Programming in System RPL

Eduardo M Kalinowski
Carsten Dominik

Second Edition

Preface to the Second Edition

Back in 1998, when the first edition of this book was released, it proved to be a good tutorial for new learners of System RPL, and also a good rerefence for more experienced programs.

However, there was still room for improvement. And when the HP49G calculator was released, the need for a second edition of this book was even greater, because no document describing all its new features existed.

For these reasons, we have put together this new edition. Those who have used the first edition of this document will find many changes and improvements, and also some 400 additional pages :-). The structure of the book has been totally changed, with the tutorial and reference parts merged. All the text has been revised and corrected. Some chapters were completely rewritten in order to make them easier to understand and more useful. There are also new chapters, describing new HP49 features, and also about things that were not described in the first edition.

We hope this book is a valuable resource for those that already knew System RPL on the HP48 and wanted more information on the new HP49 features, and for those that want to start learning System RPL in order to discover more of the power of the HP49.

April 24, 2002

Eduardo de Mattos Kalinowski
Carsten Dominik

Preface to the First Edition

The programming features of the HP48 graphical calculator are very powerful. They allow you to do virtually anything. However, the documented programming functions, that are directly accessible to the user (the user language), is not everything the calculator can do.

There is another language: the System language. The User language is a subset of the System one, with just some commands and just a fraction of its power. However, the System language is not well documented. The existing documents on that subject are turned to someone who already knows it; they are just listings of the commands with some brief descriptions. Once you already know the language, even the brief descriptions can be left out, and those documents are really a very good source of information. But how does one *learn* System RPL?

The purpose of this book is exactly that: to be a way for someone who has already learned User RPL (if you have not yet, learn it before, then come back to this), and wants to learn the *real* power of the calculator.

July 12, 1998 Eduardo de Mattos Kalinowski

Acknowledgments

This work could not have been accomplished without the help of many people. Firstly, we would like to thank Mika Heiskanen for his entry point list by subject. That document was an important source of information. His JAZZ library with its disassembler and other functions was also an indispensable gateway to the HP48 ROM code.

We would also like to thank the ACO team, for developing the HP49, and for providing us with additional information for writing some of the chapters in this book.

Our thanks go also to Wolfgang Rautenberg, who reviewed the first edition throughly and gave many suggestions for this present edition.

Many other people helped during the two editions of the book:

Al Arduengo	Joe Horn	Alberto Zamora Oyace
Jean-Yves Avenard	Werner Huysegoms	Bernard Parisse
Jurjen N.E. Bos	David Kastrup	Richard Pascal
Carlos Bourlot	Dan Kirkland	James M Prange
Sune Bredahl	Piotr Kowalewski	Thomas Rast
Jonathan Busby	Daniel Lidström	Eric Rechlin
Cyrille de Brébisson	Andreas Matthias	Melissa Reid
Stefan Ehlen	Andreas Möller	Ricardo Blasco Serrano
Len Fellman	Denis Martinez	Gerald Squelart
Peter Geelhoed	Hakim Mazouz	Pierre Tardy
Christoph Gießelink	Christian Meland	Jernej Zajc
Raymond Hellstern	John H Meyers	Adam Zwierko
Jordi Hidalgo	Heiko Oberdiek	

If we forgot someone, please forgive us, and be sure we are grateful anyway.

Disclaimer

The latest version of this document and its errata can be found at the homepage `http://move.to/hpkb`.

Short Contents

Contents

Chapter 1
Introduction

If you know how to create programs in User RPL (if you do not, you should learn it before you continue reading this book), then you only know part of what the HP49G calculator can do. The System RPL programming language gives you power to do many things which you could not even imagine. For example, in System RPL you can handle all object types available. User RPL only gives access to some of them. Or you can do math with 15-digit accuracy, use arrays with non-numeric elements, and much more. System RPL can also be used to do the same things as a User RPL program would do, but much faster.

But before we start talking of System RPL, let us go back to User RPL to explain how it really works. We know you are anxious to start with the big thing right now, but the following information is important for a good understanding of System RPL.

HP49 programs (both User and System) are not stored internally using the names of the commands. Only the addresses of the objects are stored. Each of these addresses takes only 2.5 bytes (or more, if the address is a rompointer or flashpointer). When a program is run, the only thing that is actually done is a kind of "gosub" to that address. This way of storing programs serves two purposes. 2.5 bytes is less than the name of most commands, so the program needs less memory. And execution of the program is much faster since during execution, looking up the addresses of names is no longer necessary.

Most of the times, the address points to another program with more jumps to other programs with more jumps, and so on... The calculator keeps track of holding the address to which jump back, and you can have as many jumps as necessary without worrying about it. When the called program ends, you return to where you were before. Of course, the jumps must end somewhere, either in a program written in machine language or in an object that just puts itself in the stack (numbers, strings, etc). This is quite similar to the concept of calling a function or sub-routine in high-level languages.

But if the programs are just addresses, how can they be edited? The answer is that the calculator has a table of the User commands' names and their corresponding addresses. So, when you put a User RPL program in the stack, the HP searches the table to get the name of the commands corresponding to the addresses stored in memory, and then displays the program in a readable form. You can then edit it, and after the edition is done the table is searched again for the addresses of the commands entered, and only these addresses are stored in memory. This is why it takes a long time to edit a long User RPL program, but it is also what makes them fast to run.

This all works as long as all the commands have names. Guess what? There are over four thousand commands without names. This is one of the distinctions between User and System RPL. User RPL, the language described in the manual (the « » language), can only access the named commands. (Actually, it can access the unnamed commands via the commands `SYSEVAL`, `LIBEVAL` and `FLASHEVAL`, as long as you know the address of the command. But this is not efficient (except for an occasional use)). System RPL can access all commands.

Because of that, System RPL programs cannot be edited directly. Special tools are needed for that. In Appendix A you will find information about the available tools for writing System RPL programs. Fortunately, all you need is built-in in the calculator, or is in libraries that can be downloaded to the calculator.

Programming in System RPL is more powerful and much faster, because it does no error checking. In System RPL, the programmer must be sure that no error occurs, otherwise a crash might happen. For example, if a command requires two arguments in the stack and they are not there, or if they are not of the type the function requires, a warmstart or even a memory loss could happen. Naturally, there are commands for checking if there are enough arguments, for their types, and for other possible error conditions. The difference is that you probably just need to check if all arguments are present once, when the program starts. You do not need to repeat the check later. In User RPL, every single command has error checking, so tests are done unnecessarily, slowing the program.

At this point, you might be wondering, "if the commands do not have names, how can you program in System RPL?" As said before, all commands have addresses, so you can call the address directly, using a structure like `PTR <address>`, and whatever is at that address will be executed. But there is an easier way.

The commands *have* names. The names simply are not stored in the HP49 in the same way the the names of User commands are. But the HP design team has given them names, and they are stored in the tools for creating System RPL programs. You write a program using those names, and then the System RPL compiler searches the names in the tables, and converts them to addresses. This is called compiling or assembling. Some tools can also do the opposite: convert the addresses into command names. This is called decompiling or disassembling.

Some of the commands are classified as "supported": they are guaranteed to stay at the same memory location in all ROM versions of the calculator, i.e., their address are not going to change, so programmers can use them safely. (Note that this does not mean they will be in the same address in different calculators, such as the HP48 and HP49.) But there are also commands that are classified as "unsupported". For these, there is not guarantee that they will stay at the same address in different ROM versions.

Unsupported commands are not listed in the tables of compilers, so you cannot enter their names and expect to have their address in the resulting program. You have to either call them directly by their address, or name the command yourself. In the entries listings, the names of unsupported entries will be *enclosed in single parenthesis*, like `(CURSOR@@)`.

Note that all unsupported entries listed in this book are, however, stable. It has been indicated by the HP design team that all HP49G addresses in the ranges 025ECh–0B3C7h and 25565h–40000h will very likely not change, even the unsupported commands in these ranges.

Actually, there are three kinds of entries: the description above dealt mainly with normal 2.5-byte addresses, which point directly to some ROM address. Most entries are of this kind. But there are also rompointer and flashpointer entries. Rompointers point to commands inside a library. Their names start with ~. Flashpointers, which only exist in the HP49, point to sub-routines inside the flash memory. Their names start with ^. Appendix A will describe what is necessary in order to use each kind of entries with HP49 compiler.

1.1 Your First System RPL Program

Let us create a very simple System RPL program, and explain it in detail. The program will calculate the area of a circle, given the radius in the stack. See Appendix A for information on how to compile it. If you downloaded the examples file, you will find it with the name `first`.

```
::
  CK1NOLASTWD   (check if there is an argument)
  CK&DISPATCH1  (check if it is a real number)
  BINT1 ::      (if it is)
    %2 %^       (square the radius)
    %PI         (put PI in the stack)
    %*          (and multiply)
  ;
;
```

Before we start analyzing it, it is important to note that System RPL is case-sensitive, so `pi` is different from `PI`, which is different from `pI`. Be careful when typing. Also, as you might have guessed, everything between ( )'s is considered a comment. Lines that have a * in the first column are also comments.

The first line contains the start of secondary (i.e., program) marker, `::` (called `DOCOL`). The end marker is `;` (`SEMI`).

Following, there is the command `CK1NOLASTWD`. This command checks if there is one argument in the stack, and if there is not, generates a "Too Few Arguments" error. The next command, `CK&DISPATCH0`, checks the argument type and allows the programmer to do different things for different argument types. Our program only supports one argument type: real numbers (represented here by `BINT1`, or the number one as a system binary — see Chapter 2). If any other argument type is entered, a "Bad Argument Type" error will be produced. Argument checking is described in detail in Chapter 29.

After that, there is the code to execute if the argument is a real number. Note that the code is between `::` and `;`. This is because only one object is expected after the argument type. Here, this one object is a secondary (subprogram), one kind of composite object: it is only one object, but with other objects inside it. So if we want to evaluate more than one object, they must be included in a secondary. This is similar to enclosing several statments between braces in C or between `begin` and `end` in Pascal.

The rest of the program is very simple. The number two is put in the stack, and the radius (entered by the user) is raised to that power.

Finally , π is put in the stack, and the squared radius is multiplied by it. The stack now contains the area.

This program is 25 bytes long, as opposed to the 20 of the User RPL program `« SQ p * ->NUM »`. However, the User RPL version took 0.0156 seconds to calculate (with radius 1). The System RPL took only 0.0019 seconds. Note that, even if this System RPL program is longer than an equivalent in User RPL, this generally does not happen.

1.2 About the Entries Listing

In the following chapters, the stack diagrams use codes to represent each object type. Here is a list of such codes:

Abbreviation	Meaning
`ob`	any object
`1...n`	n objects
`#`	binary integer (BINT)
`HXS`	hex string (User binary integer)
`CHR`	character
`$`	character string
`T`	`TRUE`
`F`	`FALSE`
`flag`	`TRUE` or `FALSE`
`%`	real number
`%%`	extended real number
`%C`	complex number
`%%C`	extended complex number
`z, Z, ZINT`	infinite precision integer
`N`	positive infinite precision integer
`s, symb`	symbolic
`u, unit`	unit object
`{}`	list
`A, []`	array
`V, []`	vector
`M, [[]]`	matrix
`P`	polynom, a list of `Q`s
`Q`	`ZINT` or `P`
`meta, ob1..obn #n`	meta object
`grob`	graphical object
`menu`	menu: a program or a list
`sign`	sign table

UserRPL stack diagrams use some additional abbreviations:

Abbreviation	Meaning
`x, y`	real, list, generic UserRPL object
`c, (,)`	complex number
`#`	hex string (User binary integer)

Abbreviation	**Meaning**
θ	angle (a real number)
`m, n`	integer (`ZINT` or real)
`date`	date in `DD.MMYYYY` or `MM.DDYYYY` format
`name`	global name
`prog, prg`	program
`f, func`	function
`F`	integral of `f`

Part I

HP49 Objects

Chapter 2
Binary Integers (BINTS)

Binary integers are the objects you will use most often. They are not the user-level binary integers (those that you enter starting with #); these are actually hexadecimal strings, described in Chapter 7. These system-level binary integers (called bints for short) are objects which are not so easily accessible to the user. If you happen to have one in the stack, they show like ¤ 10h. Try this if you are using a HP49: enter the following number in the stack (triple check if it is right): #3316Bh. Now, type SYSEVAL and press ENTER. You should get ¤ 10h in the stack, or perhaps ¤ 16d (or even something else), depending on the number base you are using. Internally, they are always in hexadecimal mode. With the HP49 and library 256 attached (see Appendix A), you can use the commands R~SB and SB~B to convert reals and user-level binary numbers into bints, respectively, and vice-versa.

Bints are the objects you will use most often because most commands that require a numeric argument need it in the form of a binary integer, as opposed to the real numbers needed by user functions. So, they should be easy to create. And, indeed, they are. You can put one in stack just by entering it on your program (in decimal form). But that is not recommended at all times, because you can also put a real number in stack by just entering it in the same way (we will see later how to differ one from another). So, it is a good idea to use the following structure: # <hex>. This way, you can be sure you will get a binary number, and your code is clearer. Unfortunately (or fortunatelly), you must use hexadecimal representation.

In the HP49G ROM, there are many "built-in" binary numbers. You can put one of these in the stack by just calling its address. Since almost all of them are supported, to get #6h in the stack, you just use the word BINT6. The main advantage is that if you enter # 6, it takes five bytes. The word BINT6, as all other commands (except rompointer and flashpointer commands), take only 2.5 bytes. Some words put two or even three bints in the stack, so the savings are even greater. Following, there is a list of built-in bints.

The four basic operations with bints are #+, #-, #* and #/. There are also many others, which are listed below.

Here is an example of program that just put three bints in the stack, using the three methods:

```
::
  13     (13d or Dh)
  # D    (the same, using preferred method)
  BINT13 (in this case, this method is shorter)
;
```

2.1 Reference

2.1.1 Built-in Binary Integers

Addr.	Name	Description
33107	BINT0	0d 0h aka: ZERO, any
33111	BINT1	1d 1h aka: ONE, real, MEMERR
3311B	BINT2	2d 2h aka: TWO, cmp
33125	BINT3	3d 3h aka: THREE, str
3312F	BINT4	4d 4h aka: FOUR, arry
33139	BINT5	5d 5h aka: FIVE, list
33143	BINT6	6d 6h aka: SIX, id, idnt
3314D	BINT7	7d 7h aka: SEVEN, lam
33157	BINT8	8d 8h aka: EIGHT, seco
33161	BINT9	9d 9h aka: NINE, symb
3316B	BINT10	10d Ah aka: TEN, sym

Addr.	Name	Description
33175	BINT11	11d Bh aka: ELEVEN, hxs
3317F	BINT12	12d Ch aka: TWELVE, grob
33189	BINT13	13d Dh aka: TAGGED, THIRTEEN
33193	BINT14	14d Eh aka: EXT, FOURTEEN, unitob
3319D	BINT15	15d Fh aka: FIFTEEN, rompointer
331A7	BINT16	16d 10h aka: REALOB, SIXTEEN
331B1	BINT17	17d 11h aka: SEVENTEEN, 2REAL, REALREAL
331BB	BINT18	18d 12h aka: EIGHTEEN
331C5	BINT19	19d 13h aka: NINETEEN
331CF	BINT20	20d 14h aka: TWENTY
331D9	BINT21	21d 15h aka: TWENTYONE
331E3	BINT22	22d 16h aka: TWENTYTWO
331ED	BINT23	23d 17h aka: TWENTYTHREE
331F7	BINT24	24d 18h aka: TWENTYFOUR
33201	BINT25	25d 19h aka: TWENTYFIVE
3320B	BINT26	26d 1Ah aka: REALSYM, TWENTYSIX
33215	BINT27	27d 1Bh aka: TWENTYSEVEN
3321F	BINT28	28d 1Ch aka: TWENTYEIGHT
33229	BINT29	29d 1Dh aka: TWENTYNINE

Addr.	Name	Description
33233	BINT30	30d 1Eh aka: REALEXT, THIRTY
3323D	BINT31	31d 1Fh aka: THIRTYONE
33247	BINT32	32d 20h aka: THIRTYTWO
33251	BINT33	33d 21h aka: THIRTYTHREE
3325B	BINT34	34d 22h aka: THIRTYFOUR
33265	BINT35	35d 23h aka: THIRTYFIVE
3326F	BINT36	36d 24h aka: TTHIRTYSIX
33279	BINT37	37d 25h aka: THIRTYSEVEN
33283	BINT38	38d 26h aka: THIRTYEIGHT
3328D	BINT39	39d 27h aka: THIRTYNINE
33297	BINT40	40d 28h aka: FORTY, FOURTY
332A1	BINT41	41d 29h aka: FORTYONE
332AB	BINT42	42d 2Ah aka: FORTYTWO
332B5	BINT43	43d 2Bh aka: FORTYTHREE
332BF	BINT44	44d 2Ch aka: FORTYFOUR
332C9	BINT45	45d 2Dh aka: FORTYFIVE
332D3	BINT46	46d 2Eh aka: FORTYSIX
332DD	BINT47	47d 2Fh aka: FORTYSEVEN
332E7	BINT48	48d 30h aka: FORTYEIGHT

Addr.	Name	Description
332F1	BINT49	49d 31h aka: FORTYNINE
332FB	BINT50	50d 32h aka: FIFTY
33305	BINT51	51d 33h aka: FIFTYONE
3330F	BINT52	52d 34h aka: FIFTYTWO
33319	BINT53	53d 35h aka: FIFTYTHREE, STRLIST, THREEFIVE
33323	BINT54	54d 36h aka: FIFTYFOUR
3332D	BINT55	55d 37h aka: FIFTYFIVE
33337	BINT56	56d 38h aka: FIFTYSIX
33341	BINT57	57d 39h aka: FIFTYSEVEN
3334B	BINT58	58d 3Ah aka: FIFTYEIGHT
33355	BINT59	59d 3Bh aka: FIFTYNINE
3335F	BINT60	60d 3Ch aka: SIXTY
33369	BINT61	61d 3Dh aka: SIXTYONE
33373	BINT62	62d 3Eh aka: SIXTYTWO
3337D	BINT63	63d 3Fh aka: SIXTYTHREE
33387	BINT64	64d 40h aka: BINT40h, SIXTYFOUR, YHI
33391	BINT65	65d 41h aka: ARRYREAL
3339B	BINT66	66d 42h aka: FOURTWO
333A5	BINT67	67d 43h aka: FOURTHREE

Addr.	Name	Description
333AF	BINT68	68d 44h aka: SIXTYEIGHT
333B9	BINT69	69d 45h aka: FOURFIVE
333C3	BINT70	70d 46h aka: SEVENTY
333CD	BINT71	71d 47h
333D7	BINT72	72d 48h
333E1	BINT73	73d 49h
333EB	BINT74	74d 4Ah aka: SEVENTYFOUR
333F5	BINT75	75d 4Bh
333FF	BINT76	76d 4Ch
33409	BINT77	77d 4Dh
33413	BINT78	78d 4Eh
3341D	BINT79	79d 4Fh aka: SEVENTYNINE
33427	BINT80	80d 50h aka: EIGHTY
33431	BINT81	81d 51h aka: EIGHTYONE, LISTREAL
3343B	BINT82	82d 52h aka: LISTCMP
33445	BINT83	83d 53h aka: FIVETHREE
3344F	BINT84	84d 54h aka: FIVEFOUR
33459	BINT85	85d 55h aka: 2LIST
33463	BINT86	86d 56h aka: FIVESIX
3346D	BINT87	87d 57h aka: LISTLAM
33477	BINT88	88d 58h
33481	BINT89	89d 59h
3348B	BINT90	90d 5Ah
33495	BINT91	91d 5Bh aka: BINT_91d
3349F	BINT92	92d 5Ch

Addr.	Name	Description
334A9	BINT93	93d 5Dh
334B3	BINT94	94d 5Eh
334BD	BINT95	95d 5Fh
334C7	BINT96	96d 60h aka: BINT_96d
334D1	BINT97	97d 61h aka: IDREAL
334DB	BINT98	98d 62h
334E5	BINT99	99d 63h
334EF	BINT100	100d 64h aka: ONEHUNDRED
334F9	BINT101	101d 65h
33503	BINT102	102d 66h
3350D	BINT103	103d 67h
33517	BINT104	104d 68h
33521	BINT105	105d 69h
3352B	BINT106	106d 6Ah
33535	BINT107	107d 6Bh
3353F	BINT108	108d 6Ch
33549	BINT109	109d 6Dh
33553	BINT110	110d 6Eh
3355D	BINT111	111d 6Fh aka: char
33567	BINT112	112d 70h
33571	BINT113	113d 71h
3357B	BINT114	114d 72h
33585	BINT115	115d 73h aka: BINT_115d
3358F	BINT116	116d 74h aka: BINT_116d
33599	BINT117	117d 75h
335A3	BINT118	118d 76h
335AD	BINT119	119d 77h
335B7	BINT120	120d 78h
335C1	BINT121	121d 79h
335CB	BINT122	122d 7Ah aka: BINT_122d
335D5	BINT123	123d 7Bh
335DF	BINT124	124d 7Ch

Addr.	Name	Description
335E9	BINT125	125d 7Dh
335F3	BINT126	126d 7Eh
335FD	BINT127	127d 7Fh
33607	BINT128	128d 80h aka: BINT80h
33611	BINT129	129d 81h
3361B	BINT130	130d 82h aka: BINT130d, BINT_130d, XHI-1
33625	BINT131	131d 83h aka: BINT_131d, BINT131d, XHI
3362F	(#8F)	143d 8Fh
33639	SYMBREAL	145d 91h
33643	(#92)	146d 92h
3364D	(#9A)	154d 9Ah
33657	SYMBUNIT	158d 9Eh
3EAFB	(#9F)	159d 9Fh
3366B	SYMOB	160d A0h
33675	SYMREAL	161d A1h
3367F	(#A2)	162d A2h
39E6B	(#A4)	164d A4h
33689	(#A5)	165d A5h
33693	SYMID	166d A6h
3369D	SYMLAM	167d A7h
336A7	(#A9)	169d A9h
336B1	SYMSYM	170d AAh
336BB	SYMEXT	174d AEh
3BD4C	(#AF)	175d AFh
336C5	(HXSREAL)	177d B1h
38275	(#BB)	187d BBh
336CF	(2HXS)	187d BBh
336D9	BINTC0h	192d C0h
3E7DA	(#C8)	200d C8h
336E3	2GROB	204d CCh
3BD65	(#CF)	207d CFh
336ED	TAGGEDANY	208d D0h
336F7	EXTREAL	225d E1h
33701	EXTSYM	234d EAh
3370B	2EXT	238d EEh
33715	ROMPANY	240d F0h

Addr.	Name	Description
3371F	BINT253	253d FDh
33729	BINT255d	255d FFh
33733	REALOBOB	256d 100h
3373D	#_102	258d 102h
33747	#SyntaxErr	262d 106h
33751	(BINT_263d)	263d 107h
3375B	(#110)	272d 110h
33765	3REAL	273d 111h
3E17B	(#111)	273d 111h
3376F	(Err#Kill)	291d 123h
33779	(Err#NoLstStk)	292d 124h
2777E	(#12F)	303d 12Fh
33783	(#NoRoomForSt)	305d 131h
3378D	(#132)	306d 132h
33797	(REALSTRSTR)	307d 133h
337A1	(#134)	308d 134h
337AB	(#135)	309d 135h
337B5	(#136)	310d 136h
337BF	(#137)	311d 137h
337C9	(#138)	312d 138h
337D3	(#139)	313d 139h
337DD	(#13A)	314d 13Ah
337E7	(#13B)	315d 13Bh
337F1	(#13D)	317d 13Dh
337FB	(#13E)	318d 13Eh
33805	INTEGER337	337d 151h
3380F	(#200)	512d 200h
33819	(Err#NoLstArg)	517d 205h
3A1C2	(#304)	772d 304h
33823	STRREALREAL	785d 311h
3B9FA	(#313)	787d 313h
3C11E	(#410)	1040d 410h
3B928	(#411)	1041d 411h
3382D	(ARRYREALREAL)	1041d 411h
33837	(#412)	1042d 412h
3BA2D	(#414)	1044d 414h
3B93D	(#415)	1045d 415h
33841	(#444)	1092d 444h
3C10F	(#450)	1104d 450h

Addr.	Name	Description
3B952	(#451)	1105d 451h
3384B	(ARRYLISTREAL)	1105d 451h
33855	(#452)	1106d 452h
3BA18	(#454)	1108d 454h
3B913	(#455)	1109d 455h
3A12D	(#4FF)	1279d 4FFh
3385F	(#510)	1296d 510h
33869	(#511)	1297d 511h
3BA09	(#515)	1301d 515h
33873	(#550)	1360d 550h
277F6	(#605)	1541d 605h
27800	(#606)	1542d 606h
2780A	(#607)	1543d 607h
27814	(#608)	1544d 608h
2781E	(#609)	1545d 609h
27828	(#60A)	1546d 60Ah
27832	(#60B)	1547d 60Bh
2783C	(#60C)	1548d 60Ch
27846	(#60D)	1549d 60Dh
2768E	(#60E)	1550d 60Eh
27698	(#60F)	1551d 60Fh
3387D	(IDREALOB)	1552d 610h
276AC	(#611)	1553d 611h
276B6	(#612)	1554d 612h
276C0	(#613)	1555d 613h
276CA	(#614)	1556d 614h
276D4	(#615)	1557d 615h
276DE	(#616)	1558d 616h
276E8	(#617)	1559d 617h
27792	(#618)	1560d 618h
2779C	(#619)	1561d 619h
277A6	(#61A)	1562d 61Ah
277B0	(#61B)	1563d 61Bh
277BA	(#61C)	1564d 61Ch
277C4	(#61D)	1565d 61Dh
277CE	(#61E)	1566d 61Eh
277D8	(#61F)	1567d 61Fh
277E2	(#620)	1568d 620h
277EC	(#621)	1569d 621h

Addr.	Name	Description
276F2	(#622)	1570d 622h
276FC	(#623)	1571d 623h
27706	(#624)	1572d 624h
27710	(#628)	1576d 628h
2771A	(#629)	1577d 629h
27724	(#62A)	1578d 62Ah
2772E	(#62B)	1579d 62Bh
27738	(#62C)	1580d 62Ch
27742	(#62D)	1581d 62Dh
27788	(#62E)	1582d 62Eh
33887	(IDLISTOB)	1616d 650h
33891	(#700)	1792d 700h
3C17A	(#710)	1808d 710h
3C16B	(#750)	1872d 750h
08DF7	(#7FF)	2047d 7FFh
27878	(#800)	2048d 800h
3B976	(#822)	2082d 822h
3C83C	(#82C)	2092d 82Ch
3B967	(#855)	2133d 855h
3C81E	(#85C)	2140d 85Ch
3389B	(#861)	2145d 861h
338A5	(#862)	2146d 862h
338AF	(#865)	2149d 865h
338B9	(#86E)	2158d 86Eh
3E7FF	(#8F1)	2289d 8F1h
3E759	(#8FD)	2301d 8FDh
3E7E9	(#9F1)	2545d 9F1h
3E743	(#9FD)	2557d 9FDh
2774C	(#A01)	2561d A01h
27756	(#A02)	2562d A02h
27882	Attn#	2563d A03h
338C3	ATTNERR	2563d A03h
27760	(#A04)	2564d A04h
2776A	(#A05)	2565d A05h
27774	(#A06)	2566d A06h
338CD	(#A11)	2577d A11h
338D7	(#A12)	2578d A12h
338E1	(#A1A)	2586d A1Ah
338EB	(#A21)	2593d A21h

Addr.	Name	Description
338F5	(#A22)	2594d A22h
338FF	(#A2A)	2602d A2Ah
33909	(#A61)	2657d A61h
33913	(#A62)	2658d A62h
3391D	(#A65)	2661d A65h
33927	(#A6E)	2670d A6Eh
33931	(#AA1)	2721d AA1h
3393B	(#AA2)	2722d AA2h
33945	(#AAA)	2730d AAAh
3394F	(#C06)	3078d C06h
33959	(#C07)	3079d C07h
33963	(#C08)	3080d C08h
3396D	Connecting	3082d C0Ah
33977	(#C0B)	3083d C0Bh
3C800	(#C2C)	3116d C2Ch
3C7E2	(#C5C)	3164d C5Ch
3B904	(#C22)	3106d C22h
3B8F5	(#C55)	3157d C55h
33981	#CAlarmErr	3583d DFFh
3398B	EXTOBOB	3584d E00h
3C8D0	(#2111)	8465d 2111h
03FEF	(#2614)	9748d 2614h
03FF9	(#2686)	9862d 2686h
03F8B	TYPEREAL	10547d 2933h
03FDB	(TYPEEREL)	10581d 2955h
03FA9	TYPEIDNT	10568d 2948h
03F95	(TYPECMP)	10615d 2977h
03F9F	(TYPELIST)	10868d 2A74h
20D6F	(TYPERRP)	10902d 2A96h
03FBD	(TYPESYMB)	10936d 2AB8h
03FE5	(TYPEEXT)	10970d 2ADAh
03FA9	(#2E48)	11848d 2E48h
03FD1	(TYPELAM)	11885d 2E6Dh
3C8DF	(#5B11)	23313d 5B11h
3D50D	(#A110)	41232d A110h
3D52B	(#A1A0)	41376 A1A0h
3D51C	(#AA10)	43536d AA10h
2C4D2	(#AAA0)	43680d AAA0h
3B7AD	(#BBBB)	48059d BBBBh

Addr.	Name	Description
08F1F	(#D6A8)	54952d D6A8h
38266	(#FFFF)	65535d FFFFh
03880	(#102A8)	66216d 102A8h
091B4	(#2D541)	185665d 2D541h
350F5	(#37258)	225880d 37258h
0803F	(#414C1)	267457d 414C1h
08ECE	(#536A8)	341672d 536A8h
0657E	(#61441)	398401d 61441h
33995	#EXITERR	458752d 70000h
03826	(#A8241)	688705d A8241h
39277	(#B437D)	738173d B437Dh
038DC	(#E13A8)	922536d E13A8h
3399F	MINUSONE	1048575d FFFFFh

2.1.2 Pushing Several BINTs

Addr.	Name	Description
37287	ZEROZERO	(→ #0 #0)
37294	#ZERO#ONE	(→ #0 #1)
37305	#ZERO#SEVEN	(→ #0 #7)
36B12	ONEONE	(→ #1 #1) aka: ONEDUP
37315	#ONE#27	(→ #1 #27d)
37328	#TWO#ONE	(→ #2 #1)
3733A	#TWO#TWO	(→ #2 #2)
3734A	#TWO#FOUR	(→ #2 #4)
3735C	#THREE#FOUR	(→ #3 #4)
3736E	#FIVE#FOUR	(→ #5 #4)
37380	ZEROZEROZERO	(→ #0 #0 #0)
37394	ZEROZEROONE	(→ #0 #0 #1)
373A8	ZEROZEROTWO	(→ #0 #0 #2)
3558C	DROPZERO	(ob → #0)
355A5	2DROP00	(ob ob → #0 #0)
3596D	DROPONE	(ob → #1)
36AD6	DUPZERO	(ob → ob ob #0)
36AEA	DUPONE	(ob → ob ob #1)
36B26	DUPTWO	(ob → ob ob #2)
36AFE	SWAPONE	(ob ob' → ob' ob #1)

Addr.	Name	Description
35E75	ZEROSWAP	(ob → #0 ob)
360BB	ZEROOVER	(ob → ob #0 ob)
36568	ZEROFALSE	(→ #0 F)
35EA2	ONESWAP	(ob → #1 ob)
3657C	ONEFALSE	(→ #1 F)

2.1.3 Conversion

Addr.	Name	Description
262F1	COERCE	(% → #)
35D08	COERCEDUP	(% → # #)
35EB6	COERCESWAP	(ob % → # ob)
3F481	COERCE2	(% %' → # #')
262EC	%ABSCOERCE	(% → #)
2F244	(COERCE&CKSGN)	(% → # flag) TRUE if real is greater 0, else FALSE.
2F31F	C%>AFF#	(C% → # #')
05A03	HXS>#	(hxs → #)
2F17E	2HXSLIST?	({ hxs hxs' } → # #') Converts list of two hxs to two bints. Generates "Bad Argument Value" for invalid input.
05A51	CHR>#	(chr → #)
0EF006	ˆZ2BIN	(Z → #) Convert Z to bint. Returns FFFFF for overflows. Returns 0 for negative numbers.
19D006	ˆZ>#	(z → #) Coerces Z to #, overflow error if Z<0 or Z>9999. 10000 is used to insure that the #*6 can be represented in BCD on a 5 nibbles field.
0F0006	ˆCOERCE2Z	(z2 z1 → #2 #1) Converts 2 zints to bints.

