FOREWORD to UNI-MAN VERSION 3.0d

Hello, dear MIDI-user,

You have just purchased version 3.0d of the program "UNI-MAN" - "the Most Extensive Generic SYSEX Toolkit". You have now obtained a software-package that is useful for every MIDI-user to have at hand. With this you are able to solve most problems you have with SYSEX.

To give every MIDI-user the opportunity to use this package we have searched for ways to make UNI-MAN affordable for everybody. You, as a new user, know very well that we have succeeded, because for the same price it usually is barely possible to buy a manager or editor for only **one** synthesizer. UNI-MAN is up till now without doubt the most powerful package of Zadok's 'PM-TOOLS Sound Series' software line. ***(If you want more information about Zadok's PM-TOOLS you may contact your dealer or communicate directly with Zadok Products.)***

By the way, if a Midi-user still never has heard of UNI-MAN, then, sorry to say this, he has probably lived on Mars, or some other place where no literature on MIDI was available, and the word 'midi' is only used to denote clothing of medium length. The program therefore doesnt need any introduction any more. You have only to read the many extensive and enthusiastic articles and critics in many international magazines, like Keyboards (German), Sound on Sound (English) and Fachblatt (German). This last one did not hesitate to call UNI-MAN das Universal Genie (The Universal Genius)! To think that this test only dealt with the 2.2 version with a small number of adaptors, and the current version is the much more improved 3.0 version with more than 50(!) Device Adaptors.

L'KE to set R'GHT (do not Misunderstand Uni-Man)

This UNIversal MANager, whose name is shortened as UNI-MAN (the showpiece of Zadoks PM-TOOLS SOUNDSERIES) has always been referred to by the press as a Universal Editor. Alas, this is not exactly the right way to look upon this set of tools. For if it was, we e.g. would have called it UNI-EDI. It even can cause, and does cause misunderstanding about the kind of program UNI-MAN really is.

To begin with, the point of departure with UNI-MAN is **not** the editing of sounds, but the fundamental concept is: "the solving of problems which can occur with working with sounds and/or patches". That e.g. the editing of parameters of synthesizers (and other MIDI-equipment) also falls into this category is a (not unimportant) side-issue.

For example: It is possible to make a dump for every piece of MIDI-gear. With the MIDI-monitor (UNI-MON) you can also see what happens within MIDI. UNI-DUMP is a tool with which all MIDI-data can be sent, received and stored. These kind of tools are usually not found in a patch editor program.

There are, next to these, many more useful options and help-programs in the UNI-MAN toolbox, which do not belong, strictly speaking, to a patch-edit program. In short, this makes UNI-MAN to be a SYSEX-TOOL-KIT no serious MIDI-user can do without.

It is therefore no coincidence that UNI-MAN is used all over the world in by producers, in recording studios, music schools, and conservatories. It is used on several universities as object of study, and at several software-houses for the universal treatment of all of their soundlibraries, at MIDI-workshops and seminars, home studios, etc.

We wish you much joy and success with this UNI-MAN Toolkit version 3.0!

Your ZADOK-TEAM.

TABLE OF CONTENTS

Contents of the Program Diskette

UNI-MAN 3.0d, which is no longer protected with a hardware-key, has undergone several changes. A logical result of this is that the contents of the diskette has been changed. Beginning with UNI-MAN 3.0 the advantages of the program UNI-BOY are taken over by UNI-MAN 3.0. This means, that it e.g. also can be loaded into the switcher of C-lab, together with Notator 3.15.

The contents of the disk UNI-MAN 3.0d is as follows:

UNI-DUMP.ACC	UNI-CON.PRG
UNI-LINK.PRG	UNI-MON.PRG
UNI-DUMP.RSC	UNI-NUM.RSC
UNI-CHAR.PRG	UNI-DUMP.PRG
UNI-MAN.PRG	UNI-NUM.PRG
UNI-MON.RSC	

The folder ‹RESOURCE›:

ALL-MAN.RSC	UNI-CON.RSC
ALL-REST.RSC	UNI-LINK.RSC

The folder ‹UNI-MAN›:

INIT.D7V	INI-LINK.HF
UNI-MAN.PIC	UNI-CON.SC1
UNI-MAN.SC2	INIT.SEQ
STANDARD.DA	STANDARD.NP
UNI-CHAR.SCI	UNI-MAN.SC1
UNI-MAN.SC3	

VERY IMPORTANT: You may **not** change or add something to the contents of the program-diskette, **not eve**n a new desktop or a change in an extension. If you do this, the disk will be damaged and it's working will at least be impaired. (Therefore we have removed the write-protect.)

How do you install UNI-MAN 3.0d?

IMPORTANT: If you use programs as ACCESSORIES, whether you do it from harddisk or from diskette, you must, before starting another program, first click the ACC in the desktop with the masterdisk in drive A, and then going out of it again. Only then you can start the program you wish to use. If you do not do this, the program will badly run or will not run at all. The computer may even hang. So be aware of this!

How to use UNI-DUMP from floppy as an accessory:

1: Put the Masterdisk in drive A
2: Switch the monitor on
3: Switch the computer on

The accessory will be loaded automatically during booting, and can be found (as is the case with any ACC) beneath the leftmost drop-down window.

NOTE: UNI-MON.PRG and UNI-NUM.PRG can also be used as an ACCESSORY. To use them as an ACC you must copy them to another diskette or harddisk. (Remember that it is not permissible to change the MASTER-diskette in any way whatsoever!) Do not forget to copy the '.RSC'-files also.

Use from floppy, without accessories:

1: Put any (formatted) diskette in drive A
2: Switch the monitor on
3: Switch the computer on
4: If the desktop is visible and the floppy-drive is ready (the drive-light is off) exchange this diskette with the UNI-MAN program disk.
5: Start (by double clicking) the program you want to use.

Use from harddisk.

1: Copy the total contents from the UNI-MAN program diskette to drive C (the root directory).
2: Start (by double clicking) the program you want to use. The master-(program)-disk must be **always** is drive A.

When you start the computer from scratch the accessories will be installed automatically. In that case you also have to take care of that the program- disk is in drive A.

As before, activate the ACCESSORIES before running a program.

