is removed, second UPS' structure is also removed, but pointers to it in the other two structures are changed.

1.4 Requirements

There are two versions of Ultimate Patch System. For 68000 and for 68020 or higher processors. Version for 68000 is NOT tested, so it might not work. Then, ROM and Workbench must theoretically be v36 or above. Reqtools.library v38 or higher has to exist. Ram ussage is quite low.

Ultimate Patch System has been developed on: A1200, 2mb Ram, 250mb, 14400 modem, v39 OS, v40 Workbench. Soon, after starting this project, Amiga has been expanded with aditional 4mb of Ram. And, at the end of developing, Blizzard 1220 came into my Amiga.

Ultimate Patch System has been (aditionally) tested on:
A4030, 10mb Ram, 850mb, 14400 modem, v39 OS, v40 Workbench.
A4040, 14mb Ram, 1+gb, 14400 modem, v39 OS and Workbench.

1.5 Starting Ultimate Patch System

There are two ways to start Ultimate Patch System.

1. CLI

You can simply start it with executing UPS in CLI. There are few parameters which you may want to use.

```
  NOLOAD/S,POPUP/S,POPKEY/K/F
```

NOLOAD - patchers from s:PatcherList won't be loaded.
POPUP - UPS' GUI will popup.
POPKEY - after this keyword, put hotkey for GUI popup. Default is
rcommand p.

2. Workbench

Just double click on UPS' icon. Like in CLI start, you can change few parameters in tooltypes.

CX_POPKEY - after this, put hotkey for GUI popup. Default is rcommand p.
Remove brackets!
CX_POPUP - UPS' GUI.
LOAD - patchers from s:PatcherList to load?

If UPS is started once, GUI will popup when hotkey or ShowInterface from Exchange is pressed.

1.6 Using Ultimate Patch System

Using Ultimate Patch System is quite simple. In most cases, you'll just start it. :)

GUI will popup if hotkey is pressed(default is rcommand p), if ShowInterface from Exchange is pressed, or if POPUP option on startup is selected.

MAIN WINDOW

On the left side is listview in which is list of currently loaded patchers. On the right side, listview contains list of patchlinks(things what patch that patcher) of current patcher.
'Checkbox' enables/disables patcher(patchlinks).
'Remove' removes patcher.
'Prefs' calls prefs routine of patcher(patchlink).
'Load' loads new patcher(or few of them).
'Enable All' enables all of patchlinks.
'Patchers Manager' opens Patchers Manager window.
'Libraries Overview' opens Libraries Overview window.
'About' opens About requester.
'Quit' quits Ultimate Patch System.
'Hide' closes Ultimate Patch System's windows.

PATCHERS MANAGER WINDOW

In the listview are names of patchers which will be loaded at the startup.
'Load' loads new patcher(s) in startup list.
'Remove' removes selected patcher.
'Save Startup List' saves startup list in s:PatcherList.
'Exit' closes window.
Size: - size of selected patcher.
Patches: - number of patched resources(libraries) and functions.
Path: - full path to selected patcher.
Dropped icon picture won't appear. :) It's disabled for now.

LIBRARIES OVERVIEW

With big cycle gadget you select which library functions you want to see. In listview all of function offsets are listed, so you can see theirs status.
Old: - Original address of selected function.
New: - New address of selected function.
Purpose: - Very short description of patched function.
Patcher: - Name of patcher.

1.7 Given Patchers...

Following patchers are included in this distribution:

-

DPatch
-
Palette
-
AllocMem
-
unfinished patchers

1.8 DPatch

DPatch is originally made by Goran Paulin. This version is the same as his one, but it works as an Ultimate Patch System's patcher. For more info, look in original DPatch's documentation.

1.9 Palette Patcher

This isn't a real patcher at all. It shows how can you make simple hotkey program with Ultimate Patch System. When it is started, press on hotkey (lcontrol shift p) or prefs button calls Palette Requester from reqtools.library, for activated screen. If curently setted hotkey isn't good for you, feel free to change it in source, and then assemble the source again.