2.1.4 Arithmetic Functions

Addr.	Name	Description
03DBC	#+	(# #' → #+#')
03DEF	#1+	(# → #+1)
03E2D	#2+	(# → #+2)
355FD	#3+	(# → #+3)
35602	#4+	(# → #+4)
35607	#5+	(# → #+5)
3560C	#6+	(# → #+6)
35611	#7+	(# → #+7)
35616	#8+	(# → #+8)
3561B	#9+	(# → #+9)
35620	#10+	(# → #+10)
3562A	#12+	(# → #+12)
03DE0	#-	(# #' → #-#')
2F13D	(CK#-)	(# #' → #'') If #' is greater than #, returns #0, otherwise returns #-#'.
03E0E	#1-	(# → #-1)
03E4E	#2-	(# → #-2)
355DF	#3-	(# → #-3)
355DA	#4-	(# → #-4)
355D5	#5-	(# → #-5)
355D0	#6-	(# → #-6)
03EC2	#*	(# #' → #*#')
2632D	#*OVF	(# #' → #*#') 0 ≤ result ≤ FFFFF
03E6F	#2*	(# → #*2)
356B8	#6*	(# → #*6)
3569B	#8*	(# → #*8)
35675	#10*	(# → #*10)
03EF7	#/	(# #' → #r #q)
03E8E	#2/	(# → #/2) Rounded down.
36815	#1-	(# #' → #-#'+1) aka: #-+1
36851	#1-+	(# #' → #+#'-1) $1-+ is a typo in EXTABLE. aka: #+-1, $1-+

Addr.	Name	Description
35552	#-#2/	(# #' → (#-#')/2)
357FC	#+DUP	(# #' → #+#' #+#')
35E39	#+SWAP	(ob # #' → #+#' ob)
36093	#+OVER	(ob # #' → ob #+#' ob)
3581F	#-DUP	(# #' → #-#' #-#')
35E4D	#-SWAP	(ob # #' → #-#' ob)
360A7	#-OVER	(ob # #' → ob #-#' ob)
35830	#1+DUP	(# → #+1 #+1)
35E61	#1+SWAP	(ob # → #+1 ob)
2F222	#1+ROT	(ob ob' # → ob' #+1 ob)
35841	#1-DUP	(# → #-1 #-1)
28071	#1-SWAP	(ob # → #-1 ob) aka: pull
3601B	#1-ROT	(ob ob' # → ob' #-1 ob)
281D5	#1-UNROT	(ob ob' # → #-1 ob ob')
35E89	#1-1SWAP	(# → 1 #-1) Returns the bint ONE and the result.
35912	DUP#1+	(# → # #+1)
3571E	DUP#2+	(# → # #+2)
35956	DUP#1-	(# → # #-1)
3674D	2DUP#+	(# #' → # #' #+#') aka: DUP3PICK#+
3683D	DROP#1-	(# ob → #-1)
357BB	SWAP#-	(# #' → #'-#)
3592B	SWAP#1+	(# ob → ob #+1) aka: SWP1+
29786	('RSWAP#1+)	(# → nob #+1) nob is the next object in the runstream.
28099	SWAP#1+SWAP	(# ob → #+1 ob)
36829	SWAP#1-	(# ob → ob #-1)
280AD	SWAP#1-SWAP	(# ob → #-1 ob)
28989	(SWAPDROP#1-)	(ob # → #-1)
367ED	SWAPOVER#-	(# #' → #' #-#')
36775	OVER#+	(# #' → # #'+#)
367C5	OVER#-	(# #' → # #'-#)
36761	ROT#+	(# ob #' → ob #'+#)
367B1	ROT#-	(# ob #' → ob #'-#)
36801	ROT#1+	(# ob ob' → ob ob' #+1)
28001	ROT#1+UNROT	(# ob ob' → #+1 ob ob')

Addr.	Name	Description
35E07	ROT#+SWAP	(# ob #' → #'+# ob) aka: ROT+SWAP
36789	3PICK#+	(# ob #' → # ob #'+#)
3679D	4PICK#+	(# ob1 ob2 #' → # ob1 ob2 #'+#)
35E20	4PICK#+SWAP	(# ob1 ob2 #' → # ob1 #'+# ob2) aka: 4PICK+SWAP
35511	#MIN	(# #' → #'')
3551D	#MAX	(# #' → #'')
03EB1	#AND	(# #' → #'') Bitwise AND.

2.1.5 Tests

Addr.	Name	Description
03D19	#=	(# → flag)
03D4E	#<>	(# → flag)
03CE4	#<	(# → flag)
03D83	#>	(# → flag)
03CC7	#0<>	(# → flag)
03CA6	#0=	(# → flag)
3530D	#1<>	(# → flag)
352FE	#1=	(# → flag)
36711	#2<>	(# → flag)
352F1	#2=	(# → flag)
352E0	#3=	(# → flag)
366FD	#5=	(# → flag)
366BC	#<3	(# → flag)
36739	#>1	(# → flag) aka: ONE#>
358C2	2DUP#<	(# #' → # #' flag)
358F8	2DUP#>	(# #' → # #' flag)
363CE	ONE_EQ	(# → flag) Uses EQ test.
35268	OVER#=	(# #' → # flag)
358DC	2DUP#=	(# #' → # #' flag)
36694	OVER#0=	(# #' → # #' flag)
352BD	DUP#0=	(# → # flag)
366A8	OVER#<	(# #' → # flag)

Addr.	Name	Description
3531C	DUP#1=	(# → # flag)
36725	OVER#>	(# #' → # flag)
3532B	DUP#0<>	(# → # flag)
366D0	DUP#<7	(# → # flag) Returns TRUE if the argument is smaller then #7.
36676	2#0=OR	(# # → flag) Returns TRUE if either argument is zero.

Chapter 3
Real Numbers

Real numbers can be created in two ways. The first is by just entering them, without any prefix. But this method can also be used to create bints. So how does the compiler know when you want a real number and when you want a bint? If the number includes a radix and/or an exponent, then it is a real number; otherwise it is a bint.

Because of the possible confusion, the preferred method is to use the structure `% <dec>`. This way, you will surely get a real number, and the code becomes more readable.

As for bints, there are also many built-in real numbers. They are listed below.

The basic operations using real numbers are `%+`, `%-`, `%*`, `%/` and `%^`. But there are many others, which are listed below.

There is also another kind of real number, which is not directly accessible to the user and to User RPL programs. They are the Extended (or Long) Real Numbers. They work like normal real numbers, with two differences: they have a 15-digit precision opposed to the 12-digit of the normal real numbers, and their exponents are in the range from -50000 to 50000.

Extended real numbers are created using `%% <dec>`. If you happen to get one in the stack, they display like normal reals, but always in scientific notation. The basic operations are the same, except that they are prefixed with `%%` instead of `%`. Let me make thing one clear, if it is not already: in User RPL, + adds any kind of objects, for example real numbers, user binary integers (which are hexadecimal strings, as we will see later), adds elements to lists, etc. In System RPL, the word `%+` only works for two real numbers. To add two binary integers, you must use `#+`. To add extended reals, the word is `%%+`. If you call a function with the wrong arguments, there is a possibility that your system will crash.

To convert from a real number to an extended real number, you can use the command `%>%%`. The opposite function is `%%>%`. To convert from a bint to a (normal) real number, the function is `UNCOERCE`, and the opposite function is

COERCE. Below there is a list of more conversion functions, and other functions related to real numbers.

3.1 Reference

3.1.1 Built-in Real Numbers

Addr.	Name	Description
2FB0A	%-MAXREAL	-9.99E499
2FAB1	%-9	-9
2FA9C	%-8	-8
2FA87	%-7	-7
2FA72	%-6	-6
2FA5D	%-5	-5
2FA48	%-4	-4
2FA33	%-3	-3
2FA1E	%-2	-2
2FA09	%-1	-1
2FB34	%-MINREAL	-1E-499
2F937	%0	0
2FB1F	%MINREAL	1E-499
27118	%.1	.1
339BE	%.5	.5
339D3	(%-.5)	-.5
2F94C	%1	1
270EE	(%1.8)	1.8
2F961	%2	2
339A9	%e	e
2F976	%3	3
2FAC6	%PI	π
2F98B	%4	4
2F9A0	%5	5
2F9B5	%6	6
2F9CA	%7	7
2F9DF	%8	8
2F9F4	%9	9
339E8	%10	10
2FCE6	%11	11

Addr.	Name	Description
2FCFB	%12	12
2FD10	%13	13
2FD25	%14	14
2FD3A	%15	15
2FD4F	%16	16
2FD64	%17	17
2FD79	%18	18
2FD8E	%19	19
2FDA3	%20	20
2FDB8	%21	21
2FDCD	%22	22
2FDE2	%23	23
2FDF7	%24	24
2FE0C	%25	25
2FE21	%26	26
2FE36	%27	27
2FE4B	(%28)	28
2FE60	(%29)	29
2FE75	(%30)	30
2FE8A	(%31)	31
2FE9F	(%32)	32
2FEB4	(%33)	33
2FEC9	(%34)	34
2FEDE	(%35)	35
27103	%80	80
27E5D	%100	100
339FD	%180	180
33A12	(%200)	200
33A3C	(%400)	400
33A27	%360	360
2FC7D	(%1200)	1200
2FC92	(%2400)	2400
2FCA7	(%4800)	4800
4EA22	(%TICKSsec)	8192
2FCBC	(%9600)	9600
2FCD1	(%15360)	15360
4EA37	(%TICKSmin)	491520
4EA4C	(%TICKShour)	29491200
4EA61	(%TICKSday)	707788800

Addr.	Name	Description
4EA76	(%TICKSweek)	4954521600
2FAF5	%MAXREAL	9.99E499
2F180	1REV	(→ 6.28318530718) (→ 360.) (→ 400.) Returns the angle of a full circle, corresponding to the current angular mode.

3.1.2 Built-in Extended Real Numbers

Addr.	Name	Description
2FB49	%%0	0
2FBE5	%%.1	0.1
30DC8	%%.4	0.4
2FBFF	%%.5	0.5
2DA11	cfF	0.555... %%5/9 for C↔F conversion.
2FB63	%%1	1
2DA2B	cfC	1 For C↔K conversion.
2FB7D	%%2	2
2FB97	%%3	3
2FADB	%%PI	π
30017	PI/180	$\pi/180$
2FBB1	%%4	4
2FBCB	%%5	5
27A89	%%2PI	2π
30BEA	%%7	7
2FC19	%%10	10
30CC7	%%12	12
30CEB	%%60	60

3.1.3 Stack Manipulation Combined with Reals

Addr.	Name	Description
282CC	(DROP%0)	(ob → %0)

3.1.4 Conversion

Addr.	Name	Description
2FFAC	%>%%	(% → %%)
35ECA	%>%%SWAP	(ob % → %% ob)
2FF9B	%%>%	(%% → %)
30E47	2%>%%	(% % → %% %%)
30E5B	2%%>%	(%% %%' → % %')
262F6	UNCOERCE	(# → %)
3F495	UNCOERCE2	(# # → % %)
36BFA	UNCOERCE%%	(# → %%)
2EFCA	HXS>%	(hxs → %)
05D2C	C%>%	(C% → %re %im)
2B3FD	%IP>#	(% → #IP(ABS(%))) Does ABS too.
0F6006	^Z>R	(Z → %) Converts zint to real.
18A006	^Z2%%	(Z → %%) Converts integer to long real.
197006	^OBJ2REAL	(z/% → %) Transforms ob in real.

3.1.5 Real Functions

Addr.	Name	Description
3035F	%+	(% %' → %+%')
25E69	%+SWAP	(ob % %' → %+%' ob)
26F36	%1+	(% → %+1)
3036C	%-	(% %' → %-%')
26F4A	%1-	(% → %-1)
30346	%>%%-	(% %' → %%-%%')
303A7	%*	(% %' → %*%')

Addr.	Name	Description
35C18	%10*	(% → %*10)
303E9	%/	(% %' → %/%')
3045B	%^	(% %' → %^%')
302EB	%ABS	(% → %')
3030B	%CHS	(% → -%)
302C2	%SGN	(% → -1/0/1)
3049A	%1/	(% → 1/%)
30489	%>%%1/	(% → 1/%%)
304F4	%SQRT	(% → √%)
304E1	%>%%SQRT	(% → √%%)
3051A	%EXP	(% → e^%)
3052D	%EXPM1	(% → e^%-1)
30559	%LN	(% → LN%)
30592	%LNP1	(% → LN(%+1))
3056C	%LOG	(% → LOG%)
305A5	%ALOG	(% → 10^%)
305DA	%SIN	(% → SIN%)
3062B	%COS	(% → COS%)
3067C	%TAN	(% → TAN%)
306AC	%ASIN	(% → ASIN%)
306DC	%ACOS	(% → ACOS%)
3070C	%ATAN	(% → ATAN%)
30799	%SINH	(% → SINH%)
307C5	%COSH	(% → COSH%)
307D8	%TANH	(% → TANH%)
307EB	%ASINH	(% → ASINH%)
307FE	%ACOSH	(% → ACOSH%)
30811	%ATANH	(% → ATANH%)
3031B	%MANTISSA	(% → %mant)
30824	%EXPONENT	(% → %expn)
30938	%FP	(% → %frac)
3094B	%IP	(% → %int)
30971	%FLOOR	(% → %maxint <=%)
3095E	%CEIL	(% → %minint >=%)
305C7	%MOD	(% %' → %rem)
30723	%ANGLE	(%x %y → %ang)
30746	%>%%ANGLE	(%x %y → %%ang)
30F14	RNDXY	(% %places → %')

Addr.	Name	Description
30F28	TRCXY	(% %places → %')
3084D	%COMB	(% %' → COMB(%,%'))
30860	%PERM	(% %' → PERM(%,%'))
30837	%NFACT	(% → %!) Calculates factorial of number.
30AAF	%FACT	(% → gamma(%+1)) Calculates gamma(x+1).
3046C	%NROOT	(% %n → %') Calculates the %nth root of the real number. Equivalent to user function XROOT.
300F9	%MIN	(% %' → %lesser)
300E0	%MAX	(% %' → %greater)
35DBC	%MAXorder	(% %' → %max %min)
309AD	%RAN	(→ %random) Returns next random number.
30A2F	%RANDOMIZE	(%seed →) System level RDZ: seeds the random number generator.
30A66	DORANDOMIZE	(% →) Stores given number as random number seed.
303B4	%OF	(% %' → %'/% * 100)
303F6	%T	(% %' → %pctotal)
3041B	%CH	(% %' → %pcchange)
3000D	%D>R	(%deg → %rad)
30040	%R>D	(%rad → %deg)
30E79	%REC>%POL	(%r %ang → %x %y)
30EA6	%POL>%REC	(%x %y → %r %ang)
30EDD	%SPH>%REC	(%r %ang %ph → %x %y %z)

3.1.6 Extended Real Functions

Addr.	Name	Description
3032E	%%+	(%% %%' → %%+%%')
3033A	%%-	(%% %%' → %%-%%')
30385	%%*	(%% %%' → %%*%%')
3602F	%%*ROT	(ob ob' %% %%' → ob' %%+%%' ob)
35EDE	%%*SWAP	(ob %% %%' → %%+%%' ob)
36C7C	%%*UNROT	(ob ob' %% %%' → %%+%%' ob ob')

Addr.	Name	Description
303D3	%%/	(%% %%' → %%/%%')
36C22	SWAP%%/	(%% %%' → %%'')
36BE6	%%/>%	(%% %%' → %)
3044A	%%^	(%% %%' → %%^%%')
51D006	^CK%%SQRT	(%% → %%/C%%)
30612	%%SINRAD	(%% → %%')
30767	%%ANGLERAD	(%% → %%')
302DB	%%ABS	(%% → %%abs)
306F3	%%ACOSRAD	(%% → %%rad)
3073A	%%ANGLE	(%%x %%y → %%ang)
30757	%%ANGLEDEG	(%%x %%y → %%deg)
306C3	%%ASINRAD	(%% → %%rad)
302FB	%%CHS	(%% → -%%)
3047D	%%1/	(%% → 1/%%)
30642	%%COS	(%% → %%cos)
30653	%%COSDEG	(%%deg → %%cos)
307B2	%%COSH	(%% → %%cosh)
30663	%%COSRAD	(%%rad → %%cos)
30507	%%EXP	(%% → e^%%)
30546	%%LN	(%% → ln %%)
30984	%%FLOOR	(%% → %%maxint)
		aka: %%INT
3057F	%%LNP1	(%% → %%ln(%%+1))
300C7	%%MAX	(%% %%' → %%max)
30E83	%%R>P	(%%x %%y → %%radius %%angle)
30EB0	%%P>R	(%%r %%ang → %%x %%y)
305F1	%%SIN	(%% → %%sin)
30602	%%SINDEG	(%%deg → %%sin)
30780	%%SINH	(%% → %%sinh)
304D5	%%SQRT	(%% → √%%)
30693	%%TANRAD	(%%rad → %%tan)

3.1.7 Tests

Addr.	Name	Description
302AC	%=	(% %' → flag)
302B7	%<>	(% %' → flag)
3025C	%<	(% %' → flag)
302A1	%<=	(% %' → flag)
30275	%>	(% %' → flag)
3028B	%>=	(% %' → flag)
30156	%0=	(% → flag)
36C0E	DUP%0=	(% → flag)
301BA	%0<>	(% → flag)
		Can be used to change a user flag into a system flag.
30123	%0<	(% → flag)
30184	%0>	(% → flag)
301E2	%0>=	(% → flag)
3020A	%%<	(%% %%' → flag)
30296	%%<=	(%% %%' → falg)
3026A	%%>	(%% %%' → flag)
30280	%%>=	(%% %%' → flag)
30145	%%0=	(%% → flag)
301A6	%%0<>	(%% → flag)
30112	%%0<	(%% → flag)
301F6	%%0<=	(%% → flag)
30173	%%0>	(%% → flag)
301CE	%%0>=	(%% → flag)

Chapter 4
Complex Numbers

Complex numbers can be inserted in your program with the following structure: C% <real> <imag>. The real and imaginary parts are real numbers, in decimal form. If you have the real and imaginary parts in the stack, the command %>C% will create a complex number from them. The command C%>% takes a complex number and returns the real and imaginary parts.

There exists also the Extended (also called Long) Complex Numbers, which are not directly accessible to the user. They are complex number whose real and imaginary parts are extended reals. They can be inserted in your program with C%% <real> <imag>, where the real and imaginary parts are extended reals. They show in the stack as a normal complex number, but always in scientific notation.

Below is a list of all the commands related to complex numbers, including mathematical operations.

4.1 Reference

4.1.1 Builtin Complex Numbers

Addr.	Name	Description
27DE4	C%0	(0,0)
27E09	C%1	(1,0)
27DBF	C%-1	(-1,0)
27E2E	C%%1	(%%1,%%0)

4.1.2 Conversion

Addr.	Name	Description
261D9	C%%>C%	(C%% → C%)
05C27	%>C%	(%re %im → C%)
362F2	SWAP%>C%	(%im %re → C%)
261FC	Re>C%	(%re → C%)
25E9C	C>Re%	(C% → %re)
25E9B	C>Im%	(C% → %im)
18C006	^E%%>C%%	(%%re %%im → C%%) Converts long reals to long complex.
261CF	%%>C%	(%%re %%im → C%)
25E82	C%>%%	(C% → %%re %%im)
25E83	C%>%%SWAP	(C% → %%im %%re)
05DBC	C%%>%%	(C%% → %%re %%im)
188006	^C2C%%	(C → C%%) Converts Gaussian integer to long complex.
189006	^ZZ2C%%ext	(Zre Zim → C%%) Converts Gaussian integer to long complex.
18B006	^C%>C%%	(C% → C%%) Converts complex to long complex.
15E006	^RIXCext	(Zre Zim → C) Convert integers to complex.
15F006	^IRXCext	(Zim Zre → C) Convert integers to complex.

4.1.3 Functions

Addr.	Name	Description
25E8F	C%C^C	(C% C%' → C%'')
25E90	C%C^R	(C% % → C%')
25E94	C%R^C	(% C% → C%')
25E84	C%ABS	(C% → %)
50C006	^CZABS	(complex → real) Absolute value.
261ED	C%CHS	(C% → -C%)
25E81	C%1/	(C% → 1/C%)
25E98	C%SQRT	(C% → $\sqrt{\text{C\%}}$)
25E95	C%SGN	(C% → C%/C%ABS)

Addr.	Name	Description
261F2	C%CONJ	(C% → C%')
25E88	C%ARG	(C% → %)
25E91	C%EXP	(C% → eˆC%)
25E92	C%LN	(C% → ln C%)
25E93	C%LOG	(C% → log C%)
25E87	C%ALOG	(C% → 10ˆC%)
25E96	C%SIN	(C% → sin C%)
25E8D	C%COS	(C% → cos C%)
25E99	C%TAN	(C% → tan C%)
25E89	C%ASIN	(C% → asin C%)
25E85	C%ACOS	(C% → acos C%)
25E8B	C%ATAN	(C% → atan C%)
25E97	C%SINH	(C% → sinh C%)
25E8E	C%COSH	(C% → cosh C%)
25E9A	C%TANH	(C% → tanh C%)
25E8A	C%ASINH	(C% → asinh C%)
25E86	C%ACOSH	(C% → acosh C%)
25E8C	C%ATANH	(C% → atanh C%)
261DE	C%%CHS	(C%% → -C%%)
261E3	C%%CONJ	(C%% → C%%')
515006	ˆARG2	(im re → arg(ob)) ARG.
517006	ˆQUADRANT	(re im ?re>0 ?im>0 → newre newim %) Returns Z0 Z1 Z-2 or Z-1 so that arg of corresponding complex number is Z * $\pi/2$ + theta where θ is in the interval [0,$\pi/2$].
51E006	ˆC%%SQRT	(C%% → C%%')

4.1.4 Tests

Addr.	Name	Description
261E8	C%0=	(C% → flag)
261D4	C%%0=	(C%% → flag)

Chapter 5
Integers (ZINTS)

This is a new object of the HP49. The integers (called ZINT's for shorts) are a numerical type that can represent arbitrarily large integers.

In most cases, you do not really need to worry about integers entered by the user as arguments for a program. The type checking mechanism (described in section 29.2) will in most cases transparently convert zints to real numbers.

If you want to work with integers, however, there are several functions dealing with zints. Since this object type is really a part of the HP49 CAS, these functions are not described here. Instead, turn to Chapter 42 for documentation on ZINTs.

Chapter 6
Characters and Strings

Characters and strings are two data types that hold text.

Characters are not directly available to the user. They can only hold one character. You create them with `CHR <char>` or using one of the many built-in characters (listed below). To convert a character to a bint, use `CHR>#`. The bint returned is the ASCII code for the character. The opposite function is `#>CHR`.

Strings are inserted in your program with `$ "<string>"`, or simply `"<string>"`. There are some built-in strings, listed below. It is possible to convert a character into a string, with the command `CHR>$`.

Two useful and simple functions which deal with strings are `LEN$` and `&$`. The first returns the length (the number of characters) of a string as a bint, and the second concatenates two strings. To get a substring, i.e., part of a string, use the function `SUB$`. It expects three arguments: the original string, the starting position (a bint) and the final position (also a bint). Counting starts at one. Everything between the start and end characters (inclusive) will be returned. And another function is `POS$`, which searches a string (in level three) for a character or string (in level two), starting from a specified position (a bint, in level one). The position of the first occurrence of the search string in the string is returned (as a bint) to level one. If it could not be found, `#0` is returned. There are also many other functions, see below for a list.

6.1 Reference

6.1.1 Built-in Characters

Addr.	Name	Description
33D2B	CHR_00	'\00' (character 0d 00h)
		The NULL character.
33F77	CHR_Newline	'\0a' (character 10d 0Ah)
		The newline character.
33D32	CHR_...	'...' (character 31d 1Fh)
33F93	CHR_Space	'␣' (character 32d 20h)
		The space character.
33D39	CHR_DblQuote	'"' (character 34d 22h)
33D40	CHR_#	'#' (character 35d 23h)
33F70	CHR_LeftPar	'(' (character 40d 28h)
33F85	CHR_RightPar	')' (character 41d 29h)
33D47	CHR_*	'*' (character 42d 2Ah)
33D4E	CHR_+	'+' (character 43d 2Bh)
33D55	CHR_,	',' (character 44d 2Ch)
33D5C	CHR_-	'-' (character 45d 2Dh)
33D63	CHR_.	'.' (character 46d 2Eh)
33D6A	CHR_/	'/' (character 47d 2Fh)
33D71	CHR_0	'0' (character 48d 30h)
33D78	CHR_1	'1' (character 49d 31h)
33D7F	CHR_2	'2' (character 50d 32h)
33D86	CHR_3	'3' (character 51d 33h)
33D8D	CHR_4	'4' (character 52d 34h)
33D94	CHR_5	'5' (character 53d 35h)
33D9B	CHR_6	'6' (character 54d 36h)
33DA2	CHR_7	'7' (character 55d 37h)
33DA9	CHR_8	'8' (character 56d 38h)
33DB0	CHR_9	'9' (character 57d 39h)
33DB7	CHR_:	':' (character 58d 3Ah)
33DBE	CHR_;	';' (character 59d 3Bh)
33DC5	CHR_<	'<' (character 60d 3Ch)
33DCC	CHR_=	'=' (character 61d 3Dh)
33DD3	CHR_>	'>' (character 62d 3Eh)
33DDA	CHR_A	'A' (character 65d 41h)
33DE1	CHR_B	'B' (character 66d 42h)

Addr.	Name	Description
33DE8	CHR_C	'C' (character 67d 43h)
33DEF	CHR_D	'D' (character 68d 44h)
33DF6	CHR_E	'E' (character 69d 45h)
33DFD	CHR_F	'F' (character 70d 46h)
33E04	CHR_G	'G' (character 71d 47h)
33E0B	CHR_H	'H' (character 72d 48h)
33E12	CHR_I	'I' (character 73d 49h)
33E19	CHR_J	'J' (character 74d 4Ah)
33E20	CHR_K	'K' (character 75d 4Bh)
33E27	CHR_L	'L' (character 76d 4Ch)
33E2E	CHR_M	'M' (character 77d 4Dh)
33E35	CHR_N	'N' (character 78d 4Eh)
33E3C	CHR_O	'O' (character 79d 4Fh)
33E43	CHR_P	'P' (character 80d 50h)
33E4A	CHR_Q	'Q' (character 81d 51h)
33E51	CHR_R	'R' (character 82d 52h)
33E58	CHR_S	'S' (character 83d 53h)
33E5F	CHR_T	'T' (character 84d 54h)
33E66	CHR_U	'U' (character 85d 55h)
33E6D	CHR_V	'V' (character 86d 56h)
33E74	CHR_W	'W' (character 87d 57h)
33E7B	CHR_X	'X' (character 88d 58h)
33E82	CHR_Y	'Y' (character 89d 59h)
33E89	CHR_Z	'Z' (character 90d 5Ah)
33FA1	CHR_[	'[' (character 91d 5Bh)
33FA8	CHR_]	']' (character 93d 5Dh)
33F9A	CHR_UndScore	'_' (character 95d 5Fh)
33E90	CHR_a	'a' (character 97d 61h)
33E97	CHR_b	'b' (character 98d 62h)
33E9E	CHR_c	'c' (character 99d 63h)
33EA5	CHR_d	'd' (character 100d 64h)
33EAC	CHR_e	'e' (character 101d 65h)
33EB3	CHR_f	'f' (character 102d 66h)
33EBA	CHR_g	'g' (character 103d 67h)
33EC1	CHR_h	'h' (character 104d 68h)
33EC8	CHR_i	'i' (character 105d 69h)
33ECF	CHR_j	'j' (character 106d 6Ah)
33ED6	CHR_k	'k' (character 107d 6Bh)
33EDD	CHR_l	'l' (character 108d 6Ch)

Addr.	Name	Description
33EE4	CHR_m	'm' (character 109d 5Dh)
33EEB	CHR_n	'n' (character 110d 6Eh)
33EF2	CHR_o	'o' (character 111d 6Fh)
33EF9	CHR_p	'p' (character 112d 70h)
33F00	CHR_q	'q' (character 113d 71h)
33F07	CHR_r	'r' (character 114d 72h)
33F0E	CHR_s	's' (character 115d 73h)
33F15	CHR_t	't' (character 116d 74h)
33F1C	CHR_u	'u' (character 117d 75h)
33F23	CHR_v	'v' (character 118d 76h)
33F2A	CHR_w	'w' (character 119d 77h)
33F31	CHR_x	'x' (character 120d 78h)
33F38	CHR_y	'y' (character 121d 79h)
33F3F	CHR_z	'z' (character 122d 7Ah)
33FAF	CHR_{	'{' (character 123d 7Bh)
33FB6	CHR_}	'{' (character 125d 7Dh)
33F5B	CHR_Angle	'$\measuredangle$' (character 128d 80h)
33F69	CHR_Integral	'$\int$' (character 132d 84h)
33F62	CHR_Deriv	'∂' (character 136d 88h)
33F46	CHR_->	'$\rightarrow$' (character 141d 8Dh)
33F4D	CHR_<<	'«' (character 171d ABh)
33F54	CHR_>>	'»' (character 187d BBh)
33F7E	CHR_Pi	'π' (character 135d 87h)
33F8C	CHR_Sigma	'Σ' (character 133d 85h)
33FBD	CHR_<=	'$\leq$' (character 137d 89h)
33FC4	CHR_>=	'$\geq$' (character 138d 8Ah)
33FCB	CHR_<>	'$\neq$' (character 139d 8Bh)

6.1.2 Built-in Strings

Addr.	Name	Description
055DF	NULL$	"" Empty string.
33B55	SPACE$	"␣" aka: tok_
33B39	NEWLINE$	"\0a" Newline.