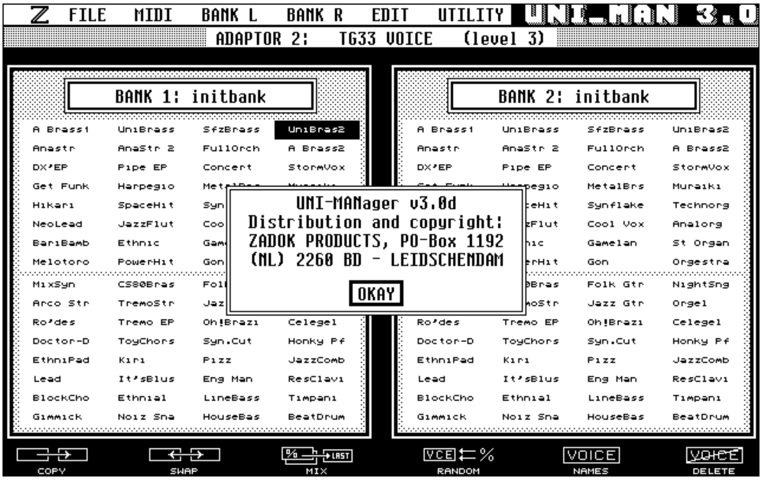
NOTE: If you do not want to use the accessories change the extensions 'ACC' into, say, 'ACX'. (Not on the master disk, only on the harddisk!) The next time you start the computer the accessory will no longer appear in the desk-top.

INTRODUCTION

Since UNI-MAN appeared on the market (now already several years ago) as the first UNIversal MANager, it has caused not only admiration, but also distrust. It was well known that every synthesizer talked its own lingo, and this well-known fact caused much disbelief in a universal approach to SYSEX-editing. Many would welcome **one single program** for managing **every** MIDI-gear, but such a program had to prove itself first.

Since the first release of the UNI-MAN system all participants (programmers, press and, of course, UNI-MAN users) have made suggestions for improvements. The practical use of this program under various circumstances has also contributed much to further improvement. Indeed, not only were some improvements desirable, but others turned out to be an absolute necessity.

Apart from this, powerful updates have provided a better insight and a vast increase in applicability. Especially the implementation of the librarian-functions with extensive random, mix, search and sort-options meant a very important forward step in the usefulness of this program.



ADAPTOR 2: TG33 VOICE (level 3)

BANK 1: initbank

A Brass1	UniBrass	SfzBrass	UniBras2
Anastr	AnaStr 2	FullOrch	A Brass2
DX'EP	Pipe EP	Concert	StormVox
Get Funk	Harpegio	Met	
Hikari	SpaceHit	Syn	
NeoLead	JazzFlut	Coo	
BariBamb	Ethnic	Gam	
Melotoro	PowerHit	Gon	
MixSyn	CS80Bras	Fol	
Arco Str	TremoStr	Jaz	
Ro'des	Tremo EP	Oh!Brazi	Celegel
Doctor-D	ToyChors	Syn.Cut	Honky Pf
EthniPad	Kiri	Pizz	JazzComb
Lead	It'sBlus	Eng Man	ResClavi
BlockCho	Ethnial	LineBass	Timpani
Gimmick	Noiz Sna	HouseBas	BeatDrum

BANK 2: initbank

A Brass1	UniBrass	SfzBrass	UniBras2
Anastr	AnaStr 2	FullOrch	A Brass2
DX'EP	Pipe EP	Concert	StormVox
	pegio	MetalBrs	Muraiki
	eHit	Synflake	Technorg
	zFlut	Cool Vox	Analorg
	ic	Gamelan	St Organ
	erHit	Gon	Orgestra
	0Bras	Folk Gtr	NightSng
	moStr	Jazz Gtr	Orgel
Ro'des	Tremo EP	Oh!Brazi	Celegel
Doctor-D	ToyChors	Syn.Cut	Honky Pf
EthniPad	Kiri	Pizz	JazzComb
Lead	It'sBlus	Eng Man	ResClavi
BlockCho	Ethnial	LineBass	Timpani
Gimmick	Noiz Sna	HouseBas	BeatDrum

UNI-MANager v3.0d
Distribution and copyright:
ZADOK PRODUCTS, PO-Box 1192
(NL) 2260 BD - LEIDSCHENDAM

All this has led to a very mature complete system embodied in version 3.0. A system, that receives much less resistance as the earlier versions. In every area where it can be applied UNI-MAN has conquered a fixed place in daily use by professionals and amateurs alike. Contrary to the beginning we get more and more reactions of satisfied users. From this we have learnt that one of the very strong points of UNI-MAN is its ability deal with equipment where **nobod**y has written software for, or even **ever** will.

The interest in this program is increasing here and abroad. It therefore gives us great pleasure that not so long ago the authoritative magazine 'MUSIK FACHBLATT', right after only some sequencers of Steinber and EMAGIC (then C- Lab), UNI-MAN has been chosen as the best and most important MIDI-software. ***(See Fachblatt no-3, March 91 and for the 'UNIVERSAL GENIUS'-test Fachblatt no-1, january 91).***

UNI-MAN Update 3.0 In General

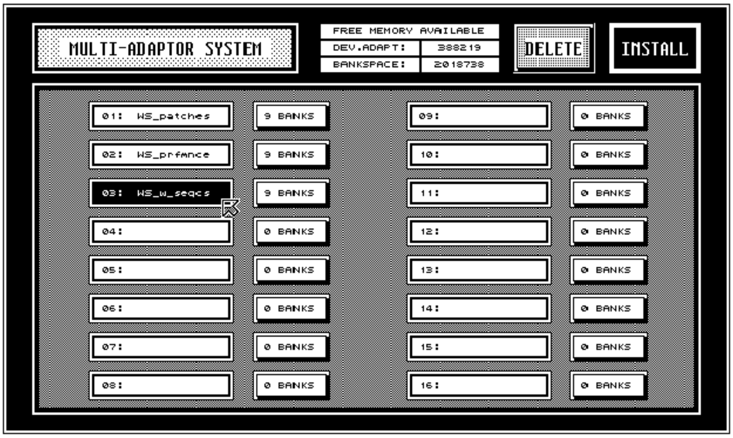
Because the programming of UNI-MAN runs according a certain (planned) line of development, the manual does not need drastic changes during continuing development of the system. Philosophy and concept remain the same, as is the case with ready made device-adaptors that are already in use. This is only possible because from the start the system has be designed with the possibility of growth in mind. The expected (foreseen) changes to version 3.0 are therefore in a large part already described and therefore do not need further explanation. Improvements on the user-level need some explanation and some structural descriptions are unavoidable. Program-technical changes (saving on the use of memory, etc.) will also be briefly mentioned.

The changes to version 3.0, as described in this added manual is the direct result of changes on the synthesizer-market. In the beginning the memory-structure of MIDI-devices was rather simple. This simplicity was very welcome to a universal program, maybe even a condition for the existence of such a program. The user also benefited from this clear and simple concept. But in recent years more and more instruments and other gear with extensive memory-configurations and complicated structures hit the market. These have made us decide to a much more flexible structure of the program.

Up to UNI-MAN version 2.2 it was not possible (nor necessary) to communicate with only a part of the memory. Until then one of the leading principles of UNI-MAN was that all system exclusive data remains unchanged within the program. In practice this meant that a BANK, once received, remained unchanged, after its transfer to UNI-MAN. This makes non-destructive editing of sysex data possible **within** the computer.