1.10 AllocMem Patcher

This is another simple patch. :) It changes allocating memory a bit, so memory fragmentation is reduced. It alocates half of 1024 or less bytes memory blocks in upper memory, and half of them in lower.

1.11 Unfinished Patchers

I planned to finish these two unfinished patches, but I simply didn't have time to finish it(but, in another UPS release, they _will_ be finished).

First one is SetFunction patcher. It replaces SetFunction() with UPS like one. It is almost finished, but it has bug in removing functions, and in patching few functions(from exec.library). I don't know why. In source, you can see example how can you wait till Remove button is clicked.

Second one is Icon patcher. When you install MagicWB, colors from 4 to 7 are allocated, so icons are drawn normally. But, these colors also exists on last four colors, so that means those four colors are practically wasted. My idea is to make patcher which remaps icons while they are loading. I did everything except pure icon remap. Try to finish it, if you have time. :) And, I think that old icons from ram: won't be deleted. It's very simple to do that option - just check from Close() if it is called from Workbench, and if it is matched with tempicon pattern. If both of if's are true, delete file after closing.

1.12 Problems

I think that all problems which may occure are described in Ultimate Patch System. If you don't understand something, just contact me.

1.13 Developer Informations

First, I'll explain few terms which you have to understand.

Patcher is one main structure which contains pointers to all other structures. For example, if you make patcher which replaces all functions of graphics.library, patcher should be called 'Graphics.library patcher', or similiar.

Patch is structure which is single function patch. In upper example, there is quite a lot patches.

Patchlink is also structure which contains pointers to patches, but it points to only similiar patches, which patch same area of library, or similiar functions. For example, patchlink which contains patches for functions for drawing lines, could be called 'Line Patches'.

Patchers could be made in two ways. First way is to assemble program which has patcher structure at the beggining(assemble it as a normal program, with hunks and reloc tables), which user loads directly in UPS. Second way is to find UPS port, and then send messages to it.

If you don't know how to code something, just contact me.

```
/**
***  Patcher file structure
**/

struct pfs {
  LONG        pfs_code[8];              /// 32 bytes of code
```

First 32 bytes of free code. In most cases should be RTS and 30 bytes of anything.

```
  LONG        pfs_ControlLong;          /// must be equal to ControlLong
```

Control mark.

```
  struct pfs *pfs_Next;                 /// pointer to next patcher
  struct pfs *pfs_Prev;                 /// pointer to prev patcher
```

For internal usage.

```
  struct PVS  pfs_NeedVersion;          /// minimum version of UPS to start
  struct PVS  pfs_UsedVersion;          /// version of UPS which was used
                                        /// while patcher was developed
  APTR        pfs_PatcherName;          /// pointer to name of patcher
```

For now, user cannot see that PatcherName.

```
  struct PVS  pfs_Version;              /// version
```

Version of patcher itself.

```
  APTR        pfs_CoderName;            /// pointer to name of programmer
  APTR        pfs_VersionString;        /// pointer to $VER: string
```

Both strings are optional.

```
  APTR        pfs_PatcherIDString;      /// listview entry
```

Text which will be listed in listview of patches.

```
  LONG        pfs_Flags;                /// flags
  LONG        pfs_Status;               /// status of flags
```

Flags and it’s status. Don’t change both of them. Flags are explained later, and 0 in Status field.

```
  APTR        pfs_ExecBase;             /// exec base
  APTR        pfs_IntBase;              /// intuition base
  APTR        pfs_GadTBase;             /// gadtools base
  APTR        pfs_DosBase;              /// dos base
  APTR        pfs_GfxBase;              /// graphics base
  APTR        pfs_CxBase;               /// commodities base
  APTR        pfs_LayBase;              /// layers base
  APTR        pfs_UPSBase;              /// UPS lib base
```