Addr.	Name	Description
27195	CRLF$	"\0d\0a"
		Carriage return and line feed.
340A4	$_RAD	"RAD"
340B4	$_GRAD	"GRAD"
33FF2	$_XYZ	"XYZ"
33FE2	$_R<Z	"R∡Z"
		"R<angle>Z"
33FD2	$_R<<	"R∡∡"
		"R<angle><angle>"
34076	$_EXIT	"EXIT"
34064	$_ECHO	"ECHO"
34088	$_Undefined	"Undefined"
34002	$_<<>>	"«»"
34010	$_{}	"{}"
3401E	$_[]	"[]"
3402C	$_''	"''"
		Two single quotes.
3403A	$_::	"::"
34048	$_LRParens	"()"
34056	$_2DQ	""""
		Two double quotes.
33B91	tok,	","
33B85	tok'	"'"
		One single quote.
33BFD	tok-	"-"
33B9D	tok.	"."
2D848	tok_g	"g"
2D86D	tok_m	"m"
2D8AD	tok_s	"s"
33A77	tok{	"{"
33AD7	tok<<	"«"
33C09	tok=	"="
272D9	tok->	"→"
33C4D	tok0	"0"
33C59	tok1	"1"
33CAD	tok8	"8"
33CB9	tok9	"9"
33ABF	tokESC	"<ESC>"
		Escape character.

Addr.	Name	Description
33AE3	tokexponent	"E"
33B79	tokquote	""""
		One double quote.
33A8F	toksharp	"#"
33AA7	(tok$)	"$"
33AB3	(tok&)	"&"
33BD9	(tok*)	"*"
33BF1	(tok+)	"+"
33BE5	(tok/)	"/"
33C65	(tok2)	"2"
33C71	(tok3)	"3"
33C7D	(tok4)	"4"
33C89	(tok5)	"5"
33C95	(tok6)	"6"
33CA1	(tok7)	"7"
33BA9	(tok;)	"8"
33ACB	(tok>>)	"»"
33AEF	(tokanglesign)	"∡"
33C2D	(tokCTGROB)	"GROB"
33C3F	(tokCTSTR)	"C$"
33C21	(tokDER)	"∂"
33B45	($DER)	"der"
33BB5	(toklparen)	"("
33BC1	(tokrparen)	")"
33AFB	(tokSIGMA)	"Σ"
33C15	(tokSQRT)	"√"
33B61	(tokUNKNOWN)	"UNKNOWN"
33A9B	(tokuscore)	"_"
33B07	(tokWHERE)	"|"
33A6B	(tok[)	"["
33A51	(tok])	"]"
33BCD	(tok^)	"^"
33A83	(tok})	"}"
33B13	(14SPACES$)	"␣␣␣␣␣␣␣␣␣␣␣␣␣␣"
		String of 14 spaces.

6.1.3 Built-in Strings with Stack Manipulation

Addr.	Name	Description
35D94	`NULL$SWAP`	`( ob → $ ob )` `NULL$`, then `SWAP`.
04D3E	`DROPNULL$`	`( ob → NULL$ )` `DROP` then `NULL$`.
25EEC	`NULL$TEMP`	`( → $ )` Creates null string in temporary memory (`NULL$`, then `TOTEMPOB`).

6.1.4 Conversion

Addr.	Name	Description
25F77	`#>$`	`( # → $ )` Creates string from the bint (decimal).
25F72	`#:>$`	`( # → "#:␣" )` Creates string from the bint and appends a colon and a space. Ex: "1:␣"
25F0F	`a%>$`	`( % → $ )` Converts real number into string using current display mode. aka: `a%>$,`
05BE9	`ID>$`	`( id/lam → $ )` Converts identifier into string.
25EB3	`DOCHR`	`( % → $ )` Creates string of the character with the number specified.
0F1006	`^Z>S`	`( Z → $ )` Converts Z into a string (decimal).
2EFC1	`hxs>$`	`( hxs → $ )` Uses current display mode and wordsize.
2EFC0	`HXS>$`	`( hxs → $ )` Does `hxs>$` and then appends base character.

6.1.5 Management

Addr.	Name	Description
05A75	#>CHR	(# → chr) Returns character with the specified ASCII code.
37AA5	CHR>$	(chr → $* Strings) Converts a character into a string.
05636	LEN$	($ → #length) Returns length in bytes.
357E2	DUPLEN$	($ → $ #) DUP then LEN$.
05622	OVERLEN$	($ ob → $ ob #len) OVER then LEN$.
361DA	NEWLINE$&$	($ → "$\0a") Appends newline character to string. aka: NEWLINE&$
2F31A	APNDCRLF	($ → $') Appends carriage return and line feed to string.
050ED	CAR$	($ → chr) Returns first character of string as a string, or NULL$ for null string.
0516C	CDR$	($ → $') Returns string without first character, or NULL$ for null string.
378FA	POS$	($ $find start# → #pos) Search for $find in $search, starting at position #start. Returns position of $find or 0 if not found. Same entry as POSCHR.
378FA	POSCHR	($search chr #start → #pos) Same entry as POS$.
37906	POS$REV	($ $find #limit → #pos) Searches backwards from #limit to #1. Same entry as POSCHRREV.
37906	POSCHRREV	($seach chr #start → #pos) Same entry as POS$REV.

Addr.	Name	Description
25EA0	COERCE$22	($ → $') If the string is longer than 22 characters, truncates it to 21 characters and appends "...".
2F16D	Blank$	(#len → $) Creates a string with the specified number of spaces.
2EEF0	PromptIdUtil	(id ob → $) Creates string of the form "id: ob".
25EF8	SEP$NL	($ → $' $'') Separates string at the first newline. $" is the substring before the first newline; $' the substring after the first newline.
09A003	(^WRAP$)	($ #width → $') Replace SPACE chars with NEWLINE in order to fit the text in the given #width. Used by ViewStrObject. Very fast (bang type).
05733	SUB$	($ #start #end → $') Returns substring between specified positions.
3628E	#1-SUB$	($ #start #end+#1 → $') Does #1- and then SUB$.
362A2	1_#1-SUB$	($ #end → $') Returns substring from the first character to the character before the specified position. aka: 1_#1-SUB
362B6	LAST$	($ #start → $') Returns substring from the specified start position to the end (inclusive).
362CA	#1+LAST$	($ #start-#1 → $') Returns substring from the specified start position to the end (exclusive).
35DA8	SUB$SWAP	(ob $ # #' → $' ob) SUB$ then SWAP.
2A5CA	SUB$1#	($ # → #') Returns bint with ASCII code of character at the specified position.

Addr.	Name	Description
34C82	EXPAND	`( hxs #nibs → hxs' )` Appends null characters to the string. Since refers to the number of nibbles, you must use a number twice as large as the number of null characters you want appended.
05193	&$	`( $ $' → $+$' )` Concatenates two strings.
36FF6	&$SWAP	`( ob $ $' → $+$' ob )` &$ then SWAP.
353CD	!append$	`( $ $' → $+$' )` Tries &$, if not enough memory does !!append$?.
3533C	!insert$	`( $ $' → $'+$ )` Does SWAP then !append$.
35F6A	!append$SWAP	`( ob $ $' → $+$' ob )` !append$ then SWAP.
35369	!!append$?	`( $ $' → $+$' )` Attempts append "in place" if target is in tempob.
353F7	!!append$	`( $ $' → $+$' )` Tries appending "in place".
353EB	!!insert$	`( $ $' → $'+$ )` Tries inserting "in place".
0525B	>H$	`( $ chr → $' )` Prepends character to string
052EE	>T$	`( $ chr → $' )` Appends character to string.
35BD7	APPEND_SPACE	`( $ → $' )` Appends space to string.
35346	SWAP&$	`( $ $' → $'+$ )` Concatenates two strings.
2EED3	TIMESTR	`( %dt %tm → "dy dt tm" )` Returns string representation of time, using current format. Example: `"WED 06/24/98 10:00:45A"`
25E7C	AND$	`( $1 $2 → $' )` Logical AND. Errors if strings are not the same length.

Addr.	Name	Description
25EF0	OR$	($ $' → $'') Logical OR. Errors if strings are not the same length.
25F0D	XOR$	($ $' → $'') Logical XOR. Errors if strings are not the same length.
2F1A7	CHARSEDIT	(→) HP49 character browser. This is an interactive application from which characters can be echoed into the command line.

6.1.6 Parsing Strings

Addr.	Name	Description
25EB7	DOSTR>	($ → ?) Internal version of STR→.
2EF62	palparse	($ → ob T) ($ → $ #pos $' F) Tries parsing a string into an object. If successful, returns object and TRUE, otherwise returns position of error, the offending part of the string $', and FALSE. If the string contains several arguments, the resulting object is a secondary containing these objects.
25E68	!*trior	(F → <SKIP>) (T T → <COLA>)
25E67	!*triand	(T T →) (F T → F T <SEMI>)
26206	tok8cktrior	($1 $1 → :: $1 <Ob1> ;) ($1 $2 → :: $1 <Ob2> <Rest> ;)
261BB	tok8trior	(GNT data $1 $1 → :: GNT data GetNextToken ;) (GNT data $1 $2 → :: $1 <Ob1> <Rest> ;)
29E67	nultrior	(NULL$ → :: ;) ($ → :: $ <Ob1> <Rest> ;)
25EDB	GetNextToken	(hxs-mask $ #start → hxs-mask $ #next $token)

Addr.	Name	Description
2F33C	getmatchtok	(hxs-mask $ #loc $_tok → hxs-mask $ #next $match)
2EF6E	ParseFail	(ob $parsed #pos $' →) Uses DispBadToken to re-edit the parsed string and displays "Syntax Error".
2EF6F	DispBadToken	(ob $parsed #pos $' →) Re-edits the parsed string, positions the cursor to the location of the error. Used by ParseFail.

6.1.7 Decompilation

Addr.	Name	Description
2F191	!DcompWidth	(# →) Sets the width (in characters) of decompiled strings. This width is used to cut the resulting string (for stack display) or to break it into lines (mostly for editing). Note that most decompilation entries reset this value to the stack or editor width. Use stkdecomp$w and editdecomp$w to make sure the current width is used and not changed.
2F190	DcompWidth@	(→ #) Recalls the width of decompiled strings (in characters).
26459	setStdWid	(→) Sets DcompWidth to the standard value for stack display, either 19 or 30 characters, depending on system flag 72 (stack minifont).
2645E	setStdEditWid	(→) Sets DcompWidth to the width for editing, either 21 or 32 characters, depending on system flag 73 (edit minifont).
25F13	stkdecomp$w	(ob → $) Decompiles for stack display using the current DcompWidth to cut the string if it is too long.

Addr.	Name	Description
25E6D	1stkdecomp$w	(ob → $) Calls setStdWid and decompiles for stack display (cutting the string if necessary).
2A842	Decomp1Line	(ob → $) Same as 1stkdecomp$w.
2A904	RPNDecomp1Line	(ob → $) Same as Decomp1Line but enforce RPN mode (system flag 95 clear) during execution.
25E6F	>Review$	(id → $) Makes a string from the variable name and its contents (decompiled with Decomp1Line), for display with the review key.
2A8E4	DecompStd1Line32	(ob → $) Sets 32 as DcompWidth and decompiles using stkdecomp$w.
2A9C4	RPNDecompStd1Line32	(ob → $) Same as DecompStd1Line32 but enforce RPN mode (system flag 95 clear) during execution.
2A8C9	DecompStd1Line	(ob → $) Calls setStdWid and decompiles, cutting if the string becomes too long.
2A9A4	RPNDecompStd1Line	(ob → $) Same as DecompStd1Line but enforce RPN mode (system flag 95 clear) during execution.
2A893	Decomp#Disp	(ob # → $) Calls setStdWid and decompiles ob (User-RPL components only), breaks the string into lines and returns the first #+1 lines. Used for multiline display in stack level 1.
2A964	RPNDecomp#Disp	(ob # → $) Same as Decomp#Disp but enforce RPN mode (system flag 95 clear) during execution.

Addr.	Name	Description
2A878	Decomp#Line	(ob # → $) Similar to Decomp#Disp, but the returned string is an internal representation of the different lines to be displayed. Used for multiline display in stack level 1.
2A944	RPNDecomp#Line	(ob # → $) Same as Decomp#Line but enforce RPN mode (system flag 95 clear) during execution.
25F11	editdecomp$w	(ob → $) Decompiles entire object for editing. It only decompiles the UserRPL components. Some System RPL entries like TakeOver are simply skipped, others are written as "External". Breaks the resulting strings into lines using the current DcompWidth.
25ECE	EDITDECOMP$	(ob → $) Calls setStdEditWid and the decompiles for editing like editdecomp$w.
2A85D	DecompEdit	(ob → $) Same as EDITDECOMP$.
2A924	RPNDecompEdit	(ob → $) Same as DecompEdit but enforce RPN mode (system flag 95 clear) during execution.
2AA43	AlgDecomp	(ob → $) Calls DecompEdit with a few checks around it.
25EAA	DECOMP$	(ob → $) Calls setStdWid and decompiles entire object (UserRPL components only). Breaks the string into lines using DcompWidth as width.
39CB3	(ob&$)	(ob $ → "ob$") Applies DECOMP$ to ob and concatenates with the string.
39C9F	($&ob)	($ ob → "$ob") Applies DECOMP$ to ob and concatenates with the string.

Addr.	Name	Description
25EB1	DO>STR	($ → $) (ob → $) Internal version of →STR.
1A7006	^DO>STRID	(id/ob → $) Like DO>STR but without quotes for id.
2A8AE	DecompEcho	(ob → $) Calls setStdEditWid and decompiles the entire object (UserRPL only) into a single line.
2A984	RPNDecompEcho	(ob → $) Same as DecompEcho but enforce RPN mode (system flag 95 clear) during execution.
2F1BF	Decomp%Short	(% #width → $) Decompiles a real number into a string of the given #width. It will drop less significant digits or add zeros as needed, but will also exceed #width when necessary. E.g. "-1.e-33" cannot be written with less than 7 characters, so even if #width is less, 7 chars will be used. %0 is always decompiled as "0".
001004	^FSTR1	(ob → $) The decompiler used by stkdecomp$w, 1stkdecomp$w, Decomp1Line, DecompStd1Line32. DcompWidth must be set before this is called.
003004	^FSTR3	(ob # → $) The decompiler used by Decomp#Line. DcompWidth must be set before this is called.
004004	^FSTR4	(ob → $) The decompiler used by editdecomp$w, DecompEdit, EDITDECOMP$. DcompWidth must be set before this is called.
005004	^FSTR5	(ob → $) The decompiler used by DecompEcho. DcompWidth must be set before this is called.

Addr.	Name	Description
006004	^FSTR6	(ob # → $) The decompiler used by Decomp#Line. DcompWidth must be set before this is called.
007004	^FSTR7	(ob → $) The decompiler used by DO>STR. DcompWidth must be set before this is called.
009004	^FSTR9	(ob → $) The decompiler used by DecompStd1Line. DcompWidth must be set before this is called.
00D004	^FSTR13	(ob → $) The decompiler used by DECOMP$. DcompWidth must be set before this is called.
35B82	palrompdcmp	(romptr → $ T) Decompiles a rompointer for the UserRPL stack. If it is a named rompointer, returns the name. Otherwise returns "XLIB n m".

6.1.8 String Tests

Addr.	Name	Description
0556F	NULL$?	(ob → flag)
36252	DUPNULL$?	(ob → ob flag)
2F321	CkChr00	($ → $ flag) Returns FALSE if string contains any null characters.

Chapter 7

Hex Strings

Hexadecimal strings are the numbers that are called Binary Integers in the manual, which can be represented in several bases. In System RPL they are called Hexadecimal Strings. They are created using the structure HXS <len> <hexbody>. len is the length of the string (number of nibbles or hexadecimal digits), in hexadecimal form, and hexbody is the actual contents of it. The tricky part about it is that because of the HP internal architecture, you must enter the contents in reverse order. To get, for example, the hex string #12AD7h, you must enter HXS 5 7DA21. To get #12345678h use HXS 8 87654321. In System RPL, hexadecimal strings can be of any length, unlike in User RPL, where they are limited to 16 nibbles or 64 bits.

To convert an hex string to and from a bint, use the commands HXS># and #>HXS. To convert an HXS to and from a real number, use #>% (or HXS>%) and %>#.

See below for more commands related to hex strings.

7.1 Reference

7.1.1 Conversion

Addr.	Name	Description
059CC	#>HXS	(# → hxs) Length will be five.
2EFCB	%>#	(% → #) Converts real number into hxs. Should be called %>HXS.

7.1.2 General Functions

Addr.	Name	Description
2EFBE	WORDSIZE	(→ #) Returns the current wordsize as a bint.
2EFAA	dostws	(# →) Sets the current wordsize.
055D5	NULLHXS	HXS 0 Puts a null hxs in the stack.
0518A	&HXS	(hxs hxs' → hxs'') Appends hxs'' to hxs'.
34C82	EXPAND	(hxs #nibs → hxs') Appends #nibs zero nibbles to the hxs.
05616	LENHXS	(hxs → #nibs) Returns length in nibbles.
05815	SUBHXS	(hxs #m #n → hxs') Returns sub hxs string.
2EFB9	bit+	(hxs hxs' → hxs'') Adds two hxs.
2EFC8	bit%#+	(% hxs → hxs') Adds real to hxs, returns hxs.
2EFC9	bit#%+	(hxs % → hxs') Adds real to hxs, returns hxs.
2EFBA	bit-	(hxs hxs' → hxs'') Subtracts hxs2 from hxs1.
2EFC6	bit%#-	(% hxs → hxs') Subtracts hxs from real, returns hxs.
2EFC7	bit#%-	(hxs % → hxs') Subtracts real from hxs, returns hxs.
2EFBC	bit*	(hxs hxs' → hxs'') Multiplies two hxs.
2EFC4	bit%#*	(% hxs → hxs') Multiplies real by hxs, returns hxs.
2EFC5	bit#%*	(hxs % → hxs') Multiplies hxs by real, returns hxs.
2EFBD	bit/	(hxs hxs' → hxs'') Divides hxs1 by hxs2.
2EFC2	bit%#/	(% hxs → hxs') Divides real by hxs, returns hxs.

Addr.	Name	Description
2EFC3	bit#%/	(hxs % → hxs') Divides hxs by real, returns hxs.
2EFAC	bitAND	(hxs hxs' → hxs'') Bitwise AND.
2EFAD	bitOR	(hxs hxs' → hxs'') Bitwise OR.
2EFAE	bitXOR	(hxs hxs' → hxs'') Bitwise XOR.
2EFAF	bitNOT	(hxs → hxs') Bitwise NOT.
2EFB8	bitASR	(hxs → hxs') Arithmetic shift one bit to the right. The most significant bit (the sign) does not change.
2EFB6	bitRL	(hxs → hxs') Shifts circularly one bit to the left.
2EFB7	bitRLB	(hxs → hxs') Shifts circularly one byte to the left
2EFB4	bitRR	(hxs → hxs') Shifts circularly one bit to the right.
2EFB5	bitRRB	(hxs → hxs') Shifts circularly one byte to the right.
2EFB0	bitSL	(hxs → hxs') Shifts one bit to the left.
2EFB1	bitSLB	(hxs → hxs') Shifts one byte to the left.
2EFB2	bitSR	(hxs → hxs') Shifts one bit to the right.
2EFB3	bitSRB	(hxs → hxs') Shifts one byte to the right.

7.1.3 Tests

Addr.	Name	Description
2EFCC	HXS==HXS	(hxs hxs' → %flag) == test
2F0EE	HXS#HXS	(hxs hxs' → %flag) $\neq$ test

Addr.	**Name**	**Description**
2EFCF	HXS<HXS	(hxs hxs' → %flag) < test
2EFCD	HXS>HXS	(hxs hxs' → %flag) > test
2EFCE	HXS>=HXS	(hxs hxs' → %flag) ≥ test
2F0EF	HXS<=HXS	(hxs hxs' → %flag) ≤ test

Chapter 8
Identifiers

Identifiers are used to represent the names of objects stored in memory (i.e., variables). To the user, they appear in the stack between single quotes, that is, `''`. In System RPL, they are created with `ID <name>`. When you use this structure, you do not always get the identifier in the stack. It is always evaluated. So, if variable `anumber` contains 123.45 and you put somewhere in your program `ID anumber`, the identifier is evaluated, recalling the contents of the variable. This way, the stack will contain 123.45. To put an id to the stack, use `' ID <name>`. As you will see on Chapter 19, the command `'` puts the object after it in the stack. This is called *quoting*. However, `ID <name>` (without the `'`) will also put the id in the stack if there is no variable called `<name>`. This is similar to be behaviour you get when you enter the name of a variable without the quotes in the HP49 command line.

You can convert a string to an id using `$>ID`. The opposite transformation is archived with `ID>$`.

There is also another kind of identifiers: the temporary identifiers, or lams. These are used when creating local variables, and you will learn about them later in Chapter 18. They are created with `LAM <name>`, and work pretty much like normal ids.

Since ids are closely related to memory access, the functions dealing with its are listed in Chapter 24.

Chapter 9
Tagged Objects

In order to insert a tagged object in your program, use the structure `TAG <tag> <object>`. Tag is a string without quotes, and the object can be anything. To create `0: 1790`, for example, you would use `TAG 0 % 1790`. An object can have multiple tags, but there is not much use for that.

The word `>TAG` creates a tagged object, given the object (in level two) and a string representing the tag (in level one). `%>TAG` works the same way, but tags an object with a real number. `ID>TAG` tags an object with an identifier. To remove all tags from an object, call `STRIPTAGS`.

A few more commands related to tagged objects are listed on below.

Note that the programmer seldom needs to worry about tagged objects, because the type dispatching mechanism (which is described in section 29.2) can automatically strip tags from the arguments to your program.

9.1 Reference

Addr.	Name	Description
05E81	>TAG	(ob $tag → tagged) Tags an object.
2F266	USER$>TAG	(ob $tag → tagged) Maximum of 255 characters in string.
2F223	%>TAG	(ob % → tagged) Converts real to string using current display mode and tags object.
05F2E	ID>TAG	(ob id/lam → tagged) Tags object with identifier or lam.
37B04	TAGOBS	(ob $tag → tagged) (ob.. { $.. } → tagged...) Tags one or more objects.

Addr.	Name	Description
37ABE	STRIPTAGS	(tagged → ob) Strips all tags from the object.
37AEB	STRIPTAGSl2	(tagged ob' → ob ob') Strips all tags from the object in level two.

Chapter 10
Arrays

There are actually two groups of objects that represent arrays in the HP49G. The first group (which will be described in this chapter) has existed since the HP48: the normal arrays (to the user they can be only of real or complex numbers), and the linked arrays, which are not accessible to the user. The HP49 introduced a new kind of object to represent arrays: the Symbolic Matrices. Since these are actually a part of the HP49 CAS, they are described in Chapter 43.

In User RPL, arrays can be only of real or complex numbers. In System RPL, you can have arrays of anything, even arrays of arrays. Note that an array is not a composite object (see Chapter 11), even if it looks like one. Also, an array can only contain one kind of object.

Using MASD, arrays are entered like this:

```
ARRY [[ % 1. % 2. %3. ]
       [ % 4. % 5. %6. ]]
```

This is not much different from entering an array in the normal HP49 command line.

You can also create an array of (normal, not extended) real or complex numbers by putting them in order in the stack, and entering a list representing the dimensions of the array (real numbers, not bints) in level one. Then run `^XEQ>ARRY`. This function does error checks to ensure there are enough arguments and if they are of the supported types.

The function `^ARSIZE` returns the number of elements in an array. You can get the dimensions of the array with `^DIMLIMITS`, which returns a list of bints representing the array dimensions. To get one element of an array, put the element number in level two, the array in level one, and run `GETATELN`. You will get the element and `TRUE` if it was found or only `FALSE` if the element does not exist. More array functions are listed below.

There is also another kind of array: the linked arrays. Linked arrays are like normal arrays, except that they have a table with pointers to all the objects in the array. This makes access to array elements faster, because when you need to access one object in the linked array, the only thing necessary is to read the pointer to that object in the table, and go directly there. With normal arrays, a sequential search is necessary.

The entries here all deal with the normal arrays (even though some of them also work for CAS' Symbolic matrices). For entries specific to Symbolic matrices, see Chapter 43.

10.1 Reference

10.1.1 General Functions

Addr.	Name	Description
0371D	GETATELN	(# [] → ob T) (# [] → F) Gets one element from array.
16D006	^MDIMS	([[]] → #rows #cols T) ([] → #elem F) Returns the size of an array. Equivalent to the HP48 command MDIMS.
35FD8	MDIMSDROP	([2D] → #m #n) MDIMS followed by DROP.
16E006	^DIMLIMITS	([] → { # }) ([[]] → {# #}) Returns the size of an array, like the User command SIZE, but the lengths are bints and not reals. Equivalent to the HP48 command DIMLIMITS.
35E006	^ARSIZE	([] → #) Returns max # in an array.
36183	OVERARSIZE	([] ob → [] ob #elts) Does OVER then ARSIZE.
260F8	PULLREALEL	([%] # → [%] %) Gets real element.
260F3	PULLCMPEL	([C%] # → [C%] C%) Gets complex element.

Addr.	Name	Description
26102	PUTEL	([%] % # → [%]') ([C%] C% # → [C%]') Puts element at specified position. Converts to "short" before. Warning: no copy to tempob first.
26107	PUTREALEL	([%] % # → [%]') Puts real element at specified position. Warning: no copy to tempob first.
260FD	PUTCMPEL	([C%] C% # → [C%]') Puts complex element at specified position. Warning: no copy to tempob first.
33B006	^MATTRAN	(M → M') Matrix transposition.
331006	^Yext	(V2 V1 → ob) Scalar product of symbolic vectors, no check.

10.1.2 Conversion

Addr.	Name	Description
169006	^BESTMATRIXTYPE	(ob → ob) Converts symbolic matrix with real/cmplex entries to a numeric array.
172006	^CKNUMARRY	(ob → ob) Tests if ob is a numeric array. Tries to convert symbolic array to numeric array.
178006	^MATRIX2ARRAY	([] → []) ([[]] → [[]]) Tries to convert a symbolic matrix to a numeric one.
001007	^ListToArry	({}/{{}} → []/[[]] TRUE) ({}/{{}} → FALSE) If possible, converts list of lists to normal array and returns TRUE. Otherwise, returns FALSE.
17F006	^XEQ>ARRY	(ob1...obn {%n} → []) (ob11...obmn {%m %n} → [[mxn]]) Builds a matrix a la →ARRY.
17C006	^XEQARRY>	([] → ob1...obn meta-arry) Explodes a matrix a la →ARRY.

Addr.	Name	Description
002007	^ArryToMatrix	([] → M) Converts array to symbolic array.

10.1.3 Statistics

Addr.	Name	Description
2EEDA	STATCLST	(→) Clears ΣDAT.
2EEDB	STATSADD%	(% →) Internal Σ+.
2EEDC	STATN	(→ N) Internal NΣ.
2EEDF	STATSMIN	(→ %) Internal MINΣ.
2EEDD	STATSMAX	(→ %) Internal MAXΣ.
2EEDE	STATMEAN	(→ %) (→ []) Internal MEAN.
2EEE0	STATSTDEV	(→ %) (→ []) Internal SDEV.
2EEE1	STATTOT	(→ %) (→ []) Internal TOT.
2EEE2	STATVAR	(→ %) (→ []) Internal VAR.

Chapter 11
Composite Objects

Composite objects hold other objects inside them. In contrast to arrays, different types of objects can be part of the same composite. We have already encountered composite objects in the Introduction, when we used a secondary to group several commands into a single object.

All composites are similar in structure: they start with a word which varies depending on the kind of object, and end with the word `SEMI`.

Besides secondaries, other composite objects are lists, symbolic objects (described in Chapter 14) and unit objects (described in Chapter 13).

You can create a list by starting it with `{`, and ending it with `}`. Inside, put as many objects as you wish, of any kind. Secondaries are delimited with `::` and `;`.