Therfor in the request of the BANK RECEIVE ADAPTOR it was also necessary to tell where during sending (COMPUTER ==› MIDI-DEVICE) this bank has to go to. This was not only simple, but also effective. If there was more than one bank (e.g. KAWAI K1), then the same adaptor must be loaded twice, and small changes had to be made in the request. It was therefore necessary to have as many adaptors as there are bank-memories in the MIDI-device. So, every device-adaptor was corresponding with the synthesizer-bank-memory it was derived for.

In this way a comfortable and consistent manager emerged, which (within M.A.S.) could also be used as a database. This approach was possible because Device-Adaptors and banks were very small for this kind of equipment. This approach, however, had the disadvantage that just the banks, and with them the sounds/patches of the corresponding Device-Adaptors, were accessible. The exchange of sounds/patches had to be made by way of loading and saving from disk.



```
FREE MEMORY AVAILABLE
DEV.ADAPT:    388219
BANKSPACE:    2018738

01:  WS_patches    9 BANKS      09:    0 BANKS
02:  WS_prfmnce    9 BANKS      10:    0 BANKS
03:  WS_w_seqcs    9 BANKS      11:    0 BANKS
04:                0 BANKS      12:    0 BANKS
05:                0 BANKS      13:    0 BANKS
06:                0 BANKS      14:    0 BANKS
07:                0 BANKS      15:    0 BANKS
08:                0 BANKS      16:    0 BANKS
```

Recently the size of soundbanks of MIDI-devices has increased more and more. The complexity of the configuration of the memory has increased with it. On top of this it now is also nearly always possible to communicate with the data-cards by way of MIDI. More and more it is possible by way of MIDI to transfer codes, as used for conversion-purposes or program-changes by device-number. Especially when applying it to the more difficult case of several nearly identical instruments, which can only be distinguished by device-number, and which contain more than one bank coördination (BANK SWITCHING).

Lastly, but not least the transmission-speed with large banks is more sensitive than previously. Large amounts of data that can be sent with high (maximum) speed cannot be read as easily by every instrument. Some MIDI-devices **require** a short pause between two dumps that belong together, and with others this is **not** allowed.

These developments have caused structural changes in UNI-MAN. Thanks to the in this additional manual described changes it **now is** possible to influence the RECEIVE- and SEND-PARAMETERS. Therefore there is now for **every** MIDI-device only **one** Device-Adaptor needed per group of parameters.

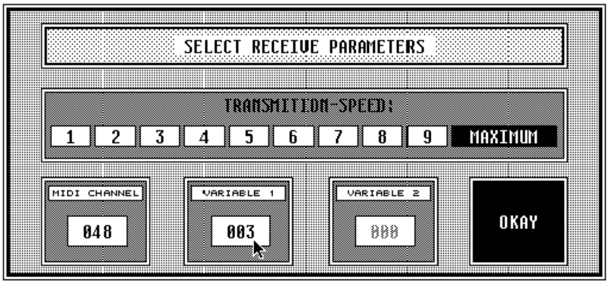
By receiving a bank as well as by sending it you can determine where these have to come from or have to go to. You can also set the MIDI-channel and the device-number. If you want to have control over the SEND PARAMETERS of the bank on disk, the same applies as to SAVE BANK. So you can determine where and with what MIDI-channel UNI-DUMP sends the bank.

Because the header is adapted during RECEIVE, SEND and SAVE BANK, every bank (and therefore every sound/patch) can be loaded in the same Device-Adaptor. This last fact may be, next to the huge saving in memory, the greatest advantage of this method of approach. The old, more usual way of dealing with the data remains nonetheless possible.

A more extensive Device-Adaptor is necessary for version 3.0 of UNI-MAN. This will cause no problems for the user. If you want to convert an old adaptor to the new format it is enough to load it once in UNI-MAN 3.0 and to save it with the correct RECEIVE- and SEND-PARAMETERS to disk.

Receive-, Send- and Save-Bank-Parameters

As soon as one of these options is selected a box will appear in which these parameters can be set.


SELECT RECEIVE PARAMETERS

TRANSMITION-SPEED:

1 2 3 4 5 6 7 8 9 MAXIMUM

MIDI CHANNEL 048

VARIABLE 1 003

VARIABLE 2 000

OKAY

In principle the speed of sending can be set to maximum. Some devices cannot cope with large amounts of data when they receive them at this speed, and therefore demand a lower speed of transfer. Usuallly the MIDI device will show this by displaying CHECKSUM ERROR or other messages like COMMUNICATION ERROR, etc. Sometimes the MIDI-device will be frozen. (TOTAL ERROR) and refuses to respond to any message. If this happens you must switch the device off for 10 seconds and trying again (with a lower transfer speed) is the only correct solution to this problem. The correct setting can only be found by experimentation. If a bulk-dump is composed of several blocks of data, it may be possible that a short pause is required between every block. By choosing a smaller number a pause will be added automatically which corresponds to the lower transfer rate.

Determining the correct speed of transfer is one of the actions that belongs to the design of a Device-Adaptor. UNI-MAN remembers this in the new version 3.0 automatically, so that it is no remaining concern for the user.

In the lower part of the box three parameters can be set which belong to the REQUEST of the bank received, and the HEADER of the bank transmitted. In principle all three parameters have the same range of adjustment and they can be used for any purpose whatever. To accommodate the user the most commonly used names have been given to these. These are MIDI-CHANNEL, VARIABLE-1 and VARIABLE-2. The range runs from 0 to 207 in the decimal number system. We have chosen this range so extensive so that it enables you to send a program-change over every MIDI-channel. An example of this is the KORG DW/EX 8000. Because some MIDI-devices require this, you can set, next to the MIDI-channel **two more variables**. *(See for example KAWAIS K1).*

The parameter MIDI-CHANNEL corresponds in the request with the symbol FC. For the parameters VARIABLE-1 and VARIABLE-2 the symbols FA and FB are used. By putting these symbols in the request the user can determine at what place the corresponding parameter must be set.

For the Programmers of Device-Adaptors

This paragraph is intended especially for those users who are planning to make Device-Adaptors. From UNI-MAN version 3.0 and upwards the program will know (automatically) at what place and with what value the header, of the banks present in the memory, must be adapted. For the setting of the above mentioned parameters the user does not have to deal with stuff like: packet-format, whether or not a checksum is present over the header, the intake of banks as one single unit, or as a set of sounds or patches, etc..

All this will be dealt with automatically and correctly. Because **sounds/patches** correspond with the temporary-buffer of the MIDI-device (or memory location serving as such) these are **not** applicable to these. The speed of transfer is not important either.