Library bases, so you don’t have to open them. UPS lib is reserved for future usage.

```
  APTR        pfs_Init;                 /// pointer to init routine
  APTR        pfs_Exit;                 /// pointer to exit routine

Pointers to init/exit routines, which will be executed after
opening/closing resources. On the end of Init routine, in D0 should be
placed 0 of anything went wrong.

  APTR        pfs_Prefs;                /// pointer to prefs routine
  APTR        pfs_PrefsKey;             /// pointer to cx keys to call
                                        /// prefs rout

Should be NULL if prefs flag isn't set.

  struct orl *pfs_ListPtr;              /// pointer to struct of requested
                                        /// opened resources

Pointer to Resource structure.

  struct mps *pfs_Patch;                /// pointer to struct for patches

Pointer to first Patch.

  struct pls *pfs_pls;      /// pointer to pls structure

Pointer to first PatchLink.

  APTR        pfs_UPSMsgPort;           /// pointer to main UPS msg port
  APTR        pfs_UPSRexxPort;          /// pointer to rexx port

For internal usage.

  struct pxs *pfs_pxs;                  /// pointer to pxs structure

Pointer to PatcherExtended structure.

  BOOL       *pfs_fakesemaphore;        /// TRUE for UPS's setfunction

For internal usage.

  LONG        pfs_User1;                ///
  LONG        pfs_User2;                ///

Here can be placed any user data.

  LONG        pfs_ControlLong2;         /// ControlLong

Control mark

  struct pfs *pfs_PFS;                  /// pointer to the beginning of struct

Pointer to the beginning of Patcher structure.

};
```

```
/**
***  Version strcuture
**/

struct PVS {
  WORD        PVS_Version;              /// version word

Version word.

  BYTE        PVS_Revision;             /// revision byte
  BYTE        PVS_User1;                ///
};



#define pfsControlLong  0x17061995
#define UPSMainVersion  39

First UPS Server version.

#define UPSMainRevision 0

#define pfsDisableable  0x00000001      /// Enable/disable patcher option
#define pfsPrefs        0x00000002      /// Prefs editor exists
#define pfsNoRemoveable 0x00000004      /// Disable Removing of Patcher



/**
***  Patcher Extended Structure
**/

struct pxs {
  LONG        pxs_CxID;                 /// Commodity hotkey id
  APTR        pxs_CxObjs;               /// Commodity object pointer
};


/**
***  Open Resource List
**/

struct orl {
  struct orl *orl_Next;                 /// pointer to next entry
  struct orl *orl_Prev;                 /// pointer to previous entry

Part of standard node structure. Always set it right!

  LONG        orl_Type;                 /// type of resource

Type of resource to open. Only library, for now.

  LONG        orl_ID;                   /// resource id

ID of this resource. All ID's should be different!

  LONG        orl_Flags;                /// flags
```

```
  APTR        orl_Name;                 /// pointer to name of resource
  APTR        orl_Base;                 /// base of opened resource

Here is placed base of opened resource.

  LONG        orl_Version;              /// version, if needed

If 0, OldOpenLibrary will be called.

  LONG        orl_DataRegs[8];          /// ...to put in data regs
  APTR        orl_Open;                 /// user open rout for user type
  APTR        orl_Close;                /// user close rout for user type
  APTR        orl_UTName;               /// pointer to name for user type
  LONG        orl_User1;                ///
  LONG        orl_User2;                ///

Not yet used.

};

#define orlNoNecessary     0x00000001   /// dont care if not opened

No matter if resource isn't really opened.

#define rt_Library         1            /// resource type is library



/**
***  Main patch struct
**/

struct mps {
  struct mps *mps_Next;                 /// pointer to next patch
  struct mps *mps_Prev;                 /// pointer to prev patch

Part of standard node structure. Always set it right!

  LONG        mps_Pri;                  /// priority, less runs first

Priority of Patch. It should be beetween -32768 and 32766. 32767 is
original function call, so don't use it. Higher priorities runs later.

  struct pfs *mps_Patcher;              /// pointer to parent Patcher
  APTR        mps_PatchIDString;        /// listview entry

Pointer to string which will be displayed in future listview gadget.

  APTR        mps_PurposeString;        /// pointer to purpose string