To concatenate two composites, put them in the stack and use `&COMP`. To add just one object to the head (beginning) or tail (end) of a composite, first put the composite in the stack, then the object, and call `>HCOMP` or `>TCOMP`, respectively. To get the length of the composite (the number of objects, as a bint), just put the composite in level one and use the command `LENCOMP`. To explode the composite into all its objects and a count (like the User RPL command `OBJ→`), use `INNERCOMP`. The only difference is that the number of objects is returned as a bint. To get one object of a composite, put the composite in level two, the object's position in level one (as a bint, naturally), and run `NTHELCOMP`. If the number were out of range, you would get a `FALSE`, otherwise the object and `TRUE`. `NTHCOMPDROP` is the above entry, followed by `DROP`. And to get part of a composite, use the function `SUBCOMP`. This function takes in level three the composite, in level two the start position (guess what? a bint) and in level one the end position (from now on, unless otherwise noted, all numeric arguments are bints). You will get a composite (of the same type, obviously) with the elements between the start and end positions, inclusive. This function checks if the numbers are not out of range. If they are, a null composite (an empty composite) is returned. The same happens if the end position is greater than the start position.

Other commands are listed in the reference section below.

11.1 Reference

11.1.1 General Operations

Addr.	Name	Description
0521F	&COMP	(comp comp' → comp'') Concatenates two composites.
052FA	>TCOMP	(comp ob → comp+ob) Adds ob to tail (end) of composite.
052C6	>HCOMP	(comp ob → ob+comp) Adds ob to head (beginning) of composite.
39C8B	(SWAP>HCOMP)	(ob comp → ob+comp) Does SWAP then >HCOMP.
05089	CARCOMP	(comp → ob_head) (comp_null → comp_null) Returns first object of the composite, or a null composite if the argument is a null composite.
361C6	?CARCOMP	(comp T → ob) (comp F → comp) If the flag is TRUE, does CARCOMP.
05153	CDRCOMP	(comp → comp-ob_head) (comp_null → comp_null) Returns the composite minus its first object, or a null composite if the argument is a null composite.
2825E	(2NELCOMPDROP)	(comp → ob2) Gets the second element of composite.
2BC006	^LASTCOMP	(comp → ob) Gets the last element of composite. Does DUPLENCOMP then NTHCOMPDROP.
0567B	LENCOMP	(comp → #n) Returns length of composite (number of objects).
3627A	DUPLENCOMP	(comp → comp #n) Does DUP then LENCOMP.
055B7	NULLCOMP?	(comp → flag) If the composite is empty, returns TRUE.
36266	DUPNULLCOMP?	(comp → comp flag) Does DUP then NULLCOMP?.

Addr.	Name	Description
056B6	NTHELCOMP	(comp #i → ob T) (comp #i → F) Returns specified element of composite and TRUE, or just FALSE if it could not be found.
35BC3	NTHCOMPDROP	(comp #i → ob) Does NTHELCOMP then DROP.
35D58	NTHCOMDDUP	(comp #i → ob ob) Does NTHCOMPDROP then DUP.

376EE	POSCOMP	(comp ob pred → #i)
		(comp ob pred → #0)
		(eg: pred = ' %<)
		Evaluates pred for all elements of composite and ob, and returns index of first object for which the pred is TRUE. If no one returned TRUE, returns #0. For example, the program below returns #4:
		`:: { %1 %2 %3 %-4 %-5 %6 %7 } %0`
		`' %< POSCOMP ;`

3776B	EQUALPOSCOMP	(comp ob → #pos) (comp ob → #0) POSCOMP with EQUAL as test.
37784	NTHOF	(ob comp → #i) (ob comp → #0) Does SWAP then EQUALPOSCOMP.
0FD006	^ListPos	(ob {} → #i / #0) Equivalent to NTHOF, but faster. However, it only works for lists.
37752	#=POSCOMP	(comp # → #i) (comp # → #0) POSCOMP with #= as test.
05821	SUBCOMP	(comp #m #n → comp') Returns a sub-composite. Makes all index checks first.
376B7	matchob?	(ob comp → T) (ob comp → ob F) Returns TRUE if ob is EQUAL to any element of the composite.

Addr.	Name	Description
371B3	Embedded?	(ob1 ob2 → flag) Returns TRUE if ob2 is embedded in, or is the same as, ob1. Otherwise returns FALSE.

37798	Find1stTrue	(comp test → ob T)
		(comp test → F)
		Tests every element for test. The first one that returns TRUE is put into the stack along with TRUE. If no object returned TRUE, FALSE is put into the stack. For example, the program below returns %-4 and TRUE.
		`:: { %1 %2 %2 %-4 %-5 %6 } ' %0<`
		`   Find1stTrue ;`

377C5	Lookup	(ob test comp → nextob T)
		(ob test comp → ob F)
		Tests every odd element (1,3,...) in the composite. If a test returns TRUE, the object after the tested one is returned, along with TRUE. If no object tests TRUE, FALSE is returned. For example, the program below returns %6 and TRUE.
		`:: %0 ' %<`
		`   { %1 %2 %3 %-4 %-5 %6 }`
		`   Lookup ;`

377DE	Lookup.1	(ob test → nextob T) (ob test → ob F) Return Stack: (comp →) Lookup with the composite already pushed (with >R) onto the runstream. Called by Lookup.
37829	EQLookup	(ob comp → nextob T) (ob comp → ob F) Lookup with EQ as test.

Addr.	Name	Description
37B54	NEXTCOMPOB	(comp #ofs → comp #ofs' ob T) (comp #ofs → comp F) Returns object at specified nibble offset from start. If the object is SEMI (i.e., the end of the composite has been reached) returns FALSE. To get the first element, use FIVE as offset value (to skip the prolog). ZERO works as well.

11.1.2 Building

There are also shortcut words to build lists and secondaries, with specified number of objects, described in the sections below.

Addr.	Name	Description
05459	{}N	(obn..ob1 #n → { obn..ob1 })
05445	::N	(ob1..obn #n → :: ob1..obn ;)
0546D	SYMBN	(ob1..obn #n → symb) Build a symbolic object.
05481	EXTN	(ob1..obn #n → u) Builds a unit object.
293F8	P{}N	(ob1..obn #n → {}) Build list with possible garbage collection.

11.1.3 Exploding

Addr.	Name	Description
054AF	INNERCOMP	(comp → obn..ob1 #n)
3622A	DUPINCOMP	(comp → comp obn..ob1 #n)
3623E	SWAPINCOMP	(comp obj → obj obn..ob1 #n)
35BAF	INCOMPDROP	(comp → obn..ob1)
35C68	INNERDUP	(comp → obn..ob1 #n #n)
2F0EC	ICMPDRPRTDRP	(comp → obn...ob4 ob2 ob1) Does INCOMPDROP then ROTDROP.
3BADA	(INNERCOMP>%)	(comp → obn..ob1 %n)
366E9	INNER#1=	(comp → obn..ob1 flag)

Addr.	Name	Description
157006	^SYMBINCOMP	(symb → ob1 .. obN #n) (ob → ob #1) ({} → {} #1) Explodes symbolic object into meta. Other objects are converted into one-object metas by pushing #1 into the stack.
12A006	^2SYMBINCOMP	(ob1 ob2 → meta1 meta2) Does ^SYMBINCOMP for 2 objects.
158006	^CKINNERCOMP	({} → ob1 .. obN #n) (ob → ob #1) Explodes a list into a meta object. Other objects are converted into one-object metas by pushing #1 into the stack.

11.1.4 Lists

Addr.	Name	Description
055E9	NULL{}	(→ {}) Pushes a null list to the stack.
36ABD	DUPNULL{}?	({} → {} flag)
159006	^DUPCKLEN{}	({} → {} #n) (ob → ob #1) Return length of list, or 1 for non-lists.
29D18	ONE{}N	(ob → { ob })
36202	TWO{}N	(ob1 ob2 → { ob1 ob2 })
36216	THREE{}N	(ob1 ob2 ob3 → { ob1 ob2 ob3 })
361EE	#1-{}N	(ob1..obn #n+1 → {})
2B42A	PUTLIST	(ob #i {} → {}') Replaces object at specified position. Assumes valid #i.
2FC006	^INSERT{}N	({} ob # → {}') Insert object into list at given position. The position must be < than length of the list. If the position is zero, >TCOMP is used.
2FB006	^NEXTPext	(list → list1 list2) Extract in list2 all occurrances of the 1st object of list, the remaining objects are stored in list1. list1 = list-list2.

Addr.	Name	Description
2FD006	ˆCOMPRIMext	({} → {}') Suppress multiple occurrances in the list.
15A006	ˆCKCARCOMP	({} → ob1) (ob → ob) Returns first element for lists, or object itself if it is not a list.
2EF5A	apndvarlst	({} ob → {}') Appends ob to list if not already there.
0FE006	ˆAppendList	({} ob → {}') Equivalent to apndvarlst, but faster.
4EB006	ˆprepvarlist	({} ob → {}') Adds ob at the beginning of the list if not present. If ob is in list, move ob to the beginning of list.
100006	ˆSortList	(L pred → L') Sorts list according to give predicate. Pred is a program that tests two elements and returns FALSE if the first is to appear earlier than the second. To sort in numerical order, for example, the predicate would be a > test.
28A006	ˆPIext	({} → ob) Returns the product of all elements of the list.
25ED3	EqList?	(ob →) Is ob a list of equations? Returns T if ob is a list of at least two elements, and the second element is not a list itself.

11.1.5 Secondaries

Addr.	Name	Description
055FD	NULL::	(→ :: ;) Returns null secondary.
37073	Ob>Seco	(ob → :: ob ;) Does ONE then ::N.
3705A	?Ob>Seco	(ob → :: ob ;) If the object is not a secondary, does Ob>Seco.
37087	2Ob>Seco	(ob1 ob2 → :: ob1 ob2 ;) Does TWO then ::N.

Addr.	Name	Description
3631A	::NEVAL	(ob1..obn #n → ?) Does ::N then EVAL.

Chapter 12
Meta Objects

A meta object (or just meta for short) is a collection of n objects and their count (as a bint). A meta object can be considered as another representation of a composite object. The word `INNERCOMP` will explode any composite into a meta object. The opposite transformation is done by several different words, depending on the kind of composite desired. The available words are listed in section 11.1.2.

Note that a single zero is an (empty) meta object, the null meta object.

It is possible to do several stack operations which treat meta objects as a single object. Generally, the name of these stack operations are in lower case. However, some words have totally misleading names, because their functions are not always used in relation to meta objects, and they were named with their other purpose in mind.

There exist also the user meta objects, which are like meta objects, but the count is represented as a real number and not as a bint. These are not very common, though.

12.1 Reference

12.1.1 Stack Functions

Addr.	**Name**	**Description**
0326E	NDROP	`( meta → )` Should be called `drop`.
37032	DROPNDROP	`( meta ob → )` Should be called `DROPdrop`.
35FB0	#1+NDROP	`( ob meta → )` Should be called `dropDROP`. aka: `N+1DROP`

Addr.	Name	Description
28211	NDROPFALSE	(meta → F) Should be called dropFALSE.
391006	ˆNDROPZERO	(obn..ob1 #n → #0) Replace Meta object with empty Meta object. Should be called dropZERO.
29A5D	psh	(meta1 meta2 → meta2 meta1) Should be called swap.
29A8F	roll2ND	(meta1 meta2 meta3 → meta2 meta3 meta1) Should be called rot.
29B12	unroll2ND	(meta1 meta2 meta3 → meta3 meta1 meta2) Should be called unrot.
3695A	SWAPUnNDROP	(meta1 meta2 → meta2) Should be called swapdrop.
36FA6	metaROTDUP	(meta1 meta2 meta3 → meta2 meta3 meta1 meta1) Should be called rotdup.

12.1.2 Combining Functions

Addr.	Name	Description
296A7	top&	(meta1 meta2 → meta1&meta2)
2973B	pshtop&	(meta1 meta2 → meta2&meta1)
36FBA	ROTUntop&	(meta1 meta2 meta3 → meta2 meta3&meta1)
36FCE	roll2top&	(meta1 meta2 meta3 → meta3 meta1&meta2) aka: rolltwotop&
2963E	psh&	(meta1 meta2 meta3 → meta1&meta3 meta2)

12.1.3 Meta and Object Operations

Addr.	Name	Description
3592B	SWAP#1+	(# ob → ob #+1) aka: SWP1+
34431	DUP#1+PICK	(n..1 #n → n..1 #n n)
34504	get1	(ob meta → meta ob)
36147	OVER#2+UNROL	(meta ob → ob meta)
29693	psh1top&	(meta ob → ob&meta)
28071	pull	(meta&ob → meta ob) aka: #1-SWAP
28085	pullrev	(ob&meta → meta ob)
29821	psh1&	(meta1 meta2 ob → ob&meta1 meta2)
298C0	psh1&rev	(meta1 meta2 ob → ob&meta1 meta2)
2F193	UobROT	(ob meta1 meta2 → meta1 meta2 ob)
29754	pullpsh1&	(meta1 meta2&ob → ob&meta1 meta2)
406006	^addt0meta	(meta1&ob meta2 → meta1 meta2) Removes the last object of meta1.
29972	pshzer	(meta → #0 meta)
36946	SWAPUnDROP	(ob meta → meta)
2F38E	xnsgeneral	(meta → LAM3&meta&LAM1) Uses contents of LAM1 and LAM3.
2F38F	xsngeneral	(meta → meta&LAM3&LAM1) Uses contents of LAM1 and LAM3.

12.1.4 Other Operations

Addr.	Name	Description
3760D	SubMetaOb	(meta #start #end → meta') Gets a sub-meta. Does range checks.

Addr.	Name	Description

37685	SubMetaOb1	`( ob1..obi..obn #n #i #n #i → ob1..obi #n #i )`
		This function can be used to take the first i objects of a meta, if you follow it with SWAPDROP. Example:
		`:: %1 %2 %3 %4 %5 BINT5`
		`BINT3 BINT5 BINT3`
		`SubMetaOb1 ;`
		results in:
		`%1 %2 %3 #5 #3`

33F006	^submeta	`( meta #begin #end → meta' )` Extracts submeta from a meta.

2F356	metatail	`( ob1..obn-i..obn #i #n+1 → ob1..ob..obn-i #n-i obn-i+1..obn #i )`
		#n is the count of the objects in meta. Takes the last #i elements of meta and creates a new one. Example:
		`:: %1 %2 %3 %4 %5`
		`BINT2 BINT6 metatail ;`
		Results:
		`%1 %2 %3 #3 %4 %5 #2`

385006	^metasplit	`( meta #i → meta1 meta2 )` Split a meta in 2 metas at position i. meta1 will contain #i elements meta2 will contain #n-i elements.
39F006	^metaEQUAL?	`( meta2 meta1 → meta2 meta1 flag )` Test equality of 2 metas.
3BF006	^EQUALPOSMETA	`( Meta ob → Meta ob #pos )` Returns last occurrence of ob in Meta. If a component of meta is a list/symb then search if ob is embedded in this component of meta.
3C0006	^EQUALPOS2META	`( Meta2 Meta1 ob → Meta2 Meta1 ob #pos )` Returns last occurrence of ob in Meta1 or in Meta2. #pos is >0 if in meta2, is <0 if in meta1 (#pos=MINUSONE-#).
198006	^METAINT?	`( Meta → Meta flag )` Tests if Meta is an integer.

Addr.	Name	Description
199006	ˆMETAPOSINT?	(Meta → Meta flag) Tests if Meta is a positive integer smaller than Zsmall.

Chapter 13
Unit Objects

Units are another kind of composite objects. It is not really difficult to include one in the program, it is just laborious.

Units start with `UNIT` and end with `;`. Inside, there are commands to define the unit. The best way to understand how a unit is reprsented is by disassembling it. The unit object `9.8_m/s^2` can be created using the code below:

```
::
  UNIT
    % 9.8
    "m"
    "s"
    %2
    um^
    um/
    umEND
  ;
;
```

As you can see, creating units is done in a reverse polish way using the words `um^`, `um*`, `um/` and `umP`. The meaning of the first three ones is easy to guess. The last is used to create prefix operators (kilo, mega, mili, etc.). First enter the prefix as a character or string, and then the unit name (all operations take unit names as characters or strings). Run `umP` and the prefixed unit is created. Then call the other functions as needed. To end a unit, use `umEND`, which joins the number (entered first) to the unit part. The code above could be made shorter if built-in characters and strings (listed on Chapter 6) were used.

Since units are composite objects, you can use, for example, `INNERCOMP` to explode a unit into a meta object. You can also create a unit from a meta object (see Chapter 12), using `EXTN`. The program below, for example, adds the unit m/s to the number in the stack:

```
::
  CK1NOLASTWD
  CKREAL
  "m"
  "s"
  um/
  umEND
  BINT5 EXTN
;
```

Note that the um words, when executed, just put themselves in the stack.

Several operations can be done with units. The complete list is given below. The most important are UM+, UM-, UM*, UM/ and UFACT, whose meanings are obvious; UMCONV, which works like user word CONVERT; UMSI, equivalent to UBASE and U>nbr, which returns the numeric part of a unit.

13.1 Reference

13.1.1 Creating Units

Addr.	Name	Description
2D74F	um*	* marker
2D759	um/	/ marker
2D763	umˆ	ˆ marker
2D76D	umP	Char prefix operator
2D777	umEND	Unit end operator
05481	EXTN	(ob1..obn #n → u) Builds a unit object.

13.1.2 General Functions

Addr.	Name	Description

2F099	U>NCQ	(u → n%% cf%% qhxs)
		Returns the number, conversion factor to base units and a vector in the form:
		`[ kg m A s K cd mol r sr ? ]`
		where each element represents the exponent of that unit. For example, 1_N U>NCQ would return:
		`%%1 %%1 [ 1 1 0 -2 0 0 0 0 0 0 ]`
		since it is equivalent to 1_kg*m/s^2

2F07A	UM>U	(% u → u') Replaces number part of unit.
2F08C	UMCONV	(u1 u2 → u1') Change units of unit1 to units of unit2.
2F090	UMSI	(u → u') Equivalent to user word UBASE.
2F095	UMU>	(u → % u') Returns number and normalized part of unit.
2F019	UNIT>$	(u → $) Converts unit to string.
2F07B	U>nbr	(u → %) Returns number part of unit.
2F098	Unbr>U	(u % → u') Replaces number part of unit.
2F09A	TempConv	??? Used by UMCONV for the conversion of temperature units.
25EE4	KeepUnit	(% ob ob' → % ob) (% ob u → u' ob) If the level one object is a unit object, replaces the numeric part of it with the number on level 3. If not, just DROP.

13.1.3 Arithmetic Functions

Addr.	Name	Description
2F081	UM+	(u u' → u'')
2F082	UM-	(u u' → u'')
2F080	UM*	(u u' → u'')
2F083	UM/	(u u' → u'')
2F07D	UM%	(u %percent → u')
2F07E	UM%CH	(u u' → %)
2F07F	UM%T	(u u' → %)
2F08F	UMMIN	(u u' → u?)
2F08E	UMMAX	(u u' → u?)
2F096	UMXROOT	(u u' → u'')
2F08A	UMABS	(u → u')
2F08B	UMCHS	(u → u')
2F092	UMSQ	(u → u')
2F093	UMSQRT	(u → u')
2D949	UMSIGN	(u → u')
2D95D	UMIP	(u → u')
2D971	UMFP	(u → u')
2D985	UMFLOOR	(u → u')
2D999	UMCEIL	(u → u')
2D9CB	UMRND	(u → u')
2D9EE	UMTRC	(u → u')
2F08D	UMCOS	(u → u')
2F091	UMSIN	(u → u')
2F094	UMTAN	(u → u')

13.1.4 Tests

Addr.	Name	Description
2F087	UM=?	(u u' → %flag)
2F07C	UM#?	(u u' → %flag)
2F086	UM<?	(u u' → %flag)
2F089	UM>?	(u u' → %flag)
2F085	UM<=?	(u u' → %flag)
2F088	UM>=?	(u u' → %flag)

Addr.	Name	Description
2F076	puretemp?	([] []' → [] []' flag) Checks of the two arrays both denote pure temperature units, i.e. if both arrays are equal to [0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]

Chapter 14
Symbolics

Symbolic objects, or algebraic expressions, are another type of composite objects. Their structure is very similar to the units'. They are delimited by SYMBOL and ;. Inside, the expression is created in a reverse polish way.

The disassembly of the equation $R = \dfrac{V}{I}$ should show how to include a symbolic object in your program, should you need one.

```
::
  SYMBOL
    ID R
    ID V
    ID I
    x/
    x=
  ;
;
```

As you have seen, the variables are represented by identifiers, and the functions are the user-accessible ones, whose names are preceded by a lowercase x in System RPL.

To create a symbolic object from a meta, you use the SYMBN function. Note that when you include a funcion, you will have to quote it, ie, put the command ' before the command to put it in the stack instead of executing it. Quoting objects will be explained in more detail in section 19.2.

On the HP49, the new CAS contains most entries dealing with symbolics. These entries are described in the CAS part of the book. mainly in Chapters 44 and 45. However, some entries which were available already on the HP48 have been kept for compatibility reasons. These entries are listed below.

14.1 Reference

14.1.1 General Operations

Addr.	Name	Description
0546D	SYMBN	(ob1..obn #n → sym)
2BD8C	(SYMBN:)	ob1..obn #n -> symb Does 'R, SWAP#1+ then SYMBN. Creates a symbolic from the meta in the stack and the next object in the runstream. This object is added to the end of the symbolic.
286E7	symcomp	(ob → ob') If ob is symbolic, does nothing, otherwise ONE SYMBN.
2F073	SWAPcompSWAP	(ob ob' → ob'' ob') Does SWAP symcomp SWAP.
28ACE	(DROP?symcomp)	(%/C%/Z/id/lam ob' → %/C%/Z/id/lam) (ob ob' → symb) Drop ob'. Then, if the object in the stack is a real, complex, zint, identifier or lam, does nothing. For other objects, calls symcomp to create a one-object symbolics.
293A3	(?symcomp)	(%/C%/Z/id/lam #1 → %/C%/Z/id/lam) (ob #1 → symb) (ob # → symb) If # is BINT1, calls DROP?symcomp. If it is any other number, calls SYMBN.
25EA2	CRUNCH	(ob → %) Internal version of →NUM.
2F110	(FINDVARS)	(sym → {}) Returns a list of the variables of the equation, recursing into programs and functions in the equation.
462006	ˆEQUATION?	(ob → ob flag) Returns TRUE if ob is a symbolic finishing by x=.

Addr.	Name	Description
463006	^USERFCN?	(ob → ob flag) Returns TRUE if ob is a symbolic finishing by xFCNAPPLY.
29CB9	uncrunch	(→) Clears numeric results flag (system flag 3) for the next command only. Example: SYMCOLCT = :: uncrunch colct ;
2BCA2	cknumdsptch1	(sym → symf) Used by one argument functions to evaluate a symbolic or numeric routine according to numeric results flag. Usage: :: cknumdsptch1 <sym> <num> ; If numeric mode, CRUNCH is applied to the level one object and COLA is applied to <num>. If symbolic mode, ckseval1: is called. Example: :: cknumdsptch1 MetaRE xRE ;
2BB21	sscknum2	(sym sym → symf) Used by two argument functions to evaluate function according to current numeric mode. Usage: :: sscknum2 <sym> <num> ; In numeric mode both arguments are CRUNCHed and <num> is COLAd. Else, cksseval2: is called. Example: SYM+ = :: sncknum2 Meta+ x+ ;
2BB3A	sncknum2	(sym % → symf) Usage: :: sncknum2 <sym> <num> ; In symbolic mode uses cksneval2:. Example: SYM+O = :: sncknum2 Meta+Con x+ ;
2BB53	nscknum2	(% sym → symf) Usage: :: nscknum2 <sym> <num> ; In symbolic mode uses cknseval2:. Example: O+SYM = :: nscknum2 Con+Meta x+ ;

14.1.2 Other Functions

Addr.	Name	Description
2EF26	SYMSHOW	(sym id/lam → symf)
2F2A9	XEQSHOWLS	(sym {} → symf)

14.1.3 Meta Symbolics Functions

Addr.	Name	Description
29986	pshzerpsharg	(meta → M_last M_rest) Pushes last sub-expression in meta. If meta is a valid expression M_rest will be empty.
3701E	pZpargSWAPUn	(meta → M_rest M_last) pshzerpsharg then psh.
36FE2	plDRPpZparg	(meta&ob → M_last M_rest) Drops ob then calls pshzerpsharg.
3F1006	^DIVMETAOBJ	(o1...on #n ob → {o1/ob...on/ob}) Division of all elements of a meta by ob. Tests if o=1.

Chapter 15
Graphics Objects (Grobs)

Graphics objects, or grobs for short, represent images, drawings, etc. If you want to write programs that draw something in the screen, then you must know how to use grobs, because the screen content is actually a grob, and you will have to draw on that grob, or to insert another grob in it.

In the reference section below, there are words for creating, manipulating and displaying graphic objects.

When dealing with graphics, keep two things in mind:

1. Several grob operations work directly on the grob without making a copy. So, all pointers to that object in the stack will be modified. You can use the word `CKREF` to ensure an object is unique. For more information on temporary memory and reference counting, see section 24.1.4. This kind of operation is denominated "bang-type", and the commands normally have an exclamation point to indicate that, like `GROB!` or `GROB!ZERO`. These operations also have no error checking, so improper or out-of-range parameters may corrupt memory.

2. The best command to place a grob in the display grob is `XYGROBDISP`. This is because this word checks if the grob to be placed in `HARDBUFF` would exceed its boundaries, and if necessary `HARDBUFF` is enlarged so that the grob fits.

15.1 Reference

15.1.1 Built-in Grobs

Addr.	Name	Description
27AA3	(NULLGROB)	($\rightarrow$ grob) 0x0 Null grob
27D3F	CROSSGROB	($\rightarrow$ grob) 5x5 Cross cursor ("+")
27D5D	MARKGROB	($\rightarrow$ grob) 5x5 Mark symbol ("x")
27D7B	(StdLabelGrob)	21x8 normal menu key
2E25C	(InvLabelGrob)	21x8 inverse menu key
0860B0	~grobAlertIcon	9x9 Alert grob
0870B0	~grobCheckKey	21x8 Check Key menu grob A tickmark and "CHK" in a menu grob.

15.1.2 Dimensions

Addr.	Name	Description
26085	GROBDIM	(grob $\rightarrow$ #height #width)
25EBB	DUPGROBDIM	(grob $\rightarrow$ grob #height #width)
36C68	GROBDIMw	(grob $\rightarrow$ #width)
2F324	CKGROBFITS	(g1 g2 #n #m $\rightarrow$ g1 g2' #n #m) Shrinks g2 if it does not fit in g1.
2F320	CHECKHEIGHT	(grob #height $\rightarrow$) Forces grob (ABUFF/GBUFF) to be at least 64 rows high.

15.1.3 Grob Handling

Addr.	Name	Description
2607B	GROB!	(grob1 grob2 #x #y $\rightarrow$) Stores grob1 into grob2. Bang type.
2EFDB	(GROB+)	(grob1 grob2 $\rightarrow$ grob) Combines two grobs using bitwise OR. Errors when grobs have different sizes.

Addr.	Name	Description
2F342	GROB+#	(flag grob1 grob2 #x #y → grob') Inserts grob2 into the specified position of grob1, using OR (if flag is TRUE) or XOR (if flag is FALSE). Does all necessary checks first.
26080	GROB!ZERO	(grob #x1 #y1 #x2 #y2 → grob') Blanks a rectangular region of the grob. Bang type.
368E7	GROB!ZERODRP	(grob #x1 #y1 #x2 #y2 →) Blanks a rectangular region of the grob. Probably only useful if grob is the text or graphics grob (see section on display-organization). Bang type.
2612F	SUBGROB	(grob #x1 #y1 #x2 #y2 → grob') Returns specified portion of grob.
25F0E	XYGROBDISP	(#x #y grob →) Stores grob in HARDBUFF with upper left corner at (#x,#y). HARDBUFF is expanded if necessary.
25ED8	GROB>GDISP	(grob →) Stores new graph grob.
260B2	MAKEGROB	(#height #width → grob) Creates a blank grob.
2F0DB	MAKEPICT#	(#w #h →) Creates blank graph grob. Minimum size is 131x64. Smaller grobs will be automatically resized.
2609E	INVGROB	(grob → grob') Inverts grob data bits. Bang type.
260E4	PIXON	(#x #y →) Sets pixel in text grob.
260DF	PIXOFF	(#x #y →) Clears pixel in text grob.
260EE	PIXON?	(#x #y → flag) Is pixel in text grob on?
260DA	PIXON3	(#x #y →) Sets pixel in graph grob.
260D5	PIXOFF3	(#x #y →) Clears pixel in graph grob.

Addr.	Name	Description
260E9	PIXON?3	(#x #y → flag) Is pixel in graph grob on?
280C1	ORDERXY#	(#x1 #y1 #x2 #y2 → #x1' #y1' #x2' #y2') Orders the bints to be appropriate for defining a rectangle in a grob. Swaps #x1 and #x2 if #x2<#x1. Swaps #y1 and #y2 if #y2<#y1.
280F8	ORDERXY%	(%x1 %y1 %x2 %y2 → %x1' %y1' %x2' %y2') ORDERXY# with real numbers.
2EF9F	LINEON	(#x1 #y1 #x2 #y2 →) Draws a line in text grob.
2EFA0	LINEOFF	(#x1 #y1 #x2 #y2 →) Clears a line in text grob.
2EFA1	TOGLINE	(#x1 #y1 #x2 #y2 →) Toggles a line in text grob.
2EFA2	LINEON3	(#x1 #y1 #x2 #y2 →) Draws a line in graph grob.
2F13F	DRAWLINE#3	(#x1 #y1 #x2 #y2 →) Draws a line in graph grob. x1<x2 is not required.
2EFA3	LINEOFF3	(#x1 #y1 #x2 #y2 →) Clears a line in graph grob.
2EFA4	TOGLINE3	(#x1 #y1 #x2 #y2 →) Toggles a line in graph grob.
2F382	TOGGLELINE#3	(#x1 #y1 #x2 #y2 →) Toggles line in graph grob. x1<x2 is not required.
2F32C	DRAWBOX#	(#x1 #y1 #x2 #y2 →) Draws rectangle in graph grob.
2EF03	DOLCD>	(→ grob) Returns current display.
2EF04	DO>LCD	(grob →) Grob to display.
0BF007	^GROBADDext	(grob2 grob1 → grob) Vertical grob addition. grob2 will be above grob1.