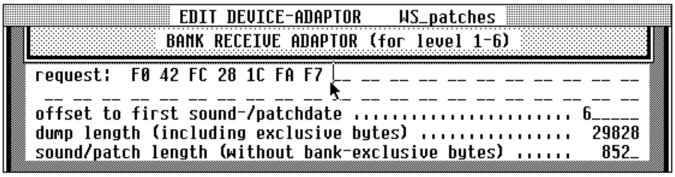
To be perfectly clear about this, so you do not misunderstand, we give an example. Suppose the MIDI-device is the KORG WAVESTATION. First we consider the request of a whole bank of patches: 'F0 42 3M 28 1C 0B F7'

F0	=	System exclusive status byte
42	=	Korg id (entification number)
3M	=	Format id (M = MIDI-channel)
28	=	Wavestation id
1C	=	All patch dump
0B	=	Bank number
F7	=	End of exclusive

As it is well known by most DA-programmers, the MIDI-channel is usually on the third place in the header, for devices which has a MIDI-channel in their SYSEX. We are dealing with this now too. So, as the programmer of the device-adaptor, you here have to fill in FC. The numbers in the request are hexadecimal. They have to be translated to values that the musician can set in his dialog-box. For MIDI-channel 1 (value 0) the user must use the decimal number 48. For 3MH = 30H (M=0) = (3x16+0) = 48. ***(You can check this out too with the UNI-NUM accessory.)*** If you save this 3.0 Device-Adaptor to disk, this value will be remembered and appears at its correct place automatically.

On the sixth spot you will find the bank parameter. (This parameter is recently added to the MIDI-protocol by the I.M.A. - the International MIDI Association). The value 0 corresponds to RAM-1, 1 corresponds to RAM-2, and 3 with the CARD. In the case we are considering here only one parameter is needed to set the memory-address. Here the DA-programmer should type FA.

If more than one request is needed, this adaptation must be done more than once.



```
EDIT DEVICE-ADAPTOR    WS_patches
BANK RECEIVE ADAPTOR (for level 1-6)
request:  F0 42 FC 28 1C FA F7 __ __ __ __ __ __ __ __ __ __ __ __
 __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __
offset to first sound-/patchdate ...................... 6_____
dump length (including exclusive bytes) ................ 29828
sound/patch length (without bank-exclusive bytes) ......  852_
```

VARIOUS UPDATES 1

Other Programming techniques

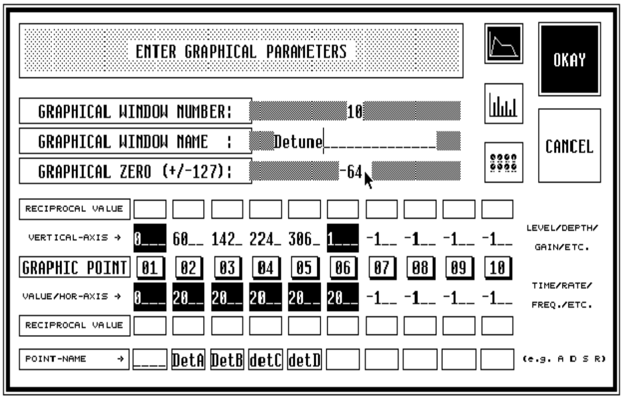
By using the most up to date programming techniques we have succeeded at shortening the program considerably. UNI–MAN is even almost 200 kB smaller than its previous versions. This means that from now on all programs belonging to the UNI–MAN–SYSTEM can be found on one single disk. But this means also a considerable gain in memory in the computer left over for use by the user. The net gain is not so spectacular as on disk, but nevertheless considerable.

But this is not all. The programs are faster. From 66 to 100% faster, dependent on what is activated. This means that most options take half the time they took before. This is very welcome, because universal programs cannot possibly be fast. This is so, because the complicated SCANNER–loops in the program that have to take 1001 possibilities into account. Besides, the user is always struggling with maximum memory barriers, which makes that not much of the memory is available for the program and the operating system.

The best demonstration of this is the option EDIT DEVICE–ADAPTOR. When you use it the first time it takes several seconds of waiting time (so called garbage–collection.) The second, and next times this goes in a flash of a second. The memory is then actualised.

Extensions to the Graphical Possibilities

In our constant strive to completeness, the Line–Graph is extended so that it can also display negative values. It can be, for example, very desirable to have all detuned values displayed. Of from version 3.0 we can set a negative offset in the GRAPHICAL WINDOW, behind the parameter GRAPHICAL ZERO. *(An example can be found in the Device-Adaptor PATCHES of the KORG WAVESTATION).*



GRAPHICAL WINDOW NUMBER: 10
GRAPHICAL WINDOW NAME : Detune
GRAPHICAL ZERO (+/-127): -64

RECIPROCAL VALUE											
VERTICAL-AXIS →	0	60	142	224	306	1	-1	-1	-1	-1	LEVEL/DEPTH/ GAIN/ETC.
GRAPHIC POINT	01	02	03	04	05	06	07	08	09	10	
VALUE/HOR-AXIS →	0	20	20	20	20	20	-1	-1	-1	-1	TIME/RATE/ FREQ./ETC.
RECIPROCAL VALUE											
POINT-NAME →	____	DetA	DetB	detC	detD						(e.g. A D S R)

Extra Computer Keyboard Functions

As well as in the MANAGER, the M.A.S. and in the PARAMETER-EDITOR we have implemented new keyboard functions.

In the MANAGER:

Key:	Function:
[F1]	Copy Sound/patch or bank
[F2]	Swap sound/patch
[F3]	Mix sound/patch or bank
[F4]	Random sound/patch or bank
[F5]	Names sound/patch
[F6]	Delete sound/patch
[F7]	Bank to Librarian
[F8]	Librarian to bank
[F9]	Compare sound/patch or bank
[F10]	Sequencer
[L]	Load functions
[S]	Save functions
[E]	Edit clicked sound/patch
[M]	M.A.S. (See also the [ESCAPE]-key)
[F]	Format disk
[D]	Delete/Show file
[P]	Print bank
[T]	MIDI THRU
[ESC]	M.A.S.

In the M.A.S:

[D]	Delete Device Adaptor
[RETURN]	Install Device-Adaptor/number of banks
[CURSOR] up/down	Select Device-Adaptor
[CURSOR] left/right	Device-Adaptor <=> banks
[-], [+] during banks	Increase/decrease number of banks

In the EDITOR:

[SHIFT] + [J]	Fix Parametergroup Jumppoint
[J]	JUMP TO fixed parametergroup.

UNI-LINK VERSION 3.0

Due to the changes in the Device-Adaptor and the adaptations made, it was also necessary to adapt UNI-LINK. Only UNI-LINK version 3.0 is able to use the 3.0 adaptors correctly and supplies us again with an adaptor which integrates the above mentioned functions.