Pointer to short description string.

  LONG        mps_ID;                   /// id of resource to be patched
  LONG        mps_Flags;                /// flags
  LONG        mps_Status;               /// status of flags
  LONG        mps_Offset;               /// offset to change
```

```
Put here offset which will be patched.

  APTR        mps_New;                  /// pointer to new routine

Pointer to routine which will be patched.

  APTR        mps_Old;                  /// pointer to old routine

Pointer to old routine. If you want to call orginal function inside of
patch of the same function, do something like this:
        lea     patch,a0
        move.l  #aa,-(a7)
        move.l  mps_pls(a0),a0
        move.l  pns_Next(a0),-(a7)
        rts
aa

  struct pns *mps_pns;                  /// pointer to pns structure
  APTR        mps_NotifyDisable;        /// notified when enabled/disabled

Routine which will be executed when patcher is disabled or enabled. In D0
is 0 if enabled.

  APTR        mps_Install;              /// install rout for user type
  APTR        mps_Remove;               /// uninstall rout for user type

For future usage.

  APTR        mps_Init;                 /// pointer to init routine
  APTR        mps_Exit;                 /// pointer to exit routine
  struct pls *mps_pls;                  /// pointer to pls structure
  LONG        mps_User1;                ///
  LONG        mps_User2;                ///
  LONG        mps_User3;                ///
  LONG        mps_User4;                ///
};


#define mpsDisableable  0x00000001      /// Enable disable patch option
#define mpsFullPatch    0x00000002      /// if full, don't call orig. patch

If FullPatch option is selected then this patch is the last patch on
patched function which will be executed.

#define mpsDisTemp      0x00000004      /// While removed, previous status

Ignore this.



/**
***  Patch node struct
**/

This function is made by UPS.
```

```
struct pns {
  WORD        pns_Jsr;                  /// jsr ($4eb9)
  APTR        pns_New;                  /// newrout

Here is your new routine.

  WORD        pns_Jmp;                  /// jmp ($4ef9)
  struct pns *pns_Next;                 /// nextrout

And, jump to next pns structure...

  struct pns *pns_Prev;                 /// pointer to prev node
  LONG        pns_Pri;                  /// pri, less first
  APTR        pns_Orig;                 /// orginal routine
  struct mps *pns_Patch;                /// pointer to main patch
  struct pfs *pns_Patcher;              /// pointer to patcher
  LONG        pns_Mark;                 /// $fc263815
  struct pns *pns_pns;                  /// pointer to this structure
  LONG        pns_User1;                ///
  LONG        pns_User2;                ///

You can always use upper informations...

};



/**
***  UPS message
**/

This is what you send to UPS' message port!

struct um {
  struct Message
              um_Message;
  LONG        um_Command;               /// command to execute
  LONG        um_A0;                    /// adress register 0
  LONG        um_A1;                    /// adress register 1
  LONG        um_A2;                    /// adress register 2
  LONG        um_A3;                    /// adress register 3
  LONG        um_D0;                    /// data register 0
  LONG        um_D1;                    /// data register 1
  LONG        um_D2;                    /// data register 2
  LONG        um_D3;                    /// data register 3
};

Commands:

#define umAllocUPS      0x00000001
  Allocate UPS, in a0 ptr to application name
#define umFreeUPS       0x00000002
  Free UPS, in a0 ptr to application name
#define umLoadPatcher   0x00000010
  Initiate LoadPatcher routine
#define umAddPatcher    0x00000015
  Load Patcher, in a0 ptr to filename, in d0 error code
```

```
#define umInstall       0x00000030
  Install Patcher, in a0 ptr to patcher, in d0 error code
#define umRemove        0x00000031
  Remove Patcher, in a0 ptr to patcher
#define umOpenResource  0x00000050
  Open Resource, in a0 ptr to resource, in d0 return BOOL.
#define umCloseResource 0x00000051
  Close Resource, in a0 ptr to resource
#define umInstallPatch  0x00000040
  Install Patch, in a0 ptr to patch, in d0 error code
#define umRemovePatch   0x00000041
  Remove Patch, in a0 ptr to patch
#define umEnablePatcher 0x00000035
  Enable patcher in a0
#define umDisablePatcher 0x00000036
  Disable patcher in a0