15.1.4 Greyscale Graphics

Addr.	Name	Description
25592	SubRepl	(grb1 grb2 #x1 #y1 #x2 #y2 #W #H → grb1') Replace a part of grb1 with a part of grb2 in REPLACE mode.
25597	SubGor	(grb1 grb2 #x1 #y1 #x2 #y2 #W #H → grb1') Replace a part of grb1 with a part of grb2 in OR mode.
2559C	SubGxor	(grb1 grb2 #x1 #y1 #x2 #y2 #W #H → grb1') Replace a part of grb1 with a part of rgb2 in XOR mode.
25565	LineW	(grb #x1 #y1 #x2 #y2 → grb') Draw a white line.
2556F	LineG1	(grb #x1 #y1 #x2 #y2 → grb') Draw a light grey line.
25574	LineG2	(grb #x1 #y1 #x2 #y2 → grb') Draw a dark grey line.
2556A	LineB	(grb #x1 #y1 #x2 #y2 → grb') Draw a black line.
25579	LineXor	(grb #x1 #y1 #x2 #y2 → grb') XOR a line.
2F218	CircleW	(grb #Cx #Cy #r → grb') Draw a white circle.
2F216	CircleG1	(grb #Cx #Cy #r → grb') Draw a light grey circle.
2F217	CircleG2	(grb #Cx #Cy #r → grb') Draw a dark grey circle.
2F215	CircleB	(grb #Cx #Cy #r → grb') Draw a black circle
2F219	CircleXor	(grb #Cx #Cy #r → grb') XOR a circle.
2557E	Sub	(grb #x1 #y1 #x2 #y2 → grb' flag) Get a part of a grob.
25583	Repl	(grb1 grb2 #x #y → grb1') Copy grb2 into grb1 in REPLACE mode.

Addr.	Name	Description
25588	Gor	(grb1 grb2 #x #y → grb1') Copy grb2 into grb1 in OR mode.
2558D	Gxor	(grb1 grb2 #x #y → grb1') Copy grb2 into grb1 in XOR mode.
255A1	Grey?	(grob → flag) Is grob a Greyscale Grob?
255B0	ScrollVGrob	(grb #W #X #Yd #Ys #h → grb') Scroll up and down a portion of a graphical object.
255BA	PixonW	(grb #x #y → grb') Make a pixel white.
255C4	PixonG1	(grb #x #y → grb') Make a pixel light grey.
255C9	PixonG2	(grb #x #y → grb') Make a pixel dark grey.
255BF	PixonB	(grb #x #y → grb') Make a pixel black.
255CE	PixonXor	(grb #x #y → grb') Apply XOR to a pixel.
255D3	FBoxW	(grb #x1 #y1 #x2 #y2 → grb') Make a white filled rectangle.
255D3	FBoxG1	(grb #x1 #y1 #x2 #y2 → grb') Make a light grey filled rectangle.
255D8	FBoxG2	(grb #x1 #y1 #x2 #y2 → grb') Make a dark grey filled rectangle.
255DD	FBoxB	(grb #x1 #y1 #x2 #y2 → grb') Make a black filled rectangle.
255E2	FBoxXor	(grb #x1 #y1 #x2 #y2 → grb') Apply XOR to a filled rectangle.
255E7	LBoxW	(grb #x1 #y1 #x2 #y2 → grb') Draw a white rectangle.
255EC	LBoxG1	(grb #x1 #y1 #x2 #y2 → grb') Draw a light grey rectangle.
255F1	LBoxG2	(grb #x1 #y1 #x2 #y2 → grb') Draw a dark grey rectangle.
255F6	LBoxB	(grb #x1 #y1 #x2 #y2 → grb') Draw a black rectangle.
255FB	LBoxXor	(grb #x1 #y1 #x2 #y2 → grb') Apply XOR to a rectangle.

Addr.	Name	Description
2F21B	ToGray	(grb → grb'/grb) Convert a B&W grob to Greyscale.
2F21A	Dither	(grb → grb'/grb) Convert a greyscale grob to B&W
255B5	Distance	(#Δx #Δy → #SQRT(Δx^2+Δy^2)) Compute the distance between two points.

15.1.5 Creating Menu Label Grobs

Addr.	Name	Description
2E166	MakeStdLabel	($ → grob) Makes standard menu label.
2E189	MakeBoxLabel	($ → grob) Makes label with a box.
2E1EB	MakeDirLabel	($ → grob) Makes directory label.
2E24D	MakeInvLabel	($ → grob) Makes inverse label.
25E7F	Box/StdLabel	($ flag → grob) If TRUE makes box label, otherwise makes standard label.
25F01	Std/BoxLabel	($ flag → grob) If TRUE makes standard label, otherwise makes box label.
25E80	Box/StdLbl:	(→ grob) Does Box/StdLabel with the next two objects from the stream. Usage: :: Box/StdLbl: $ <test> ;
2E0D5	Grob>Menu	(#col grob →) Displays grob as menu label.
2E0F3	Str>Menu	(#col $ →) Displays string as menu label.
2E11B	Id>Menu	(#col id →) Displays id as menu label.
2E107	Seco>Menu	(#col :: →) Does EVAL then DoLabel.

Addr.	Name	Description
25886	DoLabel	(#col ob →) If ob is of one of the supported types, displays a menu label. If not, generates a "Bad Argument Type" error.

15.1.6 Converting Strings to Grobs

Addr.	Name	Description
25F7C	$>GROB	($ → grob) Makes grob of the string using the system font. Linefeed does *not* make new line.
25F86	$>GROBCR	($ → grob) Makes grob of the string using the system font. Linefeed *does* make new line.
25F81	$>grob	($ → grob) Makes grob of the string using the minifont. Linefeed does *not* make new line.
25F8B	$>grobCR	($ → grob) Makes grob of the string using the minifont. Linefeed *does* make new line.
05F0B3	(~$>grobOrGROB)	($ → grob) Converts string to a grob using either the current font or the minifont, depending on system flag 90.
25F24	RIGHT$3x6	($ #n → flag grob) Transforms string into grob (using the minifont), then takes all characters starting after column #n. flag is FALSE if #n is greater than the width of the grob. In this case, the whole grob is returned.
25FEF	CENTER$3x5	(grob #x #y $ #w → grob') Creates grob from string (using the minifont) and embeds it at specified position (#x, #y). The grob is centered around #x and the to is put at #y. #w represents the maximum width of the grob created. If the text is wider, it is truncated. Bangtype.

Addr.	Name	Description
2E2AA	MakeLabel	($ #w #x grob → grob') Inserts $ into grob using CENTER$3x5 with y=5.
25FF9	LEFT$3x5	(grob #x #y $ #w → grob') Like CENTER$3x5, but the left corner of the text is positioned at #x.
26071	ERASE&LEFT$3x5	(grob #x #y $ #w → grob') Like LEFT$3x5, but erase background first.
26008	LEFT$3x5Arrow	(grob #x #y $ #w → grob') Like LEFT$3x5, but if the text does not fit, replace the last character by character 31 (dots) to show that the text was truncated.
2601C	LEFT$3x5CR	(grob #x #y $ #w #h → grob') Like LEFT$3x5, but newlines in the strings are interpreted and start new lines. Note the additional argument #h for the maximum height of the text grob.
26012	LEFT$3x5CRArrow	(grob #x #y $ #w #h → grob') Like LEFT$3x5CR, but show truncation with arrows.
25FF4	CENTER$5x7	(grob #x #y $ #w → grob') Same as CENTER$3x5, but using system font.
25FFE	LEFT$5x7	(grob #x #y $ #w → grob') Like CENTER$5x7, but the left corner of the text is positioned at #x.
2606C	ERASE&LEFT$5x7	(grob #x #y $ #w → grob') Like LEFT$5x7, but erase background first.
26003	LEFT$5x7Arrow	(grob #x #y $ #w → grob') Like LEFT$5x7, but if the text has to be truncated, replace the last character with character 31 (arrow).
26017	LEFT$5x7CR	(grob #x #y $ #w → grob') Like LEFT$5x7, but interpret newlines.
2600D	LEFT$5x7CRArrow	(grob #x #y $ #w → grob') Like LEFT$5x7CR, but show truncation with arrows.

15.1.7 Creating Grobs from Other Objects

Addr.	Name	Description
019004	^EQW3GROB	(ob → ext grob #0) (ob → #2)
01A004	^EQW3GROBStk	(ob → ext grob #0) (ob → #2)
01F004	^EQW3GROBmini	(ob → ext grob #0) (ob → #2)
01E004	^EQW3GROBsys	(ob → ext grob #0) (ob → #2)
0BE007	^XGROBext	(ob → grob) Convert object to a grob.
0C0007	^DISPLAYext	(grob ob → grob') Adds ob to grob after converting it to a grob.

Chapter 16
Library and Backup Objects

Libraries are very complex objects that hold a collection of commands. Some of these commands are named and accessible to the user, but some have no names, and so are “hidden”. Backup objects are used by the HP49 to store the contents of the entire HOME directory and restore it later. The integrity of both objects can be verified because both have a CRC code attached to them.

A rompointer (sometimes called XLIB name) is a pointer to a command in a library. The only way to access a unnamed command in a library is through a rompointer. They hold the number (often called id) of the library and the number of the command.

To insert a rompointer in your program, use the following structure: ROMPTR <lib> <cmd>, where <lib> is the number of the library, and <cmd> is the number of the command. Both numbers are specified in hexadecimal form. Rompointers are always automatically executed (like identifiers), so you have to quote them (see section 19.2) if you want one in the stack.

16.1 Reference

16.1.1 Port Operations

Addr.	Name	Description
25EEB	NEXTLIBBAK	(#addr → backup/library #nextaddr) Gets next library or backup.

16.1.2 Rompointers

Addr.	Name	Description
07E50	#>ROMPTR	(#lib #cmd → ROMPTR) Creates rompointer.
08CCC	ROMPTR>#	(ROMPTR → #lib #cmd) Splits rompointer.
07E99	ROMPTR@	(ROMPTR → ob T) (ROMPTR → F) Recalls contents of rompointer.
35C40	DUPROMPTR@	(ROMPTR → ROMPTR ob T) (ROMPTR → ROMPTR F) Does DUP then ROMPTR@.
35A88	?>ROMPTR	(ob → ob') If ROM-WORD? and TYPECOL? then RPL@.
35AAB	?ROMPTR>	(ob → ob') If TYPEROMP? and content exists INHARDROM? then return contents.
35BFF	RESOROMP	(→ ob) Recalls contents of next object in the runstream (which must be a rompointer).
34FCD	ROM-WORD?	(ob → flag)
34FC0	DUPROM-WORD?	(ob → ob flag)

16.1.3 Libraries

Addr.	Name	Description
07709	TOSRRP	(# →) Attaches library to HOME directory.
076AE	OFFSRRP	(# →) Detaches library from HOME directory.
2F2A7	XEQSETLIB	(% →) Internal ATTACH.
07638	SETHASH	(hxs #libnum →) Buggy?

16.1.4 Backup Objects

Addr.	Name	Description
081D9	BAKNAME	(bak → id T) Returns backup's name
0905F	BAK>OB	(bak → ob) Gets backup object.

Part II

General System RPL Entries

Chapter 17
Stack Operations

In System RPL, using the stack is almost the same as in User RPL. The basic operations are the same, except for little changes in the name: `DUP`, `2DUP` (equivalent to User RPL's `DUP2`), `NDUP` (`DUPN`), `DROP`, `2DROP` (`DROP2`), `NDROP` (`DROPN`), `OVER`, `PICK`, `SWAP`, `ROLL`, `UNROLL` (`ROLLD`), `ROT`, `UNROT` and `DEPTH`.

All commands that require or return a numeric argument use bints and not real numbers, unless otherwise noted.

There are many commands that do two or even three operations in sequence. They are listed in the reference section. The table below lists some useful combinations in a nice form:

	DUP	**DROP**	**SWAP**	**OVER**	**ROT**	**UNROT**
DUP	DUPDUP	DROPDUP	SWAPDUP	OVERDUP	ROTDUP	UNROTDUP
DROP	-	2DROP	SWAPDROP	-	ROTDROP	UNROTDROP
SWAP	-	DROPSWAP	-	OVERSWAP	ROTSWAP	UNROTSWAP
OVER	DUPDUP	DROPOVER	SWAPOVER	2DUP	ROTOVER	UNROTOVER
ROT	DUPROT	DROPROT	SWAPROT	-	UNROT	-
UNROT	DUPUNROT	-	ROTSWAP	OVERUNROT	-	ROT
SWAPDROP	-	DROPSWAPDROP	-	DROPDUP	DROPSWAP	UNROTSWAPDRO
DROPDUP	-	-	SWAPDROPDUP	-	-	-
DROPSWAP	-	-	SWAPDROPSWAP	-	ROTDROPSWAP	SWAPDROP
2DROP	-	3DROP	-	-	ROT2DROP	UNROT2DROP
2DUP	-	-	SWAP2DUP	-	ROT2DUP	-
3PICK	DUP3PICK	-	SWAP3PICK	OVERDUP	-	-
4PICK	-	-	SWAP4PICK	-	-	-
5PICK	-	-	-	OVER5PICK	-	-
4ROLL	-	-	SWAP4ROLL	-	-	-
4UNROLL	DUP4UNROLL	-	-	-	-	-
ROT2DROP	-	-	ROTROT2DROP	SWAPDROP	ROTROT2DROP	-

17.1 Reference

In this section, the numbers 1, 2… n are used to represent different objects, not necessarily any kind of number.

Addr.	Name	Description
03188	DUP	(ob → ob ob)
35CE0	DUPDUP	(ob → ob ob ob)
2D5006	^3DUP	(3 2 1 → 3 2 1 3 2 1)
28143	NDUPN	(ob #n → ob..ob #n)
		(ob #0 → #0)
35FF3	DUPROT	(1 2 → 2 2 1)
3457F	DUPUNROT	(1 2 → 2 1 2)
		aka: SWAPOVER
36133	DUPROLL	(1..n #n → 1 3..n #n 2)
3432C	DUP4UNROLL	(1 2 3 → 3 1 2 3)
3611F	DUPPICK	(n..1 #n → n..1 #n n-1)
35D30	DUP3PICK	(1 2 → 1 2 2 1)
		aka: 2DUPSWAP
34431	DUP#1+PICK	(n..1 #n → n..1 #n n)
031AC	2DUP	(1 2 → 1 2 1 2)
35D30	2DUPSWAP	(1 2 → 1 2 2 1)
		aka: DUP3PICK
36CA4	2DUP5ROLL	(1 2 3 → 2 3 2 3 1)
031D9	NDUP	(1..n #n → 1..n 1..n)
03244	DROP	(1 →)
357CE	DROPDUP	(1 2 → 1 1)
37032	DROPNDROP	(1..n #n ob →)
35733	DROPSWAP	(1 2 3 → 2 1)
3574D	DROPSWAPDROP	(1 2 3 → 2)
		aka: ROT2DROP, XYZ>Y
36007	DROPROT	(1 2 3 4 → 2 3 1)
3606B	DROPOVER	(1 2 3 → 1 2 1)
03258	2DROP	(1 2 →)
341D2	3DROP	(1 2 3 →)
		aka: XYZ>
341D7	4DROP	(1..4 →)
		aka: XYZW>
341DC	5DROP	(1..5 →)
341E8	6DROP	(1..6 →)
341F4	7DROP	(1..7 →)
0326E	NDROP	(1..n #n →)
35FB0	#1+NDROP	(ob 1..n #n →)
		aka: N+1DROP

Addr.	Name	Description
2F0A1	RESETDEPTH	(ob1..obn obn+1..obx #n → ob1..obn) Drops all but #n levels of the stack.
0314C	DEPTH	(1..n → 1..n #n)
28187	reversym	(1..n #n → n..1 #n)
03223	SWAP	(1 2 → 2 1)
3576E	SWAPDUP	(1 2 → 2 1 1)
368B5	SWAP2DUP	(1 2 → 2 1 2 1)
3421A	SWAPDROP	(1 2 → 2) aka: XY>Y
35857	SWAPDROPDUP	(1 2 → 2 2)
35872	SWAPDROPSWAP	(1 2 3 → 3 1) aka: UNROTDROP, XYZ>ZX
341BA	SWAPROT	(1 2 3 → 3 2 1) aka: UNROTSWAP, XYZ>ZYX
36C90	SWAP4ROLL	(1 2 3 4 → 2 4 3 1) aka: XYZW>YWZX
3457F	SWAPOVER	(1 2 → 2 1 2) aka: DUPUNROT
36CB8	SWAP3PICK	(1 2 3 → 1 3 2 1)
35018	2SWAP	(1 2 3 4 → 3 4 1 2)
03295	ROT	(1 2 3 → 2 3 1)
3579C	ROTDUP	(1 2 3 → 2 3 1 1)
35CA4	ROT2DUP	(1 2 3 → 2 3 1 3 1)
341A8	ROTDROP	(1 2 3 → 2 3) aka: XYZ>YZ
3574D	ROT2DROP	(1 2 3 → 2) aka: DROPSWAPDROP, XYZ>Y
34195	ROTDROPSWAP	(1 2 3 → 3 2) aka: XYZ>ZY
3416E	ROTSWAP	(1 2 3 → 2 1 3) aka: XYZ>YXZ
343BD	ROTROT2DROP	(1 2 3 → 3) aka: UNROT2DROP, XYZ>Z
35CCC	ROTOVER	(1 2 3 → 2 3 1 3)
3423A	4ROLL	(1 2 3 4 → 2 3 4 1) aka: FOURROLL, XYZW>YZWX
3588B	4ROLLDROP	(1 2 3 4 → 2 3 4)
35F06	4ROLLSWAP	(1 2 3 4 → 2 3 1 4)

Addr.	Name	Description
36043	4ROLLROT	(1 2 3 4 → 2 4 1 3) aka: FOURROLLROT
360E3	4ROLLOVER	(1 2 3 4 → 2 3 4 1 4)
34257	5ROLL	(1 2 3 4 5 → 2 3 4 5 1) aka: FIVEROLL
358A7	5ROLLDROP	(1 2 3 4 5 → 2 3 4 5)
34281	6ROLL	(1..6 → 2..6 1) aka: SIXROLL
342EA	7ROLL	(1..7 → 2..7 1) aka: SEVENROLL
342BB	8ROLL	(1..8 → 2..8 1) aka: EIGHTROLL
03325	ROLL	(1..n #n → 2..n 1)
35FC4	ROLLDROP	(1..n #n → 2..n)
35D80	ROLLSWAP	(1..n #n → 2..n-1 1 n)
344F2	#1+ROLL	(ob 1..n #n → 1..n ob)
34517	#2+ROLL	(a b 1..n #n → b 1..n a)
2D6006	^#3+ROLL	(obn+3...obn...ob1 #n → obn+2...ob1 obn+3)
344DD	#+ROLL	(1..n+m #n #m → 2..n+m 1)
344CB	#-ROLL	(1..n-m #n #m → 2..n-m 1)
3422B	UNROT	(1 2 3 → 3 1 2) aka: 3UNROLL, XYZ>ZXY
35D1C	UNROTDUP	(1 2 3 → 3 1 2 1)
35872	UNROTDROP	(1 2 3 → 3 1) aka: SWAPDROPSWAP, XYZ>ZX
343BD	UNROT2DROP	(1 2 3 → 3) aka: ROTROT2DROP, XYZ>Z
341BA	UNROTSWAP	(1 2 3 → 3 2 1) aka: SWAPROT, XYZ>ZYX
360CF	UNROTOVER	(1 2 3 → 3 1 2 1)
3422B	3UNROLL	(1 2 3 → 3 1 2) aka: UNROT, XYZ>ZXY
34331	4UNROLL	(1 2 3 4 → 4 1 2 3) aka: FOURUNROLL, XYZW>WXYZ
35D44	4UNROLLDUP	(1 2 3 4 → 4 1 2 3 3)
343CF	4UNROLL3DROP	(1 2 3 4 → 4) aka: XYZW>W
36057	4UNROLLROT	(1 2 3 4 → 4 3 2 1)

Addr.	Name	Description
34357	5UNROLL	(1 2 3 4 5 → 5 1 2 3 4) aka: FIVEUNROLL
3438D	6UNROLL	(1..6 → 6 1..5) aka: SIXUNROLL
35BEB	7UNROLL	(1..7 → 7 1..6)
3615B	8UNROLL	(1..8 → 8 1..7)
28225	(9UNROLL)	(1..9 → 9 1..8)
3616F	10UNROLL	(1..10 → 10 1..9)
0339E	UNROLL	(1..n #n → n 1..n-1)
34552	#1+UNROLL	(ob 1..n #n → n ob 1..n-1)
34564	#2+UNROLL	(a b 1..n #n → n a b 1..n-1)
3453D	#+UNROLL	(1..n+m #n #m → n+m 1..n+m-1)
3452B	#-UNROLL	(1..n-m #n #m → n-m 1..n+m-1)
032C2	OVER	(1 2 → 1 2 1)
35CF4	OVERDUP	(1 2 → 1 2 1 1)
35D6C	OVERSWAP	(1 2 → 1 1 2) aka: OVERUNROT
35D6C	OVERUNROT	(1 2 → 1 1 2) aka: OVERSWAP
36CF4	OVER5PICK	(1 2 3 4 → 1 2 3 4 3 1)
37046	2OVER	(1 2 3 4 → 1 2 3 4 1 2)
34485	3PICK	(1 2 3 → 1 2 3 1)
35F1A	3PICKSWAP	(1 2 3 → 1 2 1 3)
360F7	3PICKOVER	(1 2 3 → 1 2 3 1 3)
36CCC	3PICK3PICK	(1 2 3 → 1 2 3 1 2)
2F1C6	DROP3PICK	(1 2 3 4 → 1 2 3 1)
3448A	4PICK	(1 2 3 4 → 1 2 3 4 1)
35F2E	4PICKSWAP	(1 2 3 4 → 1 2 3 1 4)
36CE0	SWAP4PICK	(1 2 3 4 → 1 2 4 3 1)
3610B	4PICKOVER	(1 2 3 4 → 1 2 3 4 1 4)
3448F	5PICK	(1 2 3 4 5 → 1 2 3 4 5 1)
34494	6PICK	(1..6 → 1..6 1)
34499	7PICK	(1..7 → 1..7 1)
3449E	8PICK	(1..8 → 1..8 1)
344A3	(9PICK)	(1..9 → 1..9 1)
344A8	(10PICK)	(1..10 → 1..10 1)
032E2	PICK	(1..n #n → 1..n 1)
34436	#1+PICK	(1..n #n-1 → 1..n 1)
34451	#2+PICK	(1..n #n-2 → 1..n 1)

Addr.	Name	Description
34465	#3+PICK	(1..n #n-3 → 1..n 1)
34474	#4+PICK	(1..n #n-4 → 1..n 1)
34417	#+PICK	(1..n+m #n #m → 1..n+m 1)
34405	#-PICK	(1..n-m #n #m → 1..n-m 1)

Chapter 18
Temporary Environments

System RPL local variables (also known as temporary or lambda variables) work in the same way and have the same uses as in User RPL. You assign values to them, and these values can be recalled or changed while the variables exist. The stored values are referenced by means of local identifiers (also called lambda identifiers, or lams for short). These are very similar to the global identifiers that reference variables stored in memory (see Chapter 8), but the variables exist only temporarily.

But there is one difference: in System RPL you can give a null (that is, empty) name to local variables, therefore effectively making them unnamed variables. This saves memory and is much faster. But before learning how to create and use unnamed local variables, let us learn how to use normal, named ones.

18.1 Named Local Variables

Creating named local variables is very similar to creating temporary variables in User RPL. You have to create a list of local identifiers (called lams for short), and run the command `BIND`. To recall the contents of one of them, just enter its local identifier. To store a new value, put that value and the lam in the stack, and run `STO`. To remove the local variables from memory, use `ABND` (shortcut for “abandon”). The code is not checked for matching `BIND`/`ABND`, so you may include them in different programs if you wish. But this also means you must be sure to have an `ABND` for each `BIND`.

Here is a little program that creates two local variables, recalls their contents and assigns new values for them (it is called `LAM1`):

```
::
  %2 %3
  {
    LAM first
```

```
        LAM sec
      }
      BIND        (first contains 2, and sec 3)
      LAM first   (recall contents from first - 2)
      LAM sec     (recall contents from sec - 3)
      DUP
      ' LAM first
      STO         (store new contents in first)
      %+          (results 5)
      ' LAM sec
      STO         (store sum in sec)
      ABND        (delete variables from memory)
    ;
```

18.2 Unnamed Local Variables

As said above, you can use unnamed local variables. Technically, they have a name: the null, or empty, name; but all "unnamed" variables have the same name. Since they cannot be identifed by name, positional syntax is necessary. The above program could be rewritten using null named temporary variables this way (now called `LAM2`):

```
::
  %2 %3
  { NULLLAM NULLLAM }
  BIND
  2GETLAM     (recalls 2)
  1GETLAM     (recalls 3)
  DUP
  2PUTLAM
  %+
  1PUTLAM
  ABND
;
```

The numbering is done in the same order as the stack levels: `1GETLAM` contains what was on level one, `2GETLAM` contains what was on level two, etc. There are supported entries to recall and store directly up to the 22^{nd} variable (`1GETLAM` to `22GETLAM`, and their `PUTLAM` equivalents). To access variables with numbers higher than 23 (which probably will not happen very often), use `GETLAM`, which takes a bint representing the variable number and returns its contents; and PUTLAM, which takes an object and the variable number, and stores that object in the specified variable.

18.3 Nested Temporary Environments

It is perfectly possible to use two or more temporary environments at the same time. Nothing special needs to be done during the creation: just use another DOBIND or BIND before abandoning the previous one. When an ABND is found, it always refers to the most recent BIND.

If you only use named lams, nothing special needs to be done. There will be no confusion with names, unless you redefine an existing variable (but doing this will only make a great mess out of your program). However, when at least one of the temporary environments has unnamed lams, you must pay attention to the numbering.

Note that the GETLAM words do not necessarily refer to unnamed local variables: 1GETLAM recalls the most recently bound variable, 2GETLAM the one before that, and so on. (When binding lams, the binding starts at the stack level with the largest number, working towards the one with the smallest number, so that the last bound variable is the one whose contents where in level one.) You may use the GETLAM words also to access named lams.

Due to the nature of temporary environments, there appears to be an extra local variable (before all the others) for internal housekeeping purposes. To access the unnamed lams of a previous environment, you must add the number of variables bound in the current environment *plus one* to the number you would have used before the second binding.

The following program (named LAM3) will try to make the above explanation clearer:

```
::
  %2
  %1
  {
    LAM n2
    LAM n1
  }
  BIND
  1GETLAM   (Returns 1)
  2GETLAM   (Returns 2)
```

```
    %4
    %3
    {
      NULLLAM
      NULLLAM
    }
    BIND
    1GETLAM   (Returns 3)
    2GETLAM   (Returns 4)
    4GETLAM   (Returns 1)
    5GETLAM   (Returns 2)
    ABND
    ABND
  ;
```

First, this program binds 2 to `n2` and 1 `n1`, but these names are never used. Instead, `1GETLAM` is used to acces the most recently bound value, that is, 1, which could also be accessed via `LAM n1`. Following, `2GETLAM` returns the next-to-last value, or 2.

Things become more complicated when another environment is bound, this time to unnamed lams. Now `1GETLAM` returns 3, which belongs to the new environment, and was the last bound variable. Similarly, `2GETLAM` also returns a variable bound in this second batch.

If we wanted to access the variable that previously was number one, we need to add the number of variables bound in the new environment (that is, two) plus one (the housekeeping pseudo-variable) to the previous number. So, to get what `1GETLAM` would have returned before, we add three to one, obtaining `4GETLAM`. And this returns, as expected, 1. Similarly, `5GETLAM` returns 2, the same `2GETLAM` had returned before the second binding.

Naturally, after the first `ABND` (corresponding to the binding of values 4 and 3), `1GETLAM` and `2GETLAM` would again return 1 and 2, respectively.

If you have been able to understand the above, you will not have problems to nest temporary environments when necessary.

18.4 Other Ways of Binding

First, instead of a list of lams, you can always put each lam in the stack, followed by the number of variables to be bound, and run the command `DOBIND` instead of `BIND`. As a matter of fact, `BIND` is just `:: INNERCOMP DOBIND ;`.

When you are binding a great number of local variables, instead of entering the following code (which takes 67.5 bytes)

```
...
{ NULLLAM NULLLAM NULLLAM NULLLAM NULLLAM NULLLAM
  NULLLAM NULLLAM NULLLAM NULLLAM NULLLAM NULLLAM
  NULLLAM NULLLAM NULLLAM NULLLAM NULLLAM NULLLAM
  NULLLAM NULLLAM NULLLAM NULLLAM NULLLAM NULLLAM }
BIND
...
```

use this, which takes only 12.5 bytes, a savings of 55 bytes:

```
... NULLLAM TWENTYFOUR NDUPN {}N BIND ...
```

However, why create a composite if it is going to be exploded later? Replace `{}N BIND` for `DOBIND`, and save 2.5 more bytes.

Or you can also use `TWENTYFOUR ' NULLLAM CACHE`. However, if you use this, an extra variable is created to hold the count, so you must add one to the variable positions of the previous examples.

When decompiling code, you can sometimes find things like

```
... ZEROZEROZERO BINT3 DOBIND ...
```

which is yet another way of binding three null named variables. This works because instead of `NULLLAM`, any fixed address ROM object can be used, as `ZERO` in this example.

The following constructs are the most compact ways to create temporary environments for N null named variables.

N	**Commands to create N null named variables**
1	`1LAMBIND`
2	`ZEROZEROTWO DOBIND`
2	`FPTR2 ^2LAMBIND`
3	`FPTR2 ^3LAMBIND`
4	`4NULLLAM{} BIND`
N	`NULLLAM #N NDUPN DOBIND`

18.5 Reference

18.5.1 Builtin IDs and LAMs

Addr.	Name	Description
272FE	NULLID	(→ id) Null (empty) identifier
2B3AB	NULLLAM	(→ lam) Puts NULLLAM in the stack.
27155	'IDX	(→ id) Puts ID X unevaluated on the stack.
272F3	(ID_EQ)	ID EQ
27937	(ID_SIGMADAT)	ID ΣDAT

18.5.2 Conversion

Addr.	Name	Description
05B15	$>ID	($ → ID)
362DE	DUP$>ID	($ → $ ID)

18.5.3 Temporary Environments Words

Addr.	Name	Description
074D0	BIND	(obn..ob1 {lamn..lam1} →) Binds n objects to n differently named lams.
074E4	DOBIND	(obn..ob1 lamn..lam1 #n →) Binds n objects to n differently named lams.
36518	1LAMBIND	(ob →) Binds one object to a null named lam.
36513	DUP1LAMBIND	(ob → ob) Does DUP then 1LAMBIND.
155006	^2LAMBIND	(ob1 ob2 →) Binds two objects to null named lams.
156006	^3LAMBIND	(ob1 ob2 ob3 →) Binds three objects to null named lams.