It may be redundant to do this, but we give another example to illustrate all this. We choose again the KORG WAVESTATION.

This synthesizer has three parts of memorie:

1. RAM-1
2. RAM-2
3. CARD

Each part of memorie contains:

1. PERFORMANCES
2. PATCHES
3. WAVE-SEQUENCES

If we use UNI-MAN 2.2 and previous versions to edit the KORG WAVESTATION completely, we would require nine (3 times 3) Device-Adaptors. Because the adaptors and the banks are also very large, it used to be impossible to load one set of three Device-Adaptors completely into the memory of an Atari 1040ST, let alone to let them function properly. Thanks to the memory savings, coused by the new programming techniques, this **now** is possible with UNI-MAN 3.0.

In the WAVESTATION FOLDER the following adaptors can be found:

1. Multi-Setup-WS
2. Patches-WS
3. Performances-WS
4. Wave-Sequences-WS
5. Total-WS

You can also find a number of REQUESTS (in the folder REQUESTS) and a pair of CONFIGURATIONS (MEGA-2-4.CFG and 1040-BOY.CFG). The requests are given so that you can write a SYSTEM-SETUP and occasional GLOBAL-data to disk. You can do this by using RECEIVE MIDI-FILE, or you can use UNI-DUMP. Another way to do this is by making a small LEVEL-6 adaptor, which you can use to edit these overall-data. You can find examples of these on the original DATA-disks. In practice this need will not be so great, because we are dealing with a setting we only have to do once.

As everybody knows who is very familiar with the Wavestation, a certain way of doing things is required. To work with the Wavestation in a comfortable way it is necessary to use within the PERFORMANCE-BANK (of e.g. RAM-1) only patches and wave-sequences from RAM-1. Otherwise you will not be able to exchange soundbanks, unless you replace the complete memory (180.000 bytes without CARD).

Because it is absolutely useless to replace **only** patches or performances **only**, we have made with UNI-LINK version 3.0 a Device-Adaptor that replaces the **complete** contents of RAM-1, RAM-2 or CARD. Even adaptors that work with **several** banks, as happens here, just have to be made once. This is possible, because UNI-MAN, after feeding it the here described parameters, adapts automatically all separate HEADERS, before they are sent or received.

Despite to the fact that the matching banks are very big, such an adaptor fits within the memory of a 1040 ST computer easely. It is just needed to adapt the MIDI-channel once (**before** sending or receiving the data) to let UNI-MAN automatically make the adaptations (for all performances, patches and wave-sequences) necessary to prevent a memory-overflow.

With some older software-versions of the Korg-WS1, it can happen that the wavestation cannot keep up with the maximum speed of UNI-MANs data-transfer. In this case it is necessary to reduce this speed. If you use the option SEND MIDI FILE this speed of data-transfer can be set as well. In most cases such a reduction in speed of transfer will **not** be necessary.

UNI-MAN 3.0 d AND THE DUMMY-BANK (LEADING)

We are now going to discuss that part of the update to UNI-MAN version 3.0d that is mainly important for those users who use UNI-MAN to make their own Device-Adaptors. Those users who use UNI-MAN just as a Manager, Editor, Data-Filer, Sound-Creator, Librarian, etc., **do not** need to read the following part of this update manual. For those who find MIDI-System-Exclusive interesting it can **do** no harm to **read** this part also. It may even be advisable!

Dummy-Bank Trailing the Exclusive-Bank

We will now discuss the making of a DUMMY-BANK that **trails** the main exclusive-bank. The operations that are discussed here are possible beginning with version 3.0c of UNI-MAN. To make the program more universal drastic changes have been made when compared to the 2.2 and even the 3.0b version.

In the manual of UNI-MAN you will find that much is said about a so-called "DUMMY-BANK". This is a bank that, as most UNI-MAN users knows, contain data that do **not** belong to the sounddata.

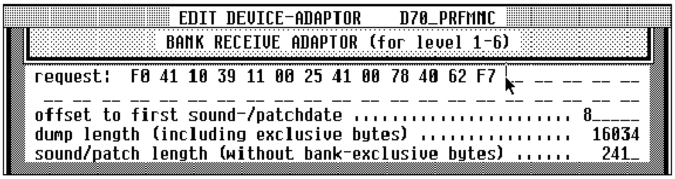
The most obvious (or, rather, most familiar) example is the DUMMY-BANK that **precedes** the main system-exclusive bank. Its function is to add the NAMES OF THE SOUNDS (or remarks about the sounds). This is used very often whenever the MIDI-device itself does not have sound/patch-names. Or if the musician or producer needs a different way of classification. All conditions for the correct functioning of such a bank are explained in more than one way.

It can happen that it is handy or desirable to send extra data **after** the sounddata. This is necessary by synthesizers that, after receiving sounddata, require a WRITE-REQUEST (e.g. the D70 from ROLAND is such an instrument). Just after receiving a write-request the display will be read correctly, and the data will be considered as final. It is important to realise that only the sounds (not the whole bank) have to have such a write-request. The write-request can be different for every sound, since every sound can have its own memory-address.

It may also be necessary to set the instrument in a certain MODE after the reception of exclusive data. In this case all extra data are identical.

How does one make such a Trailing-Dummy-Bank?

To make matters simple we assume that the connected MIDI- device sends **one** bank that contains **all** sounds/patches. As an example we take the D70 from ROLAND. As already stated, this synthesizer requires a write-request of 13 bytes long, after reception of a sound.

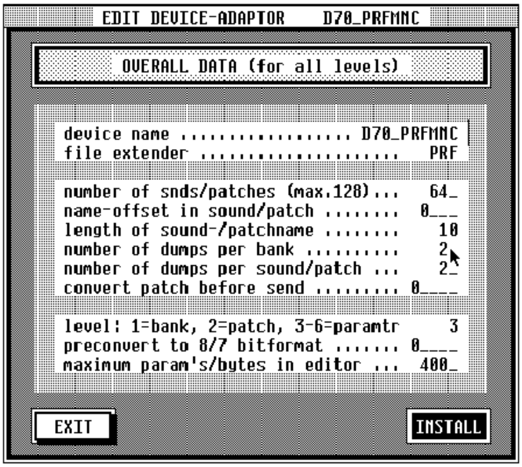


```
            EDIT DEVICE-ADAPTOR    D70_PRFMNC
            BANK RECEIVE ADAPTOR (for level 1-6)
request:  F0 41 10 39 11 00 25 41 00 78 40 62 F7 __ __ __ __ __
 __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __
offset to first sound-/patchdate ...................... 8_____
dump length (including exclusive bytes) ...............  16034
sound/patch length (without bank-exclusive bytes) ......   241_
```

In a bank there are 64 PERFORMANCES present. Because the rest is explained very extensively in the manual, we will only discuss that part pertaining to the DUMMY-BANK.