#define umEnablePLink   0x00000037
  Enadle patch link in a0
#define umDisablePLink  0x00000038
  Disable patch link in a0
#define umOpenMain      0x00000060
  Open Main Window
#define umCloseMain     0x0000006f
  Close Main Window
#define umOpenPM        0x00000061
  Open Patcher Manager Window
#define umClosePM       0x0000006e
  Close Patcher Manager Window
#define umOpenLO        0x00000062
  Open Library Overview Window
#define umCloseLO       0x0000006d
  Close Library Overview Window
#define umSpitRequester 0x00000070
  Spit requester, with text in a0
#define umPrint         0x00000073
  Print text line in a0
#define umPrefsPatcher  0x00000078
  Call prefs from Patcher in a0
#define umPrefsPatchL   0x0000007a
  Call prefs from PatchList in a0
#define umGetLibBase    0x00000008
  Get base ptr from code in d0, result in a0
#define umGetPList      0x0000000a
  Get PatcherListStructure ptr in a0, and PList itself in d0/d1/d2
#define umMakeEasyPatcher 0x0000001c
  Make easy patcher with tagarray in a0. Pointer in a0, or null for error
#define umFreeEasyPatcher 0x0000001d
  Remove easy patcher in a0.
#define umMakeEasyPatch 0x0000001e
  Add Patch to easy patcher. Pointer to tagarray in a0, and parent patcher
  in a1. Pointer to patch in a0, null for error.
#define umFreeEasyPatch 0x0000001f
  Remove east patch in a0.
#define umNoFree        0x80000000
  Don't free memory after processing. Just add this to command codes.

#define umLibExec       0x00000001
```

```
#define umLibDOS        0x00000002
#define umLibIntuition  0x00000003
#define umLibGadTools   0x00000004
#define umLibUtility    0x00000005
#define umLibCommodities 0x0000006
#define umLibIcon       0x00000007
#define umLibReqTools   0x00000008
#define umLibLayers     0x00000009
#define umLibGraphics   0x0000000a
#define umLibWorkbench  0x0000000b


These are tags for easy patcher/patch arrays.
In easy patcher, this tags have to be set:
umTagPatcherName, umTagPatcherIDString, umTagResourceName,
umTagPatchIDString, umTagPatchOffset, umTagPatch.
In easy patch, this tags have to be set:
umTagResourceName, umTagPatchIDString, umTagPatchOffset, umTagPatch.

#define umTagNeedVer            0x80000001
#define umTagPatcherName        0x80000002
#define umTagCoderName          0x80000003
#define umTagPatcherIDString    0x80000004
#define umTagPatcherDisableable 0x80000005
#define umTagPatcherPrefs       0x80000006
#define umTagPatcherNoRemove    0x8000001b
#define umTagNeedVersion        0x80000007
#define umTagVersion            0x80000008
#define umTagInit               0x80000009
#define umTagExit               0x8000000a
#define umTagPrefs              0x8000000b
#define umTagPrefsKey           0x8000000c
#define umTagResourceType       0x8000000d
#define umTagResourceName       0x8000000e
#define umTagResourceVersion    0x8000000f
#define umTagResourceID         0x80000010
#define umTagPatchPri           0x80000011
#define umTagPatchPurposeString 0x80000012
#define umTagPatchIDString      0x80000013
#define umTagPatchOffset        0x80000014
#define umTagPatchDisableable   0x80000015
#define umTagPatchFullPatch     0x80000016
#define umTagPatch              0x80000017
#define umTagPatchNotifyDisable 0x80000018
#define umTagPatchInit          0x80000019
#define umTagPatchExit          0x8000001a      /// 1b is last


/**
***  Patchers list
**/

This is for internal usage.

struct psl {
  struct MinNode
              node;                    /// simple node structure
```

```
  BPTR        psl_Segment;              /// BPTR to patcher segment
  struct pfs *psl_Patcher;              /// pointer to patcher structure
};


/**
***  Patch link structure
**/

struct pls {
  struct pls *pls_Next;                 /// pointer to next link
  struct pls *pls_Prev;                 /// pointer to prev link

This is part of standard node structure. Always fill it right!

  APTR        pls_LinkIDString;         /// listview string

Pointer to text which is put in patchlink listview gadget.

  APTR        pls_Prefs;                /// pointer to prefs routine
  LONG        pls_Flags;                /// flags
  LONG        pls_Status;               /// status
  struct mps *pls_List[1000];           /// patches

Put here pointer to patchers in this patchlink. 0 on the end of list.

}

#define plsPrefs       0x00000001

Put if prefs exists.

#define plsDisableable 0x00000002

Put if PatchLink can be disabled.

#define plsDisTemp     0x00000004       /// While removed, previous status

Ignore this.
```