Addr.	Name	Description
0DE0B0	˜nNullBind	(obn..ob1 #n →) Binds #n objects to null named lams. 1LAM has the count, 2LAM the first object. Decompiles to `:: ' NULLLAM CACHE ;`
36A77	dvarlsBIND	(ob →) Binds ob to LAM 'dvar.
07497	ABND	(→) Abandons topmost temporary environment.
34D00	CACHE	(obn..ob1 #n lam →) Binds all objects under the same name. 1LAM has the count.
34EBE	DUMP	(NULLLAM → ob1..obn #n) Inverse of CACHE. Always does garbage collection.
34D58	SAVESTACK	(→) Caches stack to SAVELAM.
34FA6	undo	(→) Dumps SAVELAM.
07943	@LAM	(lam → ob T) (lam → F) Tries recalling object from lam. If successful, returns object and TRUE, otherwise returns just FALSE.
07D1B	STOLAM	(ob lam →) Tries storing object in lam. Generates "Undefined Local Name" error if lam is not found.
075A5	GETLAM	(#n → ob) Gets contents of nth topmost lam.
34616	1GETLAM	(→ ob)
34620	2GETLAM	(→ ob)
3462A	3GETLAM	(→ ob)
34634	4GETLAM	(→ ob)
3463E	5GETLAM	(→ ob)
34648	6GETLAM	(→ ob)
34652	7GETLAM	(→ ob)
3465C	8GETLAM	(→ ob)
34666	9GETLAM	(→ ob)
34670	10GETLAM	(→ ob)

Addr.	Name	Description
3467A	11GETLAM	(→ ob)
34684	12GETLAM	(→ ob)
3468E	13GETLAM	(→ ob)
34698	14GETLAM	(→ ob)
346A2	15GETLAM	(→ ob)
346AC	16GETLAM	(→ ob)
346B6	17GETLAM	(→ ob)
346C0	18GETLAM	(→ ob)
346CA	19GETLAM	(→ ob)
346D4	20GETLAM	(→ ob)
346DE	21GETLAM	(→ ob)
346E8	22GETLAM	(→ ob)
346F2	(23GETLAM)	(→ ob)
346FC	(24GETLAM)	(→ ob)
34706	(25GETLAM)	(→ ob)
34710	(26GETLAM)	(→ ob)
3471A	(27GETLAM)	(→ ob)
075E9	PUTLAM	(ob #n →) Stores new contents to nth topmost lam.
34611	1PUTLAM	(ob →)
3461B	2PUTLAM	(ob →)
34625	3PUTLAM	(ob →)
3462F	4PUTLAM	(ob →)
34639	5PUTLAM	(ob →)
34643	6PUTLAM	(ob →)
3464D	7PUTLAM	(ob →)
34657	8PUTLAM	(ob →)
34661	9PUTLAM	(ob →)
3466B	10PUTLAM	(ob →)
34675	11PUTLAM	(ob →)
3467F	12PUTLAM	(ob →)
34689	13PUTLAM	(ob →)
34693	14PUTLAM	(ob →)
3469D	15PUTLAM	(ob →)
346A7	16PUTLAM	(ob →)
346B1	17PUTLAM	(ob →)
346BB	18PUTLAM	(ob →)
346C5	19PUTLAM	(ob →)
346CF	20PUTLAM	(ob →)

Addr.	Name	Description
346D9	21PUTLAM	(ob →)
346E3	22PUTLAM	(ob →)
346ED	(23PUTLAM)	(ob →)
346F7	(24PUTLAM)	(ob →)
34701	(25PUTLAM)	(ob →)
3470B	(26PUTLAM)	(ob →)
34715	(27PUTLAM)	(ob →)
34797	DUP4PUTLAM	(ob → ob) Does DUP then 4PUTLAM.
364FF	1GETABND	(→ 1lamob) Does 1GETLAM then ABND.
35DEE	1ABNDSWAP	(ob → 1lamob ob) Does 1GETABND then SWAP.
35F42	1GETSWAP	(ob → 1lamob ob) Does 1GETLAM then SWAP.
2F318	1GETLAMSWP1+	(# → 1lamob #+1) Does 1GETLAM then SWAP#1+.
3632E	2GETEVAL	(→ ?) Does 2GETLAM then EVAL.
3483E	GETLAMPAIR	(#n → #n ob lam F) (#n → #n T) Gets lam contents and name (10 = 1lam, 20 = 2lam, etc.)
347AB	DUPTEMPENV	(→) Duplicates topmost tempenv (clears protection word).
2B3A6	1NULLLAM{}	(→ {}) Puts a list with one NULLLAM in the stack.
271F4	(2NULLLAM{})	(→ {}) Puts a list with two times NULLLAM in the stack.
27208	(3NULLLAM{})	(→ {}) Puts a list with three times NULLLAM in the stack.
2B3B7	4NULLLAM{}	(→ {}) Puts a list with four times NULLLAM in the stack.

Chapter 19
Runstream Control

So far, you have only seen commands that do not affect the normal program flow. All the programs presented work sequentially, from the first command to the last, without any kind of change in this order. However, on all but the simplest programs, some kind of disruption in the default order is necessary. Sometimes, you need to have some part of the program repeated several times, or some actions must be executed only under certain conditions.

This chapter will describe some low-level entries that affect the normal execution order. The situations described above can be done with higer-level constructs such as loops (see Chapter 21) and conditionals (described in Chapter 20). And you will probaly use those constructs more often than most of the entries below. However, this chapter also describes some concepts that help understanding how a System RPL program, and how to change the normal program flow.

19.1 Some Concepts

As you know from the Introduction, a compiled System RPL program consists of a series of pointers to address in the memory. Basically, a program is executed by jumping to the pointed address, executing whatever is there, returning back to the program, jumping to the next address, and so on.

Actually, it is more complicated, because there are also objects such as real numbers, strings and even other programs (secondaries) embedded inside the programs. This requires some "magic" (actually, just carefully written code) to be properly handled, but it is outside the scope of this document to describe how this is dealt with. Just assume that when an object is found, it is "executed". For most objects (such as real numbers or strings), this means putting themselves in the stack, for secondaries this means executing their contents, and for others such as identifiers this means trying to recall the contents and executing them, or simply putting themselves in the stack.

Since the objects are executed in order, it becomes necessary to have some kind of variable that will always point to the next object to be executed. This is called the *interpreter pointer*, and is stored in a CPU register. After each object is executed, this pointer is advanced to point to the next object.

When a `DUP` is found in the program, what happens is as follows: actually, the only thing that is stored is the address `#03188h`. A jump is made to that address. In that address, there is some piece of machine-language code. This code is executed and in the end the interpreter pointer is advanced, and a jump is made to the next object, whatever it is.

Things get slightly more complicated when one wants to execute, for example, `INCOMPDROP`. At this command's address, there is a secondary object, whose contents happen to be `:: INNERCOMP DROP ;`. The problem is that it is necessary to switch to that (sub-)program, execute all its contents, and then return back to the program in which `INCOMPDROP` was called. Since it is perfectly possible for a sub-program to have even more sub-programs inside it, it turns out that some kind of stack is necessary. When a secondary (or any other composite) is executed, the address of the object after this composite in the calling program is pushed into this stack. The composite is then executed, by means of the interpreter pointer pointing to each of its objects. When it finishes, an address is popped from the return stack, and execution returns there. This was the address of the next object in the previous program, so execution resumes properly. This stack is called the *return stack*.

The description above is rather incomplete, but it should give you an idea of how things work. There are many details that would make a detailed explanation of System RPL programs too long and complicated, so this detailed explanation will not be given in this book.

Another important concept is that of the *runstream*. It is the sequence of objects that follow the object currently being executed. For example, during the execution of the `'` command in this program

```
:: ' DUP :: EVAL ; % 1. ;
```

the runstream contains three objects. The first is the command `DUP`. The second is the secondary that contains the `EVAL` command inside (but *not* the command `EVAL` or just the `::`), and the third is the real number one. Several words (including `'`, as you will see below), take their argument from the next object in the runstream, and not from the data stack, as most commands do. So, the "argument" to `'` is the command `DUP`.

You should now have understood why this chapter is called "Runstream Control": the commands here affect the runstream, that is, they affect the order in which the objects that form the program will be executed.

19.2 Runstream Commands

The commands described here are the basic actions available. In the reference section below you will find several commands that combine these commands with others.

Command	Stack and Description

`'`	($\rightarrow$ `ob`)
	This is very easy to understand: it pushes the object after it (that is, the first object in the runstream) in the stack. This pushed object will not be executed; execution resumes in the object after it. As an example,
	`:: %1 %2 ' %+ EVAL ;`
	is equivalent to
	`:: %1 %2 %+ ;`
	This action of pushing the next object in the stack instead of evaluating it is called *quoting* the next object.

`'R`	($\rightarrow$ `ob`) This pushes into the data stack the object that is pointed to by the topmost pointer in the return stack, and skips this pushed object. In other words, the first object in the composite that contains the composite currently being executed is pushed in the data stack, and skipped. If, however, the object that would be pushed is `SEMI`, then a null composite is pushed instead. As an example, the `RESOROMP` command is just like `ROMPTR@`, but its argument comes after it in the runstream (see Chapter 16). Here is how `RESOROMP` is defined: `:: 'R ROMPTR@ DROP ;` It just pushes the object after `RESOROMP` in the stack and calls `ROMPTR@`.
`ticR`	($\rightarrow$ `ob TRUE`) ($\rightarrow$ `FALSE`) This is similar to `'R`, but it will not push a null composite if there was no object to be pushed; instead it returns `FALSE`. If an object could be pushed, it is pushed along with `TRUE`.

Command	Stack and Description
`>R`	(`comp` →) This pushes a pointer to the body of the composite given as argument in the return stack. That means that when the current secondary ends, execution will not go back to the one that called the current composite. Before that, the composite given as argument will be executed, and only after it finishes will the execution resume at the secondary that called the current one. As an example, the code below returns in the stack the reals 3, 2 and 1, in this order: `:: ' :: % 1 ; >R % 3 % 2 ;`
`R>`	(→ `::`) Pushes in the data stack a secondary whose contents are what is pointed to by the first pointer in the return stack, which is popped. In other words, it pushes as a secondary the rest of the commands in the secondary that called the current one. This commands will then not be executed after the current secondary finishes. As an example, the code below pushes the reals 3, 2 and 1 in the stack, in this order: `:: :: R> EVAL % 1 ; % 3 % 2 ;`
`R@`	(→ `::`) This is the same as `R>`, but it does not pop the return stack. The same example of the above command, with `R>` changed into `R@` would return 3, 2, 1, 3 and 2.
`IDUP`	(→) Pushes the interpreter pointer into the return stack. This means that after the current secondary finishes, a jump will be made to the object just after the `IDUP`, thereby executing the rest of the current secondary once more.
`RDROP`	(→) Pops the return stack. That is, the remaining objects in the secondary that called the current one will not be executed.
`RDUP`	(→) Duplicates the top address in the return stack.
`RSWAP`	(→) Swaps the top two addresses in the return stack.
`?SEMI`	(`flag` →) If the flag is `TRUE`, skips the rest of the current secondary.

Command	Stack and Description
COLA	(→) This executes only the next object in the runstream, skipping the rest of the current secondary. The program below pushes only 1 in the stack: `:: COLA % 1 % 2 % 3 ;` See below for some good uses for COLA.
SKIP	(→) Skips the next object in the runstream. The program above, with COLA replaced by SKIP would push 2 and 3 in the stack.
?SKIP	(flag →) Does SKIP if the flag is true.

19.3 Some Examples

Our first example will show a useful use of COLA: when recursion is used. Suppose we have the two programs below for calculating the factorial of a number:

fact:

```
::
  CKREAL
  { LAM x } BIND
  %1              (First value for factorial)
  factiter
  ABND
;
```

factiter:

```
::
  LAM x %0= ?SEMI  (Exits if x=0)
  LAM x %*         (Multiplies by current value)
  LAM x %1- ’ LAM x STO
  COLA factiter
;
```

Note the word COLA before the recursive invocation of factiter. Without it, the program would require many return stack levels, all of which would point to SEMI. With COLA, nothing is pushed in the return stack. factiter is simply called, without storing the address of where the interpreter should jump back to. This makes the program always use a fixed number of return stack levels.

However, `COLA` is not used only in this case. It is a very useful command in other situations. Let us say that in your project you will frequently need to perform a `case` (see section 20.3) comparing if a real number is equal to 3. It is convenient to write a program to do this (like the built-in word `%1=case`) instead of repeating "`%3 %= case`" all the time.

A first attempt would be this program:

```
:: %3 %= case ;
```

However, this would not work. This is because `case` takes its arguments from the runstream, that is, the currently executed program, and not from the calling composite. This means the argument for `case` is `;`, which is not what is desired. But there is a solution: use `COLA` before the `case`. This will drop the rest of the runstream after the command after it, in a way merging the current command with the composite that called it. So, if we add `COLA` before `case`, and embed this new sub-program in another, like this:

```
:: ... :: %3 %= COLA case ; <act_T> <act_F> ...
```

it is as if the code were like this:

```
:: ... %3 %= case <act_T> <act_F> ...
```

which is what we want. Therefore, the correct way to define our sub-program is with `COLA` before `case`. This is a frequent combination, so there is a shortcut command, `COLAcase`, that is equivalent to `COLA` followed by `case`. There are other words like this, see the reference below.

The next example (which uses an error-trapping structure that will be described in Chapter 22) is the command `Đ` from the OT49 library (see section A.3.1), written by Wolfgang Rautenberg. This command is used like this:

```
« Đ ... »
```

That is, generally as the first command in a program (which, naturally, can be a System RPL program, not only a User one). It causes the program to be executed with the display off (which makes it slightly faster). All the folows the `Đ` until the end of the secondary is executed "blindly". When `Đ` is run, it turns off the display, and when the secondary finishes executing, the display is turned back on. But how can this be done, if nothing special needs to be called after the program finishes? The answer is simple: by manipulating the return stack. Here is the disassembly of that command:

```
::
  Code
  R>
```

```
  ERRSET
    COMPEVAL
  ERRTRAP
    ::
      'REVAL
      ERRJMP
    ;
  Code
;
```

The first code object turns off the display. It is a short and simple piece of machine language, but it is outside the scope of this book to describe it. Then, `R>` brings the rest of the composite that called `Ð` into the data stack. It is evaluated by `COMPEVAL`. The only difficulty in the program is that we must turn the display back on even if there was an error in the program. If there was en error, then the object after `ERRTRAP` is executed. First, `'REVAL` brings the first object after the current composite (this object happens to be the second code object, that turns on the display) into the data stack and executes it. Then, the error is triggered again with `ERRJMP`. If there was no error, the execution goes directly to the second code object, finishing the program.

As an example another way to deal with the return stack, we will study the list processor `DoL`, also in library OT49. This command expects a list in level two and any object (generally a command or a function) in level one. This object is evaluated for each of the list elements in order, and the results are collected in another list, which is then returned. This program uses some things which we have not studied yet, such as loops and the Virtual Stack. You might want to skip this example now and return to it later. Here is the code, without the argument checking part:

```
::
  OVER
  >R               (Push list elements in return stack)
  ticR             (Try to get first element)
  NOTcaseDROP      (If list is empty, drop the object)
  PushVStack&Clear (Save current stack)
  BINT0
  GetElemBotVStack (Get first list elemement)
  BEGIN
    BINT1
    GetElemBotVStack
    xEVAL          (Get object and evaluate)
    RSWAP
```

```
      ticR          (Get next element from list)
    WHILE           (Repeat while there are elements)
      RSWAP
    REPEAT
    DEPTH
    {}N             (Collect results)
    PopVStackAbove  (Get saved stack)
    4UNROLL3DROP    (Drop arguments & first object)
  ;
```

This program may be somewhat difficult to understand at first, but it manipulates very cleverly the return stack.

It starts by using >R to insert a pointer to the list contents in the return stack. If they were not removed later, then after this program finished, each of the objects in the list would be evaluated.

Then, the first object from the list is retrieved, with ticR. This also advances the pointer in the return stack to point to the second element. If the list was empty, then ticR returns FALSE. In this case, the object to be evaluated is dropped, and the empty list remains as the result of the program.

The real fun starts when there is at least one element. The whole stack is saved as a virtual stack level, but the first element of the list (retrieved with ticR previously) is retrived into the "new" stack.

Then, a loop is started. The loop used is very similar to a User RPL WHILE...REPEAT...END loop. For more details, see Chapter 21. The object is retrieved and evaluated, and then the next element from the list is retrieved with ticR. However, since the word BEGIN pushed something in the return stack (for an explanation, see section 21.1.1), it is necessary to use RSWAP to bring the pointer to the list elements back in the first return stack level, thus allowing ticR to get one of the elements. If there was an element, RSWAP is executed again to put the return stack back into its original stack, and the loop begins again, executing the object, and so on. When there are no more elements, control goes to after the REPEAT word. All results are collected in a list, and we retrieve the saved stack above the list with the results. Then the program simply drops the original list, the object to be evaluated and the first object of the list, which were in the stack when it was pushed into the Virtual Stack.

As you have seen, this program used the return stack as a storage place; the composite that was pushed there was never executed, because each of its elements were removed until there was nothing more to execute.

19.4 Reference

Addr.	Name	Description
06E8E	NOP	($\rightarrow$) Does nothing.
06EEB	'R	($\rightarrow$ ob) Pushes next object in return stack (i.e., the first object in the composite above this one) to the stack (skipping it). If top return stack is empty (contains SEMI), a null secondary is pushed and the pointer is not advanced.
06F66	'REVAL	($\rightarrow$?) Does 'R then EVAL.
36A27	'R'R	($\rightarrow$ ob1 ob2) Does 'R twice.
34BEF	ticR	($\rightarrow$ ob T) ($\rightarrow$ F) Pushes next object in return stack to stack and TRUE, of just FALSE if the top return stack body is empty. In this case, it is dropped.
36A4A	'RRDROP	($\rightarrow$ ob) Does 'R, then RDROP.
06F9F	>R	(:: $\rightarrow$) Pushes :: to top of return stack (skips prolog, i.e., the composite will be executed automatically).
0701F	R>	($\rightarrow$ ::) Creates and pops a secondary from top return stack body to stack.
07012	R@	($\rightarrow$ ::) Like R>, but the return stack is not popped.
0716B	IDUP	($\rightarrow$) Pushes top body into return stack.
06F8E	EVAL	(ob $\rightarrow$?) Evaluates object.
262FB	COMPEVAL	(comp $\rightarrow$?) EVAL just pushes a list back, this one executes it.
34BAB	2@REVAL	($\rightarrow$?) EVAL first object in the stream above the previous one.

Addr.	Name	Description
34BBB	3@REVAL	(→ ?) EVAL first object in the stream above the stream above the previous one.
34A31	GOTO	(→) Jumps to next address in stream. Address is a five-nibble address, not a system binary. Can only be used to jump to the middle of programs, cannot jump to a program prolog.
34A46	?GOTO	(flag →) If TRUE, jumps, else skips five nibbles.
34A59	NOT?GOTO	(flag →) If FALSE jumps, else skips five nibbles.
26111	RDUP	(→) Duplicates top return stack level.
06FB7	RDROP	(→) Pops the return stack.
343E1	2RDROP	(→) Pops two return stack levels.
343F3	3RDROP	(→) Pops three return stack levels.
36342	DROPRDROP	(ob →) Does DROP then RDROP.
3597F	RDROPCOLA	(→) Does RDROP then COLA.
34144	RSWAP	(→) Swap in the return stack.
368C9	RSKIP	(→) Skips first object in the return stack (i.e., the first object in the composite above this one).
2B8BE	(OBJ>R)	(ob →) Pushes an object into the return stack, for example for temporary storage. If ob is a list, the list is put as a whole onto the stream, not the individual elements.
2B8E6	(R>OBJ)	(→ ob) Gets an object from the return stack.
0312B	SEMI	(→) DROP the rest of the current stream.

19.4.1 Quoting Objects

Addr.	Name	Description
06E97	'	(→ nob (nextob)) Pushes next object in the stream to the stack (skipping it).
3696E	DUP'	(ob → ob nob) Does DUP then '.
36996	DROP'	(ob → nob) Does DROP then '.
36982	SWAP'	(ob1 ob2 → ob2 ob1 nob) Does SWAP then '.
369AA	OVER'	(ob1 ob2 → ob1 ob2 ob1 nob) Does OVER then '.
369BE	STO'	(ob id/lam → nob) Does STO then '.
369D2	TRUE'	(→ T nob) Pushes TRUE and the next object to the stack.
369FF	FALSE'	(→ F nob) Pushes FALSE and the next object to the stack.
369E6	ONEFALSE'	(→ #1 F nob) Pushes ONE, FALSE and the next object to the stack.
36A13	#1+'	(# → #+1 nob) Does #1+ then '.
36306	'NOP	(→ NOP) Pushes NOP to the stack.
3619E	'ERRJMP	(→ ERRJMP) Pushes ERRJMP to the stack.
2B90B	'DROPFALSE	(→ DROPFALSE) Pushes DROPFALSE to the stack.
25E6A	'DoBadKey	(→ DoBadKey) Pushes DoBadKey to the stack.
25E6B	'DoBadKeyT	(→ DoBadKey T) Pushes DoBadKey and TRUE to the stack.
2F32E	DROPDEADTRUE	(ob → DoBadKey T) Makes the user drop dead, then pushes TRUE.
36BBE	('x*)	(→ x*) Pushes x* (User word *) to the stack.

Addr.	Name	Description
36BD2	'xDER	(→ xDER) Pushes xDER (User word ∂) to the stack.
27B43	'IDFUNCTION	(→ xFUNCTION) Pushes xFUNCTION (User word FUNCTION) to the stack.
27B6B	'IDPOLAR	(→ xPOLAR) Pushes xPOLAR (User word POLAR) to the stack.
27B7F	'IDPARAMETER	(→ xPARAMETRIC) Pushes xPARAMETRIC (user word PARAMETRIC) to the stack.
29ED0	'Rapndit	(meta ob1...ob4 → meta&ob ob1...ob4) Takes ob from runstream and appends it to the meta starting in level 5.
36AA4	'xDEREQ	(ob → flag) Is ob eq to user command xDER?

19.4.2 Skipping Objects

Addr.	Name	Description
06FD1	COLA	Evals next obj and drops rest of this stream.
36A63	ONECOLA	Does ONE, then COLA.
3635B	SWAPCOLA	Does SWAP, then COLA.
3636F	XYZ>ZCOLA	Does UNROT2DROP, then COLA.
34AD3	COLA_EVAL	Returns and evals first obj in previous stream.
35994	COLACOLA	Drops rest of current stream does COLA in the above one.
0714D	SKIP	Skips 1 obj in the runstream.
35715	skipcola	Does SKIP, then COLA.
3570C	2skipcola	Does 2SKIP, then COLA.
35703	3skipcola	Does 3SKIP, then COLA.
356D5	5skipcola	Skips 5 objects, then does COLA.
363FB	COLASKIP	Drops rest of current stream and skips one obj in above stream.

Chapter 20
Conditionals

In System RPL, conditionals are a bit different from User RPL. The first difference is that in User RPL, a "false" is represented by the real number zero; any other value represents a "true". In System RPL, a "false" is represented by the word `FALSE`, and a "true" is represented by the word `TRUE` (amazing!). These words just put themselves in the stack when run. All commands that do a test return one of them. Words like `IT` or `case` take one of them as argument.

Should you need, you can convert a `TRUE` or `FALSE` to a (real) 0 or 1 with `COERCEFLAG`. There is not a dedicated function to do the opposite transformation, like `UNCOERCEFLAG`, but `%0<>` does the job perfectly.

There are many commands that put `TRUE`, `FALSE`, or some combination of them in the stack. See the list below.

The Boolean operators are present, too: `NOT`, `AND`, `OR` and `XOR`. There are some combinations, see below for a list.

20.1 Tests

The test words are commands which take one or more arguments and return either `TRUE` or `FALSE`, after doing some kind of comparison between the arguments. The tests for each kind of object type are listed in the reference section of the chapter of each object type. Tests for object type can be found on Chapter 29. Other kinds of tests are listed in the reference section below.

The most important of these tests are `EQ` and `EQUAL`. Both take two objects and return a flag. The first checks if the objects are the same, i.e., occupy the same address in memory. The second checks if the objects are equal in terms of contents. The difference is that `:: BINT2 # 2 EQUAL ;` returns `TRUE`, but if `EQUAL` is replace by `EQ`, then the program returns `FALSE`, because one object is the built-in bint 2, found at address #3311B, and the other is a bint whose address cannot be predicted, but certainly is not in the ROM.

Another example: if you put a string in level one, and press `ENTER`, `EQ` and EQUAL will return TRUE. However, if you enter a string, and then enter again the exact same string, only EQUAL will return TRUE. This happens because the contents of the strings are the same, but they are different objects in memory, occupying each a different address in memory. They just happen to have the same contents. When possible, you should use EQ in your programs since it is faster than EQUAL.

20.2 If. . . Then. . . Else

Most of the time, you will create this kind of conditionals with the IT and ITE commands:

Word	Stack and Action
IT	(flag →) If the flag is TRUE, the next object is executed, otherwise it is skipped.
ITE	(flag →) If the flag is TRUE, the next object is executed, and the second is skipped. If it is FALSE, the next object is skipped and the second is executed.

The following snippet changes a zero into a one, but does nothing to other numbers:

```
... DUP %0= IT %1+ ...
```

The code below will output “Equal” if two objects are equal, and “Not equal” if not:

```
... EQUAL $ "Equal" $ "Not equal" ...
```

Naturally, when you need to execute several commands, you will need to include them in a secondary.

20.3 Case

The CASE words are a combination of IT, SKIP and COLA (see Chapter 19). The basic word is case, but there are combinations of it with tests and other commands.

case takes a flag in level one. If the flag is TRUE, the next object is executed, but only it: the rest of the stream is dropped. So, `TRUE case` is equivalent to `COLA`. If the flag is `FALSE`, the next object is skipped and execution continues after it. So, `FALSE case` is the same as `SKIP`.

The example below shows how to build a familar case structure similar to that found in other languages (even User RPL!). It outputs a string representing the bint in level one.

```
::
  DUP #0=      case $ "Zero"
  DUP BINT1 #= case $ "One"
  DUP BINT2 #= case $ "Two"
  ...
;
```

There are many words that combine `case` with other words. One of them is `OVER#=case`. It is not difficult to figure out what it does: first, `OVER`. Then, #= compares two bints. Finally, the `case` works as before. Using this word, the code above could be rewritten as:

```
::
  BINT0 OVER#=case $ "Zero"
  BINT1 OVER#=case $ "One"
  BINT2 OVER#=case $ "Two"
  ...
;
```

In the reference section below, you will find a list of the words that execute a `case` besides some other action. These words are composed of an initial part, the `case` itself and a final part. Some have only the initial or final part besides the `case`, some have both. The initial part represents the commands that are executed before the `case`, and it should be pretty straightforward to understand their action, as an example `NOTcase` is equivalent to `NOT` followed by `case`. For the final part, things become more complicated, because there are two kinds of final part. The first kind has the final part written in UPPERCASE letters. The commands in the final kind are executed if the test is true. You only need to provide the action for the `FALSE` situation. For example, this snippet

```
... caseDROP <FalseAction> ...
```

is equivalent to

```
... case DROP <FalseAction> ...
```

The second type comprises the words that have the final part in lowercase letters. In this case, the commands in the final part are executed *along with the object that follows* `case` when the test is true. In other words, this snippet

```
... casedrop <TrueAction> <FalseAction> ...
```

is equivalent to

```
... case :: DROP <TrueAction> ; <FalseAction> ...
```

Unfortunately, some entries have been misnamed, and this convention was not adhered. These entries are marked clearly in the descriptions below.

Also, the "stack diagrams" of most of the words below are not true stack diagrams. What is on the left side of the arrow is the contents of the stack before calling the entry, as usual. `ob1` and `ob2` are different objects. `f1` and `f2` are different flags; `T` represents `TRUE` and `F`, `FALSE`. `#m` and `#n` represent two binary integers, `#m` being smaller than `#n`. `#set` is the number of a flag, and this flag is set, `#clr` is the number of a flag, this flag being clear. On the right of the arrow, the objects which will be executed when the stack matches the left side of the arrow are represented. The initial stream has the form: `:: <test_word> <ob1> ... <obn> ;` In the diagrams, `<rest>` represents all the objects after the object that appers before `<rest>`. In this right side of the arrow there are also objects appearing without the angle brackets already. These are objects in the data stack that result after the word is run, and not objects in the runstream.

20.4 Reference

20.4.1 Boolean Flags

Addr.	Name	Description
2602B	COERCEFLAG	(T → %1) (F → %0) Converts system flag to user flag, drops current stream.
301BA	%0<>	(% → flag) Can be used to change a user flag into a system flag.
03A81	TRUE	(→ T)
27E87	TrueTrue	(→ T T)

Addr.	Name	Description
36540	TrueFalse	(→ T F) aka: TRUEFALSE
03AC0	FALSE	(→ F)
36554	FalseTrue	(→ F T) aka: FALSETRUE
283E8	FalseFalse	(→ F F)
27E9B	failed	(→ F T)
35280	DROPTRUE	(ob → T)
2D7006	^2DROPTRUE	(ob ob' → T)
35289	DROPFALSE	(ob → F)
35B32	2DROPFALSE	(ob1 ob2 → F)
28211	NDROPFALSE	(ob1..obn #n → F)
2812F	SWAPTRUE	(ob1 ob2 → ob2 ob1 T)
374BE	SWAPDROPTRUE	(ob1 ob2 → ob2 T)
35EF2	XYZ>ZTRUE	(ob1 ob2 ob3 → ob3 T)
2962A	RDROPFALSE	(→ F) Puts FALSE in the stack and drops rest of current stream.
03AF2	NOT	(flag → flag') Returns FALSE if the input is TRUE, and vice-versa.
03B46	AND	(flag1 flag2 → flag) Returns TRUE if both flags are TRUE.
03B75	OR	(flag1 flag2 → flag) Returns TRUE if either flag is TRUE.
03ADA	XOR	(flag1 flag2 → flag) Returns TRUE if flags are different.
365F9	ORNOT	(flag1 flag2 → flag) Returns FALSE if either flag is TRUE.
35C7C	NOTAND	(flag1 flag2 → flag) Returns TRUE if flag1 is TRUE and flag2 is FALSE.
35CB8	ROTAND	(flag1 ob flag2 → ob flag) Returns TRUE if either flag is TRUE.