Overall-Data

To begin with you will have to fed in 2 (in the OVERALL DATA box) under the heading DUMPS PER BANK and NUMBER OF DUMPS PER SOUND/PATCH. Especially in the case whereby the sound/patch data **only** have to have a write-request, many people just want to answer the second question with a 2. This would be correct, if the data would be present in the first bank. In case of an extra DUMMY-BANK UNI-MAN need to get its additional data from the second bank. Therefore **both** questions must be answered with a 2.



```
device name .................. D70_PRFMNC
file extender ....................   PRF

number of snds/patches (max.128)...   64_
name-offset in sound/patch ........  0___
length of sound-/patchname ........    10
number of dumps per bank ..........    2
number of dumps per sound/patch ...    2_
convert patch before send ........ 0____

level: 1=bank, 2=patch, 3-6=paramtr     3
preconvert to 8/7 bitformat ....... 0____
maximum param's/bytes in editor ...  400_
```

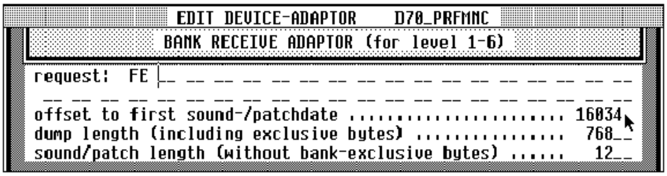
In such a case, we need therefore:

* Two BANK-RECEIVE-ADAPTORS
* Two SOUND/PATCH RECEIVE ADAPTORS
* Two SOUND/PATCH SEND ADAPTORS

In the last Device-Adaptor-boxes (in this case the second one) you must fill-in the data of the DUMMY-BANK.

Bank-Receive-Adaptor

In principle the same rules apply to the making of a DUMMY- BANK that **precedes** the sounddata. The condition is, that the DUMMY-BANK-LENGTH is not allowed to be 0 ,or smaller than 0, and that the OFFSET TO FIRST SOUND/PATCH DATA-BYTE in the bank receive-adaptor is equal to the length of the first bank (or the total sum of the lengths of the banks) which precede(d) it.



```
EDIT DEVICE-ADAPTOR    D70_PRFMNC
BANK RECEIVE ADAPTOR (for level 1-6)
request:  FE |__ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __
 __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __
offset to first sound-/patchdate ........................ 16034
dump length (including exclusive bytes) ................  768__
sound/patch length (without bank-exclusive bytes) ......   12__
```

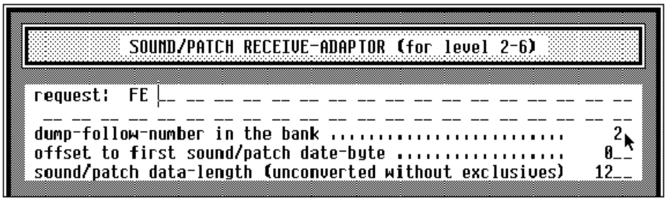
To prevent UNI-MAN waiting for data for the second (dummy-) bank during RECEIVE BANK, the DUMMY-BANK-request FE must be given. *(See chapter 6.2.11 from the original manual).*

Because the original banklength is 16034 Kbyte, the second (dummy-) bank has an OFFSET of 16034. The write-request contains 13 bytes, of which the last one (F7) is generated by UNI-MAN itself. So, the length per data-string (of the request) is just 12 bytes. As DUMP-LENGTH in this matter we must have 64 x 12 = 768, while the SOUND/PATCH-LENGTH is 12, of course.

The DUMP-FOLLOW-NUMBER of the DUMMYBANK constantly is 2, because (in this case) it coms behind the original exclusive-bank.

Sound/Patch-Receive-Adaptor

For the second sound/patch receive adaptor it is also necessary that on the location of the request you must fed in FE. This will prevent the second bank to receive sound/patch-data.



```
SOUND/PATCH RECEIVE-ADAPTOR (for level 2-6)
request:  FE |__ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __
 __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __
dump-follow-number in the bank .........................     2
offset to first sound/patch date-byte ..................     0__
sound/patch data-length (unconverted without exclusives)   12__
```

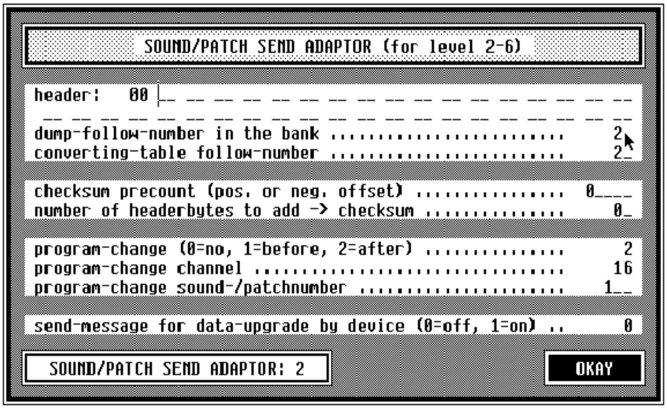
The DUMMY-BANK does not contain a request. Therefore you must put a 0 after OFFSET TO FIRST SOUND/PATCH DATA.

The DUMP LENGTH is again 64 x 12 = 768

The SOUND/PATCH DATA LENGTH is again 12 bytes.

Sound/Patch-Send-Adaptor

In the second sound/patch send adaptor you **must** fed in **00** at the location of the request. This to tell UNI-MAN that there is **no** HEADER. That is, at least for the (easy) solution we have chosen here.



```
SOUND/PATCH SEND ADAPTOR (for level 2-6)

header:   00 |__ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __
 __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __
dump-follow-number in the bank ........................     2
converting-table follow-number ........................     2_

checksum precount (pos. or neg. offset) ................  0____
number of headerbytes to add -> checksum ..............    0_

program-change (0=no, 1=before, 2=after) ..............     2
program-change channel ................................    16
program-change sound-/patchnumber .....................   1__

send-message for data-upgrade by device (0=off, 1=on) ..    0

SOUND/PATCH SEND ADAPTOR: 2                              OKAY
```

A better (more difficult) solution, which is much more universal and makes use of comfortable UNI-MAN-features will be described a little bit further *(see NOTE).*

Because there is no FE the data of this bank **will** be sent. In this way the data will **not** be sent with the bank, but **will** be sent with the sounds. To add a short pause you can add FD. This is only necessary if the MIDI-device requires it. In all other cases one or several extra FDs mean unnecessary delays.

The other data correspond with the SOUND/PATCH RECEIVE ADAPTOR.