1.14 ToDo list

```
 - improve Ultimate Patch System's structures
 - add more UPS datatypes like devices, trap vectors...
 - improve GUI, and add no-topaz 8 support
 - improve listview in Libraries Overview window
 - add .icon viewer in Patchers Manager window
 - include more patchers in main package
 - remove enforcer read hits(read, not write!)
```

1.15 Distribution rules

Ultimate Patch System is freely distributable. No charge may be made for Ultimate Patch System, other than a nominal copy fee. Ultimate Patch System may not be distributed with a COMMERCIAL or SHAREWARE product without the authors prior consent. Ultimate Patch System must be distributed with all documentation, developers stuff and other files intact and unaltered. Permission is expressly granted to Fred Fish to distribute on his fine collection of disks.

For common user, Ultimate Patch System is totaly free. But, _any_ donations will be gladly accepted(money, self-made stuff, postcards...).

For user who want earn money(in any way) on patchers, have to pay symbolic fee to me. Fee is £5, 10USD or 15Dem.

If user or company wants to include Ultimate Patch System in his production, you should contact me first.

1.16 Ultimate Patch System's history

This is first version of Ultimate Patch, so program history isn't available yet. In fact, there were over fifty versions made, but they were only internally used.

1.17 Credits And Stuff

Ultimate Patch System is written in C, and compiled with SAS/C 6.5, except few routines written in assembler, compiled with PhxAss by Frank Wille.

Graphics User Interface was created using the excellent GadToolsBox v2.0b, from Jaba Development.

Ultimate Patch System uses reqtools.library, which is Copyright Nico François.

Ultimate Patch System has been written in GoldEd by Dietmar Eilert.

Thanks also go to people from Amiga.hr, fidonet conference, which helped me with many answers to my questions. Especially, thanks go to Goran Paulin and Miljenko Vrankovic, because of Enforcer testing.

Also, greets to Alien Dezign(especially Michael Knoke) for making MCP.

1.18 Contacting the author

Goran Mitrovic, author of Ultimate Patch System:

- snail mail :
  Goran Mitrovic

```
     Trg kralja Tomislava 5
     48000 Koprivnica
     Croatia
     Europe

 - internet :
     goran.mitrovic@tvri.fido.hr
     gmit@public.srce.hr (for files)

 - fidonet :
     Goran Mitrovic@2:381/106



Goran Paulin, author of DPatch:

 - snail mail:
     Goran Paulin
     Rade Supica 1
     51000 Rijeka
     Croatia
     Europe

 - internet :
     Goran.Paulin@tvri.fido.hr
     gpaulin@oliver.efri.hr

 - fidonet :
     Goran Paulin@2:381/106
```