20.4.2 General Tests

Addr.	Name	Description
03B2E	EQ	(ob1 ob2 → flag) Returns TRUE if both objects are the same, i.e., they occupy the same physical space in memory. Only the addresses of the objects are tested.
36621	2DUPEQ	(ob1 ob2 → ob1 ob2 flag) Does 2DUP then EQ.
3664E	EQOR	(flag ob1 ob2 → flag') Does EQ then OR.
3607F	EQOVER	(ob3 ob1 ob2 → ob3 flag ob3) Does EQ then OVER.
3663A	EQ:	(ob → flag) EQ with the next object in the current stream.
36635	DUPEQ:	(ob → ob flag) Does DUP then EQ:.
03B97	EQUAL	(ob1 ob2 → flag) Returns TRUE if the objects are equal (but not necessarily the same), i.e., their prologs and contents are the same.
3660D	EQUALNOT	(ob1 ob2 → flag) Returns TRUE if the objects are different.
36662	EQUALOR	(flag ob1 ob2 → flag') Does EQUAL then OR.
0FF006	ˆContains?	(ob1 ob2 → ob1 ob2 flag) Tests if ob1 contains ob2. If ob1 is a symbolic then ob1 is searched for embedded ob2. If ob1 is a list then ob1 is traversed for a direct match. Otherwise, tests if ob1 and ob2 are equal.

20.4.3 True/False Tests

Addr.	Name	Description
34AA1	?SEMI	(T → :: ;) (F → :: <ob1> <rest> ;)
34A92	NOT?SEMI	(T → :: <ob1> <rest> ;) (F → :: ;)

Addr.	Name	Description
3692D	?SEMIDROP	(ob T → :: ob ;) (ob F → :: <ob1> <rest> ;)
34BD8	NOT?DROP	(ob T → :: ob <ob1> <rest> ;) (ob F → :: <ob1> <rest> ;)
35F56	?SWAP	(ob1 ob2 T → :: ob2 ob1 <ob1> <rest> ;) (ob1 ob2 F → :: ob1 ob2 <ob1> <rest> ;)
35DDA	?SKIPSWAP	(ob1 ob2 T → :: ob1 ob2 <ob1> <rest> ;) (ob1 ob2 F → :: ob2 ob1 <ob1> <rest> ;)
35F97	?SWAPDROP	(ob1 ob2 T → :: ob1 <ob1> <rest> ;) (ob1 ob2 F → :: ob2 <ob1> <rest> ;)
35F7E	NOT?SWAPDROP	(ob1 ob2 T → :: ob2 <ob1> <rest> ;) (ob1 ob2 F → :: ob1 <ob1> <rest> ;)
070FD	RPIT	(T ob → :: ob <ob1> <rest> ;) (F ob → :: <ob1> <rest> ;) ob is actually executed, and not pushed in the stack.
070C3	RPITE	(T ob1 ob2 → :: ob1 <ob1> <rest> ;) (F ob1 ob2 → ob2 <ob1> <rest> ;) ob1 or ob2 is actually executed, and not pushed in the stack.
34AF4	COLARPITE	(T ob1 ob2 → :: ob1 ;) (F ob1 ob2 → :: ob2 ;) ob1 or ob2 is actually executed, and not pushed in the stack.
34B4F	2'RCOLARPITE	Return to composite and ITE there.
34A22	IT	(T → :: <ob1> <rest> ;) (F → :: <ob2> <rest> ;)
0712A	?SKIP	(T → :: <ob2> <rest> ;) (F → :: <ob1> <rest> ;) aka: NOT_IT

Addr.	Name	Description
34B3E	ITE	(T → :: <ob1> <ob3> <rest> ;)
		(F → :: <ob2> <rest> ;)
36865	COLAITE	(T → :: <ob1> ;)
		(F → :: <ob2> ;)
34ABE	ITE_DROP	(ob T → :: <ob2> <rest> ;)
		(ob F → :: ob <ob1> <rest> ;)
36EED	ANDITE	(f1 f2 → :: <ob1> <ob3> <rest> ;)
		(f1 f2 → :: <ob2> <rest> ;)
349F9	case	(T → :: <ob1> ;)
		(F → :: <ob2> <rest> ;)
34A13	NOTcase	(T → :: <ob2> <rest> ;)
		(F → :: <ob1> ;)
36D4E	ANDcase	(f1 f2 → :: <ob1> ;)
		(f1 f2 → :: <ob2> <rest> ;)
36E6B	ANDNOTcase	(f1 f2 → :: <ob1> ;)
		(f1 f2 → :: <ob2> <rest> ;)
359E3	ORcase	(f1 f2 → :: <ob1> ;)
		(f1 f2 → :: <ob2> <rest> ;)
3495D	casedrop	(ob T → :: <ob1> ;)
		(ob F → :: ob <ob2> <rest> ;)
3494E	NOTcasedrop	(ob T → :: ob <ob2> <rest> ;)
		(ob F → :: <ob1> ;)
34985	case2drop	(ob1 ob2 T → :: <ob1> ;)
		(ob1 ob2 F → :: ob1 ob2 <ob2> <rest> ;)
34976	NOTcase2drop	(ob1 ob2 T → :: ob1 ob2 <ob2> <rest> ;)
		(ob1 ob2 F → :: <ob1> ;)
349B1	caseDROP	(ob T → :: ;)
		(ob F → :: ob <ob1> <rest> ;)
349C6	NOTcaseDROP	(ob T → :: ob <ob1> <rest> ;)
		(ob F → :: ;)
368FB	casedrptru	(ob T → T)
		(ob F → :: ob <ob1> <rest> ;)
		Note: should be caseDRPTRU.
365B3	casedrpfls	(ob T → F)
		(ob F → :: ob <ob1> <rest> ;)
		Note: should be caseDRPFLS.

Addr.	Name	Description
36B3A	NOTcsdrpfls	(ob T → :: ob <ob1> <rest> ;) (ob F → F) Note: should be NOTcaseDRPFLS.
349D6	case2DROP	(ob1 ob2 T → :: ;) (ob1 ob2 F → :: ob1 ob2 <ob1> <rest> ;)
349EA	NOTcase2DROP	(ob1 ob2 T → :: ob1 ob2 <ob1> <rest> ;) (ob1 ob2 F → :: ;)
365CC	case2drpfls	(ob1 ob2 T → F) (ob1 ob2 F → :: ob1 ob2 <ob1> <rest> ;) Note: should be case2DRPFLS.
3652C	caseTRUE	(T → T) (F → :: <ob1> <rest> ;)
36914	NOTcaseTRUE	(T → :: <ob1> <rest> ;) (F → T)
365E5	caseFALSE	(T → F) (F → :: <ob1> <rest> ;)
2B2C5	NOTcaseFALSE	(T → :: <ob1> <rest> ;) (F → F)
359AD	COLAcase	(T → :: <ob1> ;) (F → :: <ob2> <rest> ;) Drops the rest of current stream and executes case in the stream above.
359C8	COLANOTcase	(T → :: <ob2> <rest> ;) (F → :: <ob1> ;) Drops the rest of current stream and executes NOTcase in the stream above.

20.4.4 Binary Integer Tests

Addr.	Name	Description
363B5	#=?SKIP	(#m #n → :: <ob2> <rest> ;) (#m #n → :: <ob1> <rest> ;)
363E2	#>?SKIP	(#m #n → :: <ob1> <rest> ;) (#m #n → :: <ob2> <rest> ;)

Addr.	Name	Description
35C54	#=ITE	(#m #n → :: <ob1> <ob3> <rest> ;)
		(#m #n → :: <ob2> <rest> ;)
36F29	#<ITE	(#m #n → :: <ob1> <ob3> <rest> ;)
		(#m #n → :: <ob2> <rest> ;)
36F3D	#>ITE	(#m #n → :: <ob2> <rest> ;)
		(#m #n → :: <ob1> <ob3> <rest> ;)
348D2	#=case	(#m #n → :: <ob1> ;)
		(#m #n → :: <ob2> <rest> ;)
348E2	OVER#=case	(#m #n → :: #m <ob1> ;)
		(#m #n → :: #m <ob2> <rest> ;)
34939	#=casedrop	(#m #n → :: <ob1> ;)
		(#m #n → :: #m <ob2> <rest> ;)
		Note: should be OVER#=casedrop.
36590	#=casedrpfls	(#m #n → F)
		(#m #n → :: #m <ob1> <rest> ;)
		Note: should be OVER#=caseDRPFLS.
36D9E	#<>case	(#m #n → :: <ob2> <rest> ;)
		(#m #n → :: <ob1> ;)
36D76	#<case	(#m #n → :: <ob1> ;)
		(#m #n → :: <ob2> <rest> ;)
36DCB	#>case	(#m #n → :: <ob2> <rest> ;)
		(#m #n → :: <ob1> ;)
34A7E	#0=?SEMI	(#0 → :: ;)
		(# → :: <ob1> <rest> ;)
36383	#0=?SKIP	(#0 → :: <ob2> <rest> ;)
		(# → :: <ob1> <rest> ;)
36F15	#0=ITE	(#0 → :: <ob1> <ob3> <rest> ;)
		(# → :: <ob2> <rest>)
36ED4	DUP#0=IT	(#0 → :: #0 <ob1> <rest> ;)
		(# → :: # <ob2> <rest> ;)
36F51	DUP#0=ITE	(#0 → :: #0 <ob1> <ob3> <rest> ;)
		(# → :: # <ob2> <rest> ;)
348FC	#0=case	(#0 → :: <ob1> ;)
		(# → :: <ob2> <rest> ;)
348F7	DUP#0=case	(#0 → :: #0 <ob1> ;)
		(# → :: # <ob2> <rest> ;)
3490E	DUP#0=csedrp	(#0 → :: <ob1> ;)
		(# → :: # <ob2> <rest> ;)

Addr.	Name	Description
36D21	DUP#0=csDROP	(#0 → :: ;)
		(# → :: # <ob1> <rest> ;)
36D8A	#1=case	(#1 → :: <ob1> ;)
		(# → :: <ob2> <rest> ;)
3639C	#1=?SKIP	(#1 → :: <ob2> <rest> ;)
		(# → :: <ob1> <rest> ;)
36DB2	#>2case	(#0/#1/#2 → :: <ob2> <rest> ;)
		(# → :: <ob1> ;)
25E72	?CaseKeyDef	(# #' → :: ' ob1 T ;)
		(# #' → :: <ob2> <rest> ;)
		Compares two bints. If equal, quotes the next object from the runsream and returns it along with TRUE.
25E73	?CaseRomptr@	(# #' → ob T)
		(# #' → F)
		(# #' → :: <ob2> <rest> ;)
		Compares two bints. If equal, tries to resolve the rompointer which must be the next object in the runstream. The ROMPTR@ pushes TRUE when successful, so this entry can be used directly for key handlers.

20.4.5 Real and Complex Number Tests

Addr.	Name	Description
2B149	%0=case	(%0 → :: %0 <ob1> ;)
		(ob → :: ob <ob2> <rest> ;)
36DDF	j%0=case	(%0 → :: <ob1> ;)
		(ob → :: <ob2> <rest> ;)
2B15D	C%0=case	(C%0 → :: C%0 <ob1> ;)
		(ob → :: ob <ob2> <rest> ;)
2B11C	num0=case	(0 → :: 0 <ob1> ;)
		(ob → :: ob <ob2> <rest> ;)
		Both a real and a complex zero are TRUE conditions for this test.
2B1A3	%1=case	(%1 → :: %1 <ob1> ;)
		(ob → :: ob <ob2> <rest> ;)

Addr.	Name	Description
2B1C1	C%1=case	(C%1 → :: C%1 <ob1> ;) (ob → :: ob <ob2> <rest> ;)
2B176	num1=case	(1 → :: 1 <ob1> ;) (ob → :: ob <ob2> <rest> ;) Both a real and a complex one are TRUE conditions for this test.
2B20C	%2=case	(%2 → :: %2 <ob1> ;) (ob → :: ob <ob2> <rest> ;)
2B22A	C%2=case	(C%2 → :: C%2 <ob1> ;) (ob → :: ob <ob2> <rest> ;)
2B1DF	num2=case	(2 → :: 2 <ob1> ;) (ob → :: ob <ob2> <rest> ;) Both a real and a complex two are TRUE conditions for this test.
2B289	%-1=case	(%-1 → :: %-1 <ob1> ;) (ob → :: ob <ob2> <rest> ;)
2B2A7	C%-1=case	(C%-1 → :: C%-1 <ob1> ;) (ob → ob <ob2> <rest> ;)
2B25C	num-1=case	(-1 → :: -1 <ob1> ;) (ob → :: ob <ob2> <rest> ;) Both a real and a complex -1 are TRUE conditions for this test.

20.4.6 Meta Object Tests

Addr.	Name	Description
2AFFB	MEQ1stcase	(meta&ob1 ob2 → ob1=ob2 ? case) Meta&ob1 ob2 ob1=ob2 ? case
2AF37	AEQ1stcase	(meta&ob → ob=nob ? case) Meta&ob ob=nob ? case
2B01B	MEQopscase	(meta1&ob1 meta2&ob2 ob3 →) Meta1&ob1 Meta2&ob2 ob3
2B06A	AEQopscase	meta1&ob1 meta2&ob2 Meta1&ob1 Meta2&ob2
2B083	Mid1stcase	(meta&ob → ob is id) lam ? case Meta&ob ob is id or lam ? case

Addr.	Name	Description
2AE32	M-1stcasechs	(Meta&NEG → Meta COLA) (Meta → Meta SKIP) (Meta&(%<0) → Meta&ABS(%) COLA) Meta&NEG Meta COLA ; Meta Meta SKIP Meta&(%<0) Meta&ABS(%) COLA

20.4.7 General Object Tests

Addr.	Name	Description
36EBB	EQIT	(ob1 ob1 → :: <ob1> <rest> ;) (ob1 ob2 → :: <ob2> <rest> ;)
36F01	EQITE	(ob1 ob1 → :: <ob1> <ob3> <rest> ;) (ob1 ob2 → :: <ob2> <rest> ;)
36D3A	jEQcase	(ob1 ob1 → :: <ob1> ;) (ob1 ob2 → :: <ob2> <rest> ;)
34999	EQcase	(ob1 ob1 → :: ob1 <ob1> ;) (ob1 ob2 → :: ob1 <ob2> <rest> ;) Note: Should be called OVEREQcase.
359F7	REQcase	(ob → :: ob <ob2> ;) (ob → :: ob <ob3> <rest> ;) EQcase with the next object in the runstream.
34920	EQcasedrop	(ob1 ob1 → :: <ob1> ;) (ob1 ob2 → :: ob1 <ob2> <rest> ;) Note: should be OVEREQcasedrop.
35A10	REQcasedrop	(ob → <ob2> ;) (ob → <ob3> <rest> ;) EQcasedrop with the next object in the runstream.
36D62	EQUALcase	(ob1 ob1 → :: <ob1> ;) (ob1 ob2 → :: <ob2> <rest> ;)
36E7F	EQUALNOTcase	(ob1 ob1 → :: <ob2> <rest> ;) (ob1 ob2 → :: <ob1> ;)
36D08	EQUALcasedrp	(ob ob1 ob2 → :: <ob1> ;) (ob ob1 ob2 → :: ob <ob2> <rest> ;)

Addr.	Name	Description
2AD81	EQUALcasedrop	(ob1 ob2 → :: <ob1> ;) (ob1 ob2 → :: ob1 <ob2> <rest> ;)
29E99	tok=casedrop	($ $' → :: <ob1> ;) ($ $' → :: $ <ob2> <rest> ;) Note: should be OVERtok=casedrop.
2ADBD	nonopcase	(seco → :: seco <ob2> <rest> ;) (ob → :: ob <ob1> ;)
2B0CC	idntcase	(id → :: id <ob1> ;) (ob → :: ob <ob2> <rest> ;)
36E93	dIDNTNcase	(id → :: id <ob2> <rest> ;) (ob → :: ob <ob1> ;)
2B0EF	idntlamcase	(id/lam → :: id <ob1> ;) (ob → :: ob <ob2> <rest> ;)
36DF3	REALcase	(% → :: <ob1> ;) (ob → :: <ob2> <rest> ;)
36EA7	dREALNcase	(% → :: % <ob2> <rest> ;) (ob → :: ob <ob1> ;)
36E07	dARRYcase	([] → :: [] <ob1> ;) (ob → :: ob <ob2> <rest> ;)
36E43	dLISTcase	({} → :: {} ob1 ;) (ob → :: ob <ob2> <rest> ;)
260C6	NOTLISTcase	({} → :: {} <ob2> <rest> ;) (ob → :: ob <ob1> ;)
260D0	NOTSECOcase	(seco → :: seco <ob2> <rest> ;) (ob → :: ob <ob1> ;)
260CB	NOTROMPcase	(romp → :: romp <ob2> <rest> ;) (ob → :: ob <ob1> ;)
2ADE0	numb1stcase	(%/C%/[]/[L] → :: <ob1> ;) (ob → :: ob2 <rest> ;) If %, C%, [ ] or [L] then COLA, else SKIP.

20.4.8 Miscellaneous

Addr.	Name	Description
36F65	UserITE	(#set → :: <ob1> <ob3> <rest> ;) (#clr → :: <ob2> <rest> ;)
36F79	SysITE	(#set → :: <ob1> <ob3> <rest> ;) (#clr → :: <ob2> <rest> ;)
36C4F	caseDoBadKey	(T → :: DoBadKey ;) (F → :: <ob1> <rest> ;) aka: caseDEADKEY
36C36	caseDrpBadKy	(ob T → :: DoBadKey ;) (ob F → :: ob <ob1> <rest> ;)
361B2	caseERRJMP	(T → :: ERRJMP ;) (F → :: <ob> <rest> ;)
36B53	caseSIZEERR	(T → :: SIZEERR ;) (F → :: <ob> <rest> ;)
36B67	NcaseSIZEERR	(T → :: <ob> <rest> ;) (F → :: SIZEERR ;)
36BAA	NcaseTYPEERR	(T → :: <ob1> <rest> ;) (F → :: TYPEERR ;)
25EEE	NoEdit?case	(→ :: <ob1> <rest> ;) (→ :: <rest> ;) Tests if there is no edit line active.
36E57	EditExstCase	(→ :: <ob1> <rest> ;) (→ :: <rest> ;) Tests if there is an edit line active.
2BE36	(ALGcase)	(→ :: <ob1> ;) (→ :: <ob2> <rest>) Tests for algebraic mode and does case.

Chapter 21
Loops

As in User RPL, there are two types of loops in System RPL: indefinite loops and definite loops. Indefinite loops are loops in which you do not know beforehand how many times it will be executed: it will repeat until a specific condition is met. They are created in a very similar manner to User RPL indefinite loops. Definite loops, on the other hand, are executed a number of times specified before its start. They not created exactly like in User RPL, but their use is simple and more powerful. For example, you can change the number of times to run the loop while running it.

In the descriptions below, the elements between < > can consist of several objects, unless otherwise noted.

21.1 Indefinite Loops

In System RPL, indefinite loops can be made in three ways. The first is the WHILE loop. It is created like this:

```
BEGIN
  <test clause>
WHILE
  <loop object>
REPEAT
```

This kind of loop executes `<test clause>`, and if the test is `TRUE`, `<loop object>` is executed, and the loop starts again. If the test returned `FALSE`, then execution resumes past `REPEAT`. If the first test returned `FALSE`, this loop would never be executed.

This loop requires `<loop object>` to be a single object. Most of the times, this will be a composite.

The second type of indefinite loop is the `UNTIL` loop. It is created like this:

```
BEGIN
  <loop clause>
UNTIL
```

This loop is always executed at least once. The word `UNTIL` expects a flag. If it is `FALSE`, the `<loop clause>` is executed again. If it is `TRUE`, execution continues past `UNTIL`.

There is also a third type of indefinite loop:

```
BEGIN
  <loop object>
AGAIN
```

This loop has no test. To exit it, an error condition must happen, or the return stack must be directly manipulated. This is useful if the loop code contains several different locations at which decisions about repeating or exiting the loop have to be made.

21.1.1 How Indefinite Loops Work

Indefinite loops are formed by combinations the words `BEGIN`, `WHILE`, `REPEAT`, `UNTIL` and `AGAIN`. These have nothing special, they are commands just like the others, that when combined allow loops to be made. They work by manipulating the runstream and the return stack, so be sure you understand this concepts (see section 19.1 if in doubt).

Word	Stack and action
`BEGIN`	(→) This copies the interpreter pointer into the return stack.
`UNTIL`	(`flag` →) If the flag is `TRUE`, pops the return stack, otherwise sets the interpreter pointer to the topmost address of the return stack, without popping it.
`WHILE`	(`flag` →) If the flag is `TRUE`, does nothing. Otherwise, pops the return stack and skips the next two objects in the runstream.
`REPEAT`	(→) Sets the interpreter pointer to the topmost pointer of the return stack, without popping it.

Word	Stack and action
AGAIN	(→) Sets the interpreter pointer to the topmost address of the return stack, without popping it.

From the descriptions above, you should have understood how the loops work, and also why the BEGIN...WHILE...REPEAT loops requires a single object between WHILE and REPEAT.

21.2 Definite Loops

Definite loops are created with DO and LOOP (or other equivalent words). DO takes two bints from the stack, representing the stop and start values. The start value is stored as the current index, which can be recalled with INDEX@. The stop value can be recalled with ISTOP@. You can store a new value to one of them with INDEXSTO and ISTOPSTO, respectively.

DO's counterparts are LOOP and +LOOP. The former increments the index value by one, and checks if the new value is greater than or equal to the stop value, exiting the loop if it is. If not, the loop is executed again. +LOOP works similarly, incrementing the index by the bint in level one.

The standard form of a DO loop is

```
stop start DO <loop clause> LOOP
```

which executes <loop clause> for each index value from start to stop-1. Note that the stop value is greater than what it would be in User RPL, so pay attention. Also, the "stop" value comes before the "start" value.

There are several words provided to be used with DO loops, like ONE_DO. They are listed below.

Here is an example of a simple loop which outputs the bints #1h, #2h, #3h and #4h to the stack:

```
::
  BINT5 BINT1
  DO
    INDEX@
  LOOP
;
```

It could be changed to:

```
::
  BINT5 ONE_DO
    INDEX@
  LOOP
;
```

21.2.1 How a DO Loop Works

If you have some familiarity with concepts such as the return stack and the runstream (described in section 19.1), this section will explain to you how a `DO` loop works.

When the word `DO` is executed, it pushes the interpreter pointer (which points to the first object after the `DO`) to the return stack. It also creates a `DoLoop` environment, storing the initial and stop values.

Execution continues normally, running all commands between `DO` and `LOOP`.

When `LOOP` is executed, it increments the current value in the most recent `DoLoop` environment. If it is greater than or equal to the stop value of that environment, the environment is destroyed, and one level is popped out of the return stack. This removes the pointer to the first object after `DO`, and execution continues normally after `LOOP`. If the value is smaller, then the interpreter pointer is set to the top value in the return stack, causing the execution to re-start at the first object after the `DO`.

21.3 Reference

21.3.1 Indefinite Loops

Addr.	Name	Description
0716B	IDUP	($\rightarrow$) Pushes interpreter pointer into the return stack.
071A2	BEGIN	($\rightarrow$) Pushes interpreter pointer into the return stack.

Addr.	Name	Description
071AB	AGAIN	(→) Sets the interpreter pointer to the topmost value in the return stack, without popping it.
071E5	REPEAT	(→) Sets the interpreter pointer to the topmost value in the return stack, without popping it.
071C8	UNTIL	(flag →) If FALSE then AGAIN, otherwise RDROP.
3640F	NOT_UNTIL	(flag →) NOT then UNTIL.
35B96	#0=UNTIL	(# → #) Actually, should be DUP#0=UNTIL.
071EE	WHILE	(flag →) If TRUE does nothing, otherwise RDROP then 2SKIP.
36428	NOT_WHILE	(flag →) NOT then WHILE.
36441	DUP#0<>WHILE	(# →) Try to guess what it does.

21.3.2 Definite Loops

Addr.	Name	Description
073F7	DO	(#stop #start →)
073C3	ZERO_DO	(#stop →)
364C8	DUP#0_DO	(#stop → #stop)
073CE	ONE_DO	(#stop →)
073DB	#1+_ONE_DO	(#stop →)
364E1	toLEN_DO	({} → {}) From ONE to #elements.
07334	LOOP	(→)
073A5	+LOOP	(# →) Increments index by specified number.
364AF	DROPLOOP	(ob →)
36496	SWAPLOOP	(ob1 ob2 → ob2 ob1)
34AAD	SEMILOOP	(→)
07221	INDEX@	(→ #) Recalls topmost loop counter value.

Addr.	Name	Description
3645A	DUPINDEX@	(ob → ob #)
3646E	SWAPINDEX@	(ob1 ob2 → ob2 ob1 #)
36482	OVERINDEX@	(ob1 ob2 → ob1 ob2 ob1 #)
367D9	INDEX@#-	(# → #')
07270	INDEXSTO	(# →) Stores new topmost loop counter value.
07249	ISTOP@	(→ #) Recalls topmost loop stop value.
07295	ISTOPSTO	(# →) Stores new topmost loop stop value.
283FC	ISTOP-INDEX	(→ #)
07258	JINDEX@	(→ #) Recalls second topmost loop counter value.
072AD	JINDEXSTO	(# →) Stores new second topmost loop counter value.
07264	JSTOP@	(→ #) Recalls second topmost loop stop value.
072C2	JSTOPSTO	(# →) Stores new second topmost loop stop value.
3709B	ExitAtLOOP	(→) Does not exit loop immediately. Just stores zero as the stop value, so all objects until the next LOOP will be evaluated. aka: ZEROISTOPSTO

Chapter 22
Error Handling

When an error occurs in a System RPL program, normally the program is aborted and a message box is popped with the error message. However, sometimes it is desired for the program to trap the error and if possible continue execution, or perhaps show that an error happened in a different way.

Other times, the programs need to *generate* an error. For example, if the user gave invalid input for the program, it should abort with a “Invalid Argument Type” error, instead of risking crashing the machine.

22.1 Trapping Errors

You can intercept the execution of the error handling subsystem, i.e., trap an error generated by your program, using the following structure:

```
::
  ...
  ERRSET
    :: <suspect objects> ;
  ERRTRAP
    :: <if-error objects> ;
  ...
;
```

If <suspect objects> and/or <if-error objects> are only a single object, it is not necessary to include them inside a secondary, naturally.

It works like this: if the <suspect objects> generates an error, the execution continues at <if-error objects>. Otherwise, it continues past it.

The action of <if-error objects> is completely flexible. Normally, it will handle the error and then continue or exit the program. The current error number can be recalled with ERROR@, and then your program can do different actions on different kinds of errors. The error messages and numbers can be found in Appendix E.

22.1.1 The Protection Word

Each temporary environment (see Chapter 18), `DO`/`LOOP` environment (see Chapter 21) and virtual stack level (see Chapter 23) has a *protection word*. Its purpose is to allow the error handling subsystem to distinguish which environments were created before the error trap, and which were created after. This way, all environments that were created after the error trap was set will be deleted in case of an error. For example, consider the following code:

```
::
  ...
  1LAMBIND
  ...
  TEN ZERO_DO
    ERRSET ::
      ...
      1LAMBIND
      ...
      FIVE ONE_DO
        <suspect object is here>
      LOOP
      ABND
    ;
    ERRTRAP ::
      <error handling>
    ;
  LOOP
  ...
  ABND
;
```

If an error is generated, then the error will be trapped. The inner `DO`/`LOOP` and temporary environments will be deleted, thanks to the protection word.

When one of these environments is created, its protection word is set to zero. The word `ERRSET` increments the protection word of the most recent environment of each of the three kinds. This way, these environments now have a non-zero protection word. (The protection word was initialized to zero when the environment was created.)

The words `ERRTRAP` and `ERRJMP` delete these kinds of environments (from the newest to the oldest) until they find one (of each type) with a non-zero protection word. These environments were the ones that already existed before the setting of the error trap, because they have had their values increased by `ERRSET`. This way, all environments created after the setting of the trap (which still have the protection word as zero) are deleted. Another effect of `ERRTRAP` and `ERRJMP` is that they decrement the protection word of those first environments found with a non-zero protection word, so that the process works correctly if there are several levels of nesting.

22.2 Generating Errors

The error handling subsystem is invoked by the word `ERRJMP`. If an error trap was set, the error handler will be executed. If none was set, then the default one will be run.

In most cases, when you generate an error, you will let the default error handler deal with it. This default handler does a beep (if this feature is enabled), and displays a description of the error in a message box.

The displayed message depends on two things: the error number, which defines the error message (such as “Bad Argument Type” or “Too Few Arguments”) and the last stored command name.

This last stored command name is automatically stored by the `CK<n>` words described in Chapter 29. As mentioned there, if you are writing a program that is not part of a library, no command name should be stored, because otherwise an ugly name will be shown.

To define the error number, use the word `ERRORSTO`. It expects a bint as argument: the number of the error. The errors are listed in Appendix E.

There are some words that automate this process, generating some common errors, such as `SETTYPEERROR`. These words are listed in the reference section below. There are also shortcut words for generating some CAS error messages. These are described in Chapter 52.

Sometimes, however, it is desired to generate an error message that is not in the built-in error list. In order to do that, first you need to store the desired message by means of the command `EXITMSGSTO`. Then, store `#70000` as the error number. Note that there is a built-in bint, called `#EXITERR`, which contains that number. Now, just call `ERRJMP`.

The process above can be simplified by using the words `DO#EXIT` and `DO$EXIT`. The first takes a bint as argument, stores that number and calls `ERRJMP`. The latter is used with strings, it takes a string as argument and does the actions described in the previous chapter. However, both entries also call

AtUserStack, which tells the error handling system not to delete any objects in the stack. So, do not use this word if there are objects in the stack (put by your program) that should be deleted. The automatic deletion of non-user objects in the stack when an error occurs will be described in more detail in section 29.1.