Within UNI-MAN many users make a private edit-buffer. To make things easy this can be located at e.g. sound/patch number 1. This is especially useful with MIDI-devices that do not contain a TEMPORARY- (EDIT-) BUFFER. In this case this means that all write-request-data are exactly the same. All sounds will then be sent to the same address. For those who want to bring an already existing DEVICE-ADAPTOR to perfection it is very easy to send every sound/patch to its own address. The sound/patch data-are then localised at the right place (in the device).

NOTE: A better DUMMY-SEND-ADAPTOR, with which you can send to every address, and which always has the correct checksum over the request automatically, looks as follows:

At the place of the HEADER (where you now will find 00) the unchanged part (over which a checksum is unnecessary) can be fed. In the example of the D70 performances this is: 'F0 41 10 39 12'. The DUMMY-BANK then consists of 64 times the address MSB and LSB, followed by the size MSB and LSB. In our example we have chosen the same address for all performances, and therefore this is invariably: '04 58 6A 00 00 01'.

In the second bank-receive-adaptor we install the ROLAND checksum-mode (2). To do so, UNI-MAN will generate a checksum with every sent performance, and shall finish with F7. Using this system will also work correctly if we enter its own address to every performance.

The Dummy–Bank itself

To make your own DUMMY–BANK (without using all kinds of program tricks) you can use UNI–CON, which is part of the UNI–MAN Toolkit. In the above example the DUMMY–BANK contains 64 write–requests, all having 12 or 6 bytes. We start with a DUMMY–BANK that contains all data of the write–request. (12 bytes). At first we must set these 64 x 12 = 768 after each other and save them to disk. Then we have to paste it **behind** the actual bank.

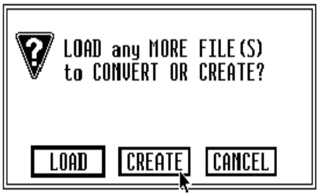
We just need to do this once. UNI–MAN remembers these data (behind the INIT–BANK) and writes them to disk with every bank.

It is also possible to do all this in one go. We do not need to do any other action. What you do will depend on your own preferences. We will consider both possibilities.

UNI–CON 3.0

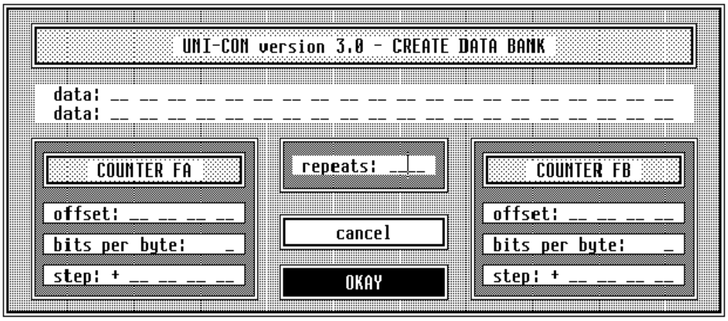
Start the program UNI–CON. Remember, it must be at least version 3.0, because the things we are going to explain here are not possible with earlier versions. Choose the option SPECIALS. As you will probably expect an item–selector will appear. You can choose to load a bank that precedes a DUMMY–BANK, or to make a DUMMY–BANK. In our case we focus our attention to the INIT–BANK as the one shed for the D70. To do everything in one go you must click the D70 bank, and click OKAY. This bank will then be loaded.

In the other case you can choose CANCEL. In both cases the same question: "LOAD any MORE FILE(S) to CONVERT or CREATE?" will appear.


LOAD any MORE FILE(S) to CONVERT OR CREATE?

Now choose CREATE. In the box which will then appear ***(see figure CREATE DATA BANK)*** you can prepare a DUMMY–BANK. The figure suggests that this may be difficult. But the contrary will be the case. We have chosen for this solution, because the data of the MIDI–implementation 'fits' easily. To prevent that the maker of this Device–Adaptor has to make painstaking calculations for all format–data from back and forth, the data can remain in their original form. You must realise that we are still talking about the creation of a Device–Adaptor. What we explain here has nothing to do with using UNI–MAN as a MANAGER and/or EDITOR, etc.

In the two rows at the top you can compose a data string with 40 bytes. Usually you will fill in the data here which, as in our case, will form a write–request. It is also possible that it is a MODE–switch, program–change afterwards, reset– or a display–update–command, or something similar to these.


UNI-CON version 3.0 - CREATE DATA BANK

data: __
data: __

COUNTER FA
offset: __ __ __ __
bits per byte: _
step: + __ __ __ __

repeats: ___

COUNTER FB
offset: __ __ __ __
bits per byte: _
step: + __ __ __ __

At the position where a counter (counting-byte) is present in the built data-string, you have to fill in FA and/or FB. On the bottom left and right hand side you can enter these counters. They operate completely independent of eachother. Usually **one** byte per counter will suffice. There are, however, situations in which two, three, and even four bytes per counter are necessary. This last case can happen when you will have to change an address that is larger than can be put in one, two or three bytes. In the example (which serves as a model for many cases you can encounter in practice) we have the same address in all cases, so that at the position of the address no counter is required.

No counter is needed elsewhere either, so that (in this case) only the data-string and the number of repeats are required. Next to the count-bytes FA and FB four other counters are available: FC, FD, FE and FF. They represent a value of **one byte** and count automatically in an increasing or decreasing way. They are intended for the **initialisation** of MIDI-data values within an exclusive string. Therefore these count-bytes represent the following values:

* FC: from 127 to 1
* FD: from 127 to 0
* FE: from 1 to 127
* FF: from 0 to 127

Each of these counters can be used (within the same exclusive string) up to four times.

Working **with** counters is more difficult. But, as remarked earlier, these data can be taken over directly from the system exclusive implementation (or manual).

Per counter you can determine in what format the counter has to display its data. This will often be the '7 bits Hexadecimal' format. Another format is the 'nibble' (4 bits in one byte). This data-format usually is mentioned in the MIDI-implementation. These concepts are explained extensively in the manual. *(See Chapter 6.3 and 6.5)*

The size of the change per step can be set (below) in maximal 4 bytes. It is important that the values you fill in here correspond with the format you set *(see also UNI-MAN manual chapter 6.3 and 6.5).* The only thing to do is setting the polarity (+ or -) of every counter. The polarity determines (of course) whether the counter counts up or counts down. Clicking the counter polarizes the variable.

At the centre of the box you can set how many times this string has to appear. In most cases this corresponds to the number of sounds/patches, as above.

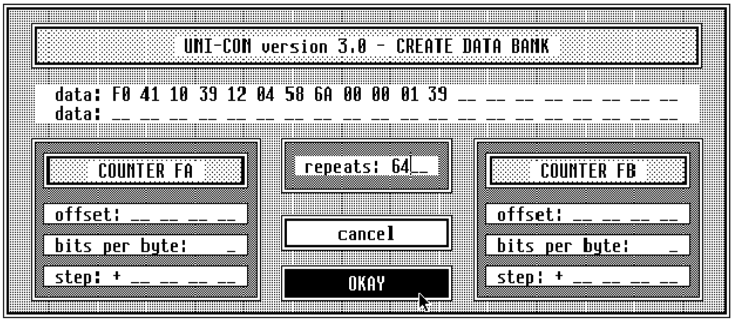
In case there is a bank loaded first, the created DUMMY-BANK will be shown next to the bank already present. In the main screen two files appear which will be glued together in the usual way. In the other cases only the composite data-bank will be in the memory and will be indicated in the main screen as 'CREATED......' . In this last case it is necessary to (after you have written them to disk) to merge them into one INIT-BANK in a second session. It is, of course, also possible to load the SYSTEM-EXCLUSIVE-BANK **after** the creation of the DUMMY-BANK. Pay attention to the right order ***(see chapter 7.6).***

NOTE: In contrast to earlier versions it is now possible to work with BYTE-STRINGS in UNI-CON that are smaller than 17 bytes.

UNI-MAN will see to it that, during saving of a single sound, the corresponding write-request will be saved with the sound. This may be important when using UNI-DUMP. In this way sounds/patches can be prepared for a particular location in the bank. If you do not want this, you can choose a different way of doing things. ***(See below, where we deal with the bank.)***

To prevent overwriting of the sound/patch-addresses of the DUMMY-BANK, UNI-MAN will ignore (during loading of a sound/patch) these extra data. In this way the addressing will always remain intact.

As an extra example we have chosen the data that apply to a DUMMY-BANK for the write-request of the RA-50 ***(see figure WRITE PATCHES RA-50).*** This is a real-time-arranger of ROLAND, which has a fairly extensive MIDI-implementation. The variable FA will be used here to count the patch-number, and FB is the CHECKSUM of the request.


UNI-CON version 3.0 - CREATE DATA BANK

data: F0 41 10 39 12 04 58 6A 00 00 01 39 __ __ __ __ __ __ __ __
data: __

COUNTER FA
offset: __ __ __ __
bits per byte: _
step: + __ __ __ __

repeats: 64__

COUNTER FB
offset: __ __ __ __
bits per byte: _
step: + __ __ __ __

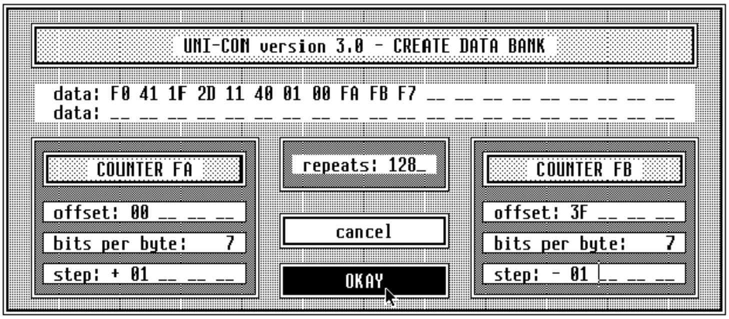
Because the patch-number will be increased by 1, the checksum must be decreased by 1. In this way the checksum will remain constant. The checksum-value 3F is the complementary value to 128 (80H) over the address '40 01 00' and the patchnumber FA. These data can be found in the RA-50 manual too.

For some devices a problem can arise when using UNI-DUMP (in combination with a DUMMY-BANK' which **trails**). Because UNI-MAN saves all data in one file, it can happen that the data are not in the correct order that is required for sending. UNI-DUMP is not capable to send other than EXCLUSIVE-data, but nevertheless, all write-requests in one bank **can** go wrong. Some MIDI-devices fill in all memory-positions with the same (last) sound/patch. If you have this problem, you can deal with it in a number of ways.

Solution 1: Put the Device-Adaptor (with the INIT-BANK belonging to it) through UNI-LINK. This program will remove all DUMMY-BANKS. The INIT-BANK laid out can be sent with UNI-DUMP without problems.

Solution 2: Make a Device-Adaptor that operates on a sound-to-sound basis. This will add a write-request to every sound. Such a bank will be accepted under all circumstances. This bank, too, only has to be made once in the above described way as an INIT-BANK. (see EX 8000 and SPX900/1000). This is a very reliable solution.

Solution 3: Save the bank to disk from UNI-MAN, and remove the DUMMY-BANK with UNI-CON.


UNI-CON version 3.0 - CREATE DATA BANK

data: F0 41 1F 2D 11 40 01 00 FA FB F7 __ __ __ __ __ __ __ __ __
data: __

COUNTER FA		COUNTER FB
offset: 00 __ __ __	repeats: 128_	offset: 3F __ __ __
bits per byte: 7		bits per byte: 7
step: + 01 __ __ __		step: - 01 __ __ __

VARIOUS UPDATES 2

Individual Parameter Handling

An extensive adaptation/extension makes that UNI-MAN cannot only be used to operate on a sound/patch, but also on **a single parameter.** This means **universal real-time facilities**. From now on, instruments that can only be edited on a parameter to parameter basis can be edited with UNI-MAN also.

'JUMP TO' Fixed Parameter

The most important extension in the editor is a much demanded memory-function for certain parameters. It often happens that some parameters can only be edited in a useful way in connection with other parameters. If these parameters are not close together it takes some time and inconvenience to search for the others.

Example: A ROLAND D50 sound consists of one COMMON and two PARTIALS for UPPER and LOWER. To be able to listen to a certain partial very closely it must be possible to switch the other partials on/off. With the partial-mute function (COMMON-parameter) partials can be switched on and off. In this case it is desirable to have the partial-mute function under a 'JUMP-TO' button in such a way that this parameter is exactly in the middle. Only then it is possible to choose the desirable mute-setting, directly after the jump with the mouse (or the [+] and [-] keys), and then to jump back.

By using the **right** mouse-button to click the button 'JUMP-TO', you can fix the (middle) parameter where to jump to, when you click this button with the **left** mouse-button at a later time. You will understand that this comfortable function can be very time-saving (especially when it is used intensively).

Adaptation Edit-Formulas

Another aspect of coherence between different parameters is the option to use formulas *(see paragraph 6.5 EDIT FORMULAS of the manual)* to go back to values of parameters at hand. From version 3.0c on it is possible to give a negative parameter value after READ BYTE, and to use it in the calculation of the actual parameter.

With [SHIFT] + READ BYTE it is possible to sum the value found elsewhere with the actual value. The formula represents this for example as '+Rb -22' (Add the value 22 bytes back to this value).