22.3 Reference

22.3.1 General Words

Addr.	Name	Description
26067	ERRBEEP	($\rightarrow$) Beeps.
04CE6	ERROR@	($\rightarrow$ #) Returns current error number.
04D0E	ERRORSTO	(# $\rightarrow$) Stores new error number.
36883	ERROROUT	(# $\rightarrow$) Stores new error number and calls ERRJMP.
04D33	ERRORCLR	($\rightarrow$) Stores zero as new error number.
04ED1	ERRJMP	($\rightarrow$) Invokes error handling sub-system.
04E07	GETEXITMSG	($\rightarrow$ $) Gets EXITMSG (user defined error message).
04E37	EXITMSGSTO	($ $\rightarrow$) Stores $ as EXITMSG.
25EAE	DO#EXIT	(# $\rightarrow$) Stores new error number, does AtUserStack and then ERRJMP.
25EB0	DO%EXIT	(% $\rightarrow$) Same as above, but takes real number as argument.
25EAF	DO$EXIT	($ $\rightarrow$) Stores string as EXITMSG, #70000 as error number, does AtUserStack and then ERRJMP.
04EA4	ABORT	($\rightarrow$) Does ERRORCLR and ERRJMP.

Addr.	Name	Description
04E5E	ERRSET	(→) Sets new error trap.
04EB8	ERRTRAP	(→) Error trap marker. If no error happens, still removes all temporary environments created since ERRSET.
04D87	JstGetTHEMESG	(# → $) Fetches message from message table. To get a message from a library, use the formula: libnum*#100+msgnum. aka: JstGETTHEMSG
04D64	GETTHEMESG	(# → $) If #70000 then does GETEXITMSG, else does JstGetTHEMESG.
39332	(?GETMSG)	(# → $msg) (ob → ob) If the argument is a bint, does JstGETTHEMSG to fetch a message. Other arguments are returned unchanged.

22.3.2 Error Generating Words

Addr.	Name	Description
04FB6	SETMEMERR	Error 001h Generates "Insufficient Memory" error.
05016	SETROMPERR	Error 004h Generates "Undefined XLIB Name" error.
04FF2	SETPORTNOTAV	Error 00Ah Generates "Port Not Available" error.
26134	SYNTAXERR	Error 106h Generates "Invalid Syntax" error.
260C1	NOHALTERR	Error 126h Generates "HALT Not Allowed" error.
26116	SETCIRCERR	Error 129h Generates "Circular Reference" error.
262E2	SETSTACKERR	Error 201h Generates "Too Few Arguments" error.

Addr.	Name	Description
262DD	SETTYPEERR	Error 202h Generates "Bad Argument Type" error.
262D8	SETSIZEERR	Error 203h Generates "Bad Argument Value" error.
262E7	SETNONEXTERR	Error 204h Generates "Undefined Name" error.
2F458	SETIVLERR	Error 304h Generates "Undefined Result" error.
2F37B	SetIOPARErr	Error C12h Generates "Invalid IOPAR" error.
3721C	Sig?ErrJmp	(# $\rightarrow$) Calls `ERRJMP` if the error number is any of {13E 123 DFF}.
25F10	ederr	($\rightarrow$) Error handler for applications which use `savefmt1` to save the current display format. Calls `rstfmt1` and then errors out.

Chapter 23
The Virtual Stack

The HP49 has a "Virtual Stack" feature. It is a set of commands that can manipulate an RPN Stack: basically, you can save the stack and then restore it.

There exists, in fact, a stack of stacks (a metastack?). The topmost (and in normal conditions, the only) one is the normal RPN stack, in which the user enter objects, and from which commands take and return arguments. This stack will be referred as RPN stack. The set (or, more specifically, the stack) of stacks will be referred as "Virtual Stack", with uppercase initials.

You can push the RPN stack (or part of it), making these pushed objects a level of the "Virtual Stack". A level of the Virtual Stack will be called "virtual stack", with lowercase initials. After pushing the RPN stack, you can manipulate it in any way, and you can at any time restore the contents previously pushed. Or you can push another stack, thus having two stored virtual stacks, in addition to a "new" RPN stack which can be used independently.

Each of these pushed virtual stacks holds a number of objects, and the count of objects. The number of objects is determined when the virtual stack is pushed, and it is not possible to add more objects later. The words that return the virtual stack as a meta return this count, the others do not. When pushing, the words that push the stack as a meta allow you to push only part of the stack; the others push everything in the RPN stack. But you can pop as a meta a stack that was not pushed as one, or push a stack as a meta and pop is not being a meta. The only difference is that the count of elements may or may not be returned.

The Virtual Stack is used in nearly every HP49 application. It is extremely useful (and really fast) when you want to save immediately a complete stack, without using much memory.

It is the Virtual Stack that allows you to enter a full command line in an Input Form and get the results of that command line in the field, for example. Suppose in an InputForm you type `DROP`. You will get an error, "Too Few Arguments" even if the stack was not empty. Before the HP49 runs the command, it saves the stack into the Virtual Stack, then run the command. Once the command has been run, it restores the pushed virtual stack above the new one.

The Virtual Stack is located inside a string which is the first object in TEMPOB. It has a similar structure as a Local Variable stack. It is made with blocks, and is protected exactly like local variables. If you trap an error, the virtual stacks created inside the `ERRSET` and `ERRTRAP` will be automatically deleted, exactly as are local variable blocks. (See section 22.1.1 for more information.)

For examples of the application of the Virtual Stack, see the `DoL` list processor in section 19.3 and the HP48 Browser example in section 34.7. Following, there is a list of the commands that deal with the Virtual Stack.

23.1 Reference

Addr.	Name	Description
25F1E	PushVStack	(obn..ob1 → obn..ob1) Virtual Stack: (→ [obn..ob1]) Pushes the RPN stack onto the Virtual Stack. The RPN stack is unchanged.
25F1F	PushVStack&Clear	(obn..ob1 →) Virtual Stack: (→ [obn..ob1]) Does `PushVStack` and then clears the RPN stack.
25F1A	PopMetaVStackDROP	(→ obn..ob1) Virtual Stack: ([obn..ob1] →) Pops the topmost virtual stack into the RPN stack. The previous contents of the RPN stack are preserved. (The Meta in the name means that a count is returned, but the `DROP` removes it afterwards.)

Addr.	Name	Description
25F1B	PopVStack	(obm..ob1 → obn'..ob1') Virtual Stack: ([obn'..ob1'] →) Pops the topmost virtual stack into the RPN stack. The previous contents of the RPN stack are lost.
25F17	GetMetaVStackDROP	(→ obn..ob1) Virtual Stack: ([obn..ob1] → [obn..ob1]) Inserts the objects from the topmost virtual stack into the RPN stack. The Virtual Stack is unchanged. (The Meta in the name means that a count is returned, but it is removed by DROP.)
25F18	GetVStack	(obm..ob1 → obn'..ob1') Virtual Stack: ([obn'..ob1'] → [obn'..ob1']) Copies the topmost virtual stack into the RPN stack. The Virtual Stack is not changed, but the current RPN stack is lost.
26265	PushMetaVStack	(obn..ob1 #n → obn..ob1 #n) Virtual Stack: (→ [obn..ob1]) Pushes #n objects as a new virtual stack. Any other objects in the RPN stack are not pushed. The RPN stack is unchanged.
25F1D	PushMetaVStack&Drop	(obn..ob1 #n →) Virtual Stack: (→ [obn..ob1]) Does PushMetaVStack then drops the pushed objects. Any other objects present in the RPN stack are neither pushed nor dropped.

Addr.	Name	Description
25F19	PopMetaVStack	(→ obn..ob1 #n) Virtual Stack: ([obn..ob1] →) Insers the contents of the most recent virtual stack into the RPN stack, followed by the count. The previous contents of the RPN stack are not lost.
2624C	GetMetaVStack	(→ obn..ob1 #n) Virtual Stack: ([obn..ob1] → [obn..ob1]) Inserts the objects from the topmost virtual stack into the RPN stack, along with the count. The Virtual Stack is unchanged.
25F20	PushVStack&Keep	(obn..ob1 obm'..ob1' #m → obm'..ob1' #m) Virtual Stack: (→ [obn..ob1]) Pushes the contents of the RPN stack which do not belong to the meta (ie, are "above" it) into a new virtual stack, removing these elements, but keeping the meta.
25F21	PushVStack&KeepDROP	(obn..ob1 obm'..ob1' #m → obm'..ob1') Virtual Stack: (→ [obn..ob1]) Does PushVStack&Keep and then DROP.
25F1C	PopVStackAbove	(obm'..ob1' → obn..ob1 obm'..ob1') Virtual Stack: ([obn..ob1] →) Pops the contents of the topmost virtual stack (like PopMetaVStackDROP would have done) into the RPN stack, but *above* the current contents of the RPN stack. This undoes PushVStack&Keep (or PushVStack&KeepDROP).

Addr.	Name	Description
26215	DropVStack	(→) Virtual Stack: ([obn..ob1] →) Drops the topmost virtual stack from the Virtual Stack.
26229	GetElemTopVStack	(#i → obi) Virtual Stack: ([obn..ob1] → [obn..ob1]) Returns the ith object from the topmost virtual stack, counting from the top. "Counting from the top" means that object # 0 is the one at the highest-numbered level (n), # 1 is the one at level n-1, and so on. Note: no checking wheter #i is valid.
2626F	PutElemTopVStack	(new_ob #i →) Virtual Stack: ([obn..ob(n-i)..ob1] → [obn..new_ob..ob1]) Replaces the ith object from the topmost virtual stack with new_ob, counting from the top. Note: no checking wheter #i is valid.
26224	GetElemBotVStack	(#i → obi) Virtual Stack: ([obn..ob1] → [obn..ob1]) Returns the ith object from the topmost virtual stack, counting from the bottom. "Counting from the bottom" means that # 0 is the object in the lowest numbered level (generally thought of as 1), # 1 is at level 2, etc. Note: no checking wheter #i is valid.
2626A	PutElemBotVStack	(new_ob #i →) Virtual Stack: ([obn..obi..ob1] → [obn..new_ob..ob1]) Replaces the ith object from the topmost virtual stack with new_ob, counting from the bottom. Note: no checking wheter #i is valid.

Addr.	Name	Description
26233	`GetVStackProtectWord`	($\rightarrow$ #) Hacking stuff: Gets the protection word of the last VStack level.
2622E	`SetVStackProtectWord`	(# $\rightarrow$) Hacking stuff: Sets the protection word of the last VStack level.

Chapter 24
Memory Operations

The basic equivalents to the user commands `STO` and `RCL` are the words `CREATE`, `STO` and `@`:

Word	Stack and Action
`CREATE`	(`ob id` $\rightarrow$) Creates a variable with the name id and contents ob. An error occours if ob is or contains the current directory ("Directory Recursion"). This word does not check if there is already a variable with name id: even if there is, another one is created.
`STO`	(`ob id` $\rightarrow$) (`ob lam` $\rightarrow$) In the lam case, the temporary identifier is rebound to ob. An error occurs if the lam is unbound. In the id case, `STO` attempts to replace the contents of the variable named id with ob. If a variable with that name was not found, a new variable is created.
`@`	(`id` $\rightarrow$ `ob TRUE`) (`id` $\rightarrow$ `FALSE`) (`lam` $\rightarrow$ `ob TRUE`) (`lam` $\rightarrow$ `FALSE`) Attempts to return the contents stored in the variable or temporary identifier. Returns the stored object and `TRUE` if successful, or just `FALSE` if no variable or lam was found with that name. In the case of variables, searching starts in the current directory and works upwards through parent directories if necessary.

One problem with `STO` and `@` is that if you give, say, `SIN` as the argument, the whole body of the function is stored in the variable. For that reason, it is better to use `SAFESTO` and `SAFE@`, which work like `STO` and `@`, but they automatically convert ROM bodies into XLIB names (`SAFESTO`) and back again (`SAFE@`).

Note that the `SAFE` in these and other entries only means that they do the conversions described above. With other aspects, there is no safety in these entries.

There are many other words related to memory, which you will find in the list below.

24.1 Reference

24.1.1 Recalling, Storing and Purging

Addr.	Name	Description
0797B	@	(id/lam → ob T) (id/lam → F) Basic recalling function.
35C2C	DUP@	(id/lam → id/lam ob T) (id/lam → id/lam F) Does DUP then @.
35A5B	SAFE@	(id/lam → ob T) (id/lam → F) For lams does @. For ids does ?ROMPTR> to the ob found.
35A56	DUPSAFE@	(id/lam → id/lam ob T) (id/lam → id/lam F) Does DUP then SAFE@.
25EF7	SAFE@_HERE	(id → ob F) (id → T) Same as SAFE@, but works only in the current directory.
2F064	Sys@	(ID → ob T) (ID → F) Switches temporarily to the HOME directory and executes @ there.
2F2A3	XEQRCL	(id → ob) Same as SAFE@, but errors if variable is not found. Also works for lams, but you get the wrong error.
2F24E	LISTRCL	({path id} → ob) Recalls from specified path.

Addr.	Name	Description
07D27	STO	(ob id/lam →) For ids this assumes ob is not pco. If replacing some object, that object is copied to TEMPOB and pointers are updated. For lams: Errors if lam is unbound.
35A29	SAFESTO	(ob id/lam →) For ids, does ?>ROMPTR to the object before storing.
2F380	SysSTO	(ob ID →) Switches temporarily to the HOME directory and executes STO there.
25E79	XEQSTOID	(ob id/lam →) Same as SAFESTO, but will only store in the current directory and will not overwrite a directory. aka: ?STO_HERE
25F0C	XEQStoKey	(ob ID →)
3E823	xSTO>	(ob id →) (ob symb →) Like xSTO, but if the level 1 argument is symbolic, use the first element of it as the variable to write to.
0BD007	^PROMPTSTO1	(id/lam →) Inputs value for a variable and stores it.
085D3	REPLACE	(newob oldob → newob) Replaces oldob (in memory) with newob.
08C27	PURGE	(id →) Purges variable. Does no type check first.
25E78	?PURGE_HERE	(id →) Like PURGE, but only works in current directory.
1D3006	^SAFEPURGE	(idnt/lam →) Purge idnt/lam if it exist.
08696	CREATE	(ob id →) Creates a variable in the current directory. Errors if id is or contains current directory. Assumes id is not a pco.

Addr.	Name	Description
25EC4	DoHere:	(→) Next object in the runstream is evaluated for the current directory only.
36A8B	'LAMLNAMESTO	(ob →) STO to LAM LAMLNAME.

24.1.2 Directories

Addr.	Name	Description
25EA1	CREATEDIR	(id →) Creates an empty directory. Calls ?PURGE_HERE first to delete the original.
08326	LASTRAM-WORD	(rrp → ob T) (rrp → F) Recalls first object in directory.
25EE7	LastNonNull	(rrp → ob T) (rrp → F) Recalls first object in directory (not null named).
08376	PREVRAM-WORD	(ob → ob' T) (ob → F) Recalls next object in directory.
25EF2	PrevNonNull	(ob → ob' T) (ob → F) Recalls next object in directory (not null named).
082E3	RAM-WORDNAME	(ob → id) Recalls name of object in current directory.
25F14	XEQPGDIR	(id →) Purges a directory. Checks references, etc. first.
2F296	XEQORDER	({id1 id2..} →) Orders the variables in the directory by moving the given variables to the beginning of the directory.
25EB9	DOVARS	(→ {id1 id2..}) Returns list of variables from current directory.

Addr.	Name	Description
25EB8	DOTVARS%	(% → {}) Returns a list of variables in the current directory with user type given by the number. Internal TVARS if a single number was given.
0BD002	^DOTVARS{}	({# #' ...} → {}) Returns a list of variables in the current directory with user type given by any of the numbers in the list. This is the core of the TVARS program.
25EF1	PATHDIR	(→ {HOME dir1 dir2..}) Returns current path.
2F265	UPDIR	(→) Goes to parent directory.
08D5A	CONTEXT@	(→ rrp) Recalls current directory.
08D08	CONTEXT!	(rrp →) Sets new current directory.
08DD4	SYSRRP?	(rrp → flag) Is rrp HOME?
08D92	HOMEDIR	(→) Sets HOME as current directory. aka: SYSCONTEXT
3712C	SaveVarRes	(→) Binds current and last directories to two null-named lams.
37186	RestVarRes	(→) First sets HOME as both the current and last directories (in case an error happens). Then, restores the current and last directories from 1LAM and 2LAM.

24.1.3 The Hidden Directory

Addr.	Name	Description
3714A	SetHiddenRes	`( → )` Sets the hidden directory as the current and last directories.
370C3	WithHidden	`( → ? )` Executes next command in hidden directory.

370AF	RclHiddenVar	`( id → ob T )`
		`( id → F )`
		Recalls variable in hidden directory. Same as
		`:: WithHidden @ ;`

37104	StoHiddenVar	`( ob id → )`
		Stores variable in hidden directory. Same as
		`:: WithHidden STO ;`

37118	PuHiddenVar	`( id → )`
		Purges variable in hidden directory. Same as
		`:: WithHidden PURGE ;`

24.1.4 Temporary Memory

The objects in the stack are in a area called “temporary memory”. As the name says, it is intended for temporary storage.

When you duplicate an object in the stack, you do not actually create a copy of it: the stack contains only pointers to objects, and only this pointer is duplicated.

When you modify an object, most commands automatically make a new copy of the object in question and modify the copy. In other words, if you enter a string in the stack, press ENTER and edit the string, you have two different strings now. This only happens because a copy of the string was made before editing it. If the copy was not made, the two strings would have been modified, because they were actually the same object.

There are a few commands that do not make a copy of the object before editing it. This means that all copies of the object, in the stack or even stored in memory will be modified at the same time. Sometimes this is desired, sometimes not. These commands are sometimes called “bang type”. When this kind of command appears in this book this is noted in their description. When you use these commands, you must be careful not to modify too much objects simulateneously... You can use the commands TOTEMPOB or CKREF to make another copy of the object: with this, it becomes safe to use this “bang type” commands.

Addr.	**Name**	**Description**
06657	TOTEMPOB	(ob → ob') Copies object to TEMPOB and returns pointer to the new copy.
35C90	TOTEMPSWAP	(ob1 ob2 → ob2' ob1) Does TOTEMPOB then SWAP.
25E9F	CKREF	(ob → ob') If object is in TEMPOB, is not embedded in a composite and not referenced, does nothing. Else copies it to TEMPOB and returns the copy.
3700A	SWAPCKREF	(ob1 ob2 → ob2 ob1') Does SWAP then CKREF.
06B4E	INTEMNOTREF?	(ob → ob flag) If the object is in TEMPOB area, is not embedded in a composite and is not referenced, returns the object and TRUE, otherwise returns the object and FALSE.
01E0E8	~INTEMPOB?	(ob → ob flag)

Chapter 25
Time and Alarms

This chapter contains a list of entries related to times, dates and the internal list of alarms.

25.1 Reference

Addr.	Name	Description
26120	SLOW	(→) 15 millisecond delay.
26125	VERYSLOW	(→) 300 millisecond delay.
2F37E	SORTASLOW	(→) 1.2 second delay (4 x VERYSLOW).
2612A	VERYVERYSLOW	(→) 3 second delay.
2F2D4	dowait	(%secs →) Waits specified number of seconds.
3005E	%>HMS	(% → %hms) Converts from decimal to H.MMSS format.
30912	%%H>HMS	(%% → %%hms) Same as %>HMS, but for long reals.
30077	%HMS>	(%hms → %) Converts from H.MMSS format to decimal.
3008B	%HMS+	(%hms1 %hms2 → %hms) Adds time in hms format.
300B3	%HMS-	(%hms1 %hms2 → %hms) Subtracts time in hms format.
2EECF	TOD	(→ %time) Returns current time.

Addr.	Name	Description
2F388	VerifyTOD	(%time → %time) Checks for validaty of time. Errors if not valid.
2EED0	DATE	(→ %date) Returns current date.
2EED2	DATE+DAYS	(%date %days → %date') Adds specified number of days to date.
2EED1	DDAYS	(%date1 %date2 → %days) Returns number of days between two dates.
2EED7	CLKTICKS	(→ hxs) Returns tick count. aka: SysTime
2EED3	TIMESTR	(%dt %tm → "dy dt tm") Returns string representation of time, using current format. Example: "WED 06/24/98 10:00:45A"
2F329	Date>d$	(%date → $) Returns string representation of date, using current format.
2F381	TOD>t$	(%time → $) Returns string represent the time, using current format.
2F1AB	Date>hxs13	(%date → hxs) Converts date to ticks.
2F003	(Ticks>Date)	(hxs → %date) Returns date from hxs of internal alarm list format.
2F002	(Ticks>TOD)	(hxs → %time) Returns time from hxs of internal alarm list format.
2F004	(Ticks>Rpt)	(hxs → %rpt) Converts hxs in internal alarm list format to repetition interval.

25.1.1 Alarms

The internal alarms list has this format:

```
{ { hxs action } { ... } ... }
```

The length of each hxs is 24 nibbles. The least significant 13 nibbles represent the tick value for the time and date. The next 10 nibbles represent the repeat interval, if any. The most significant nibble represents the status of the alarm (pending, acknowledged, etc.).

Addr.	Name	Description
2F178	ALARMS@	`( → {} )` Returns internal alarms list.
2F37F	STOALM	`( %date %time acti %rep → % )` Stores an alarm. %repeat is the number of ticks between every repetition. Since there are 8192 ticks in a second, 60 seconds in a minute, and 60 minutes in an hour, to make an alarm that repeats every hour, %repetition would be 8192*60*60 = 29491200. Returns real number representing the position of the alarm in the list.
2F0AC	PURGALARM%	`( % → )` Internal DELALARM.
2F314	RCLALARM%	`( %n → {} )` Recalls nth alarm. List is in the format of STOALARMLS.
25FA9	ALARM?	`( → flag )` Returns TRUE if an alarm is due.

Chapter 26
System Functions

Following, there is a list of functions dealing with the system, such as configuring some aspects of the calculator and turning the calculator off. The functions dealing with user and system flags are also described here.

26.1 Reference

26.1.1 User and System Flags

Addr.	Name	Description
2614D	SetSysFlag	(# →) Sets the system flag with number #.
26044	ClrSysFlag	(# →) Clears the system flag with number #.
26170	TestSysFlag	(# → flag) Returns TRUE if system flag is set.
26152	SetUserFlag	(# →) Set the user flag with number #.
26049	ClrUserFlag	(# →) Clear the user flag with number #.
26175	TestUserFlag	(# → flag) Returns TRUE if user flag is set.
2F259	RCLSYSF	(→ hxs) Recalls system flags from 1 to 64.
2F25F	(STOSYSF)	(hxs →) Stores system flags from 1 to 64.
2F23E	DOSTOSYSF	(hxs →) Stores system flags from 1 to 64, checking for changes in LASTARG flag.

Addr.	Name	Description
2F25A	(RCLSYSF2)	(→ hxs) Recalls system flags from 65 to 128.
2F260	(STOSYSF2)	(hxs →) Stores system flags from 65 to 128.
2F25B	RCLUSERF	(→ hxs) Recalls user flags from 1 to 64.
2F261	(STOUSERF)	(hxs →) Stores user flags from 1 to 64.
2F25C	(RCLUSERF2)	(→ hxs) Recalls user flags from 65 to 128.
2F262	(STOUSERF2)	(hxs →) Stores user flags from 65 to 128.
2F3A9	(STOALLF)	(hxs_usr hxs_sys →) Stores user and system flags from 1 to 64. First is user flags, second is system flags.
2F3AA	(STOALLF2)	(hxs_sys1 hxs_usr1 hxs_sys2 hxs_usr2 →) Expects 4 hxs and stores them as user and system flags.
3B76C	(DOSTOALLF2)	({} →) Stores system and user flags. Expects a list with two or four hxs. The first two are the system and user flags, respectively, from 1 to 64. The last two, if present, are the system and user flags, respectively, from 65 to 128.
25F23	SaveSysFlags	(→) Save system flags in a virtual stack.
25F22	RestoreSysFlags	(→) Restore system flags from virtual stack, popping that level.
2ABF0	RunSafeFlags	Run Stream: (ob →) Evaluates the next object in the runstream, but saves and restores the system flags around it. Uses DoRunSafe. This is very useful.

Addr.	Name	Description
2AB69	RunInApprox	Run Stream: (ob →) Eval next object in runstream with system flags 20, 21 clear and 22, 105, 102, 120 set.
2AC0E	DoRunSafe	(ob → hxs1 hxs2) Evaluate ob and put the system flags as they were before the evaluation on the stack. Used by RunSafeFlags and RunSafeFlagsNoError.
2ABD7	RunSafeFlagsNoError	Run Stream: (ob →) :: 'R DoRunSafe 2DROP ;
2EFA5	DOHEX	(→) Switch stack display format of HEX strings to hexadecimal.
2EFA8	DODEC	(→) Switch stack display format of HEX strings to decimal.
2EFA6	DOBIN	(→) Switch stack display format of HEX strings to binary.
2EFA7	DOOCT	(→) Switch stack display of HEX strings to octal.
2EFBF	BASE	(→ #) Returns #10h, #10d, #10b or #10o. In decimal terms, 16 for hexadecimal base, 10 for decimal base, 8 for octal base or 2 for binary base.
2605D	DOSTD	(→) Internal version of user word STD.
26053	DOFIX	(# →) Internal version of user word FIX.
26058	DOSCI	(# →) Internal version of user word SCI.
2604E	DOENG	(# →) Internal version of user word ENG.
261A7	savefmt1	(→) Saves the current number format, and changes to STD mode.

Addr.	Name	Description
261A2	rstfmt1	(→) Restores the number format saved by savefmt1. Only one set of flags can be saved, there is no nesting of these entries.
2FFDB	SETRAD	(→) Set angular mode to RAD.
25EF3	RAD?	(→ flag) Is angular mode RAD?
2FFBD	SETDEG	(→) Set angular mode DEG.
2FFEF	SETGRAD	(→) Set angular mode GRAD.
25EBA	DPRADIX?	(→ flag) Returns TRUE if current radix is ".".

26.1.2 General Functions

Addr.	Name	Description
25EB2	DOBEEP	(%freq %dur →) Beeps. Analog to user function BEEP.
261AC	setbeep	(#ms #Hz →) Also beeps.
041A7	TurnOff	(→) Internal OFF.
041ED	DEEPSLEEP	(→ flag) Puts HP into deepsleep mode. Returns TRUE if "Invalid Card Data" message.
01118	LowBat?	(→ flag) Returns TRUE if low battery.
0426A	ShowInvRomp	(→) Flashes "Invalid Card Data" message.
2EE5D	?FlashAlert	(→) Displays system warnings.
05F42	GARBAGE	(→) Forces garbage collection.
05F61	MEM	(→ #) Returns amount of free memory in nibbles. Does not do garbage collection. (The user word does.)

Addr.	Name	Description
05902	OSIZE	(ob → #) Returns object size in nibbles. Forces garbage collection.
05944	OCRC	(ob → #nib hxs) Returns size in nibbles and checksum as hxs.
2F257	OCRC%	(ob → hxs %bytes) Returns checksum and size in bytes.
2F267	VARSIZE	(id → hxs %bytes) Returns checksum and size in bytes of specified variable.
394C8	INHARDROM?	(ob → ob flag) Is object address < #80000h?
05AB3	CHANGETYPE	(ob #prolog → ob') Changes prolog of object, does TOTEMPOB.
25F90	>LANGUAGE	(# →) Sets the current language for messages. Internal version of x→LANGUAGE.
25F95	LANGUAGE>	(→ #) Returns the current language for messages. Internal version of the xLANGUAGE→ command.
256BE	NOBLINK	(→) Clears the BLINKFLAG, SysNib5.
25E71	?BlinkCursor	(→) Makes the cursor Blink if in App-mode or Editline.

Chapter 27
Serial Communications

The entries listed here allow the programmer to write programs that communicate with other machines via the HP49G serial interface.

27.1 Reference

Addr.	Name	Description
2EEBB	SENDLIST	({} →) Internal SEND.
2EEBC	GETNAME	($/id/lam →) Internal KGET.
2EEBD	DOFINISH	(→) Internal FINISH.
2EEBE	DOPKT	($ $' →) Internal PKT.
2EEC1	DOBAUD	(% →) Internal BAUD.
2EEC2	DOPARITY	(% →) Internal PARITY.
2EEC3	DOTRANSIO	(% →) Internal TRANSIO.
2EEC4	DOKERRM	(→ $) Internal KERRM.
2EEC5	DOBUFLEN	(→ % 0/1) Internal BUFLEN.
2EEC6	DOSBRK	(→) Internal SBRK.
2EEC7	DOSRECV	(% →) Internal SRECV.

Addr.	Name	Description
2EEC9	CLOSEUART	(→) Internal CLOSEIO.
2EECB	DOCR	(→) Internal CR.
2EECD	DODELAY	(% →) Internal DELAY.
2F31A	APNDCRLF	($ → $') Appends carriage return and line feed to string.
2716D	StdIOPAR	(→ {}) Default IOPAR: { 9600 0 0 0 3 1 }.
2EEBF	GetIOPAR	(→ %baud % % % % %) Recalls IOPAR and explodes it into the stack.
2F062	StoIOPAR	({} →) STO the list of IO parameters in the HOME directory in the variable IOPAR.
2F37B	SetIOPARErr	(→) Throws the IOPAR error: "Invalid IOPAR".
2F34F	KVISLF	($ → $') Like KVIS, but insert <cr> in front of each newline for PC's.
2F34E	KVIS	($ → $') Translate special characters into digraphs for ASCII transfer to a PC.
2F34D	KINVISLF	($ → $') Translate digraphs in the string to characters. and remove <cr> from th end of lines.
2F389	VERSTRING	(→ $) Returns version string.

Chapter 28
The HP49 Filer

The HP49 File Manager (Filer for short) allows the programmer to write various applications that deal with files.

Two built-in applications that use the filer are the File Manager and the Font Browser.

28.1 Using the Filer

The general entry to call the filer is `^FILER_MANAGERTYPE`. It takes three arguments: `Filer_Type`, `Filer_Path` and `Filer_List`.

28.1.1 The Filer_Type Argument

This argument allows to select the object types that should be displayed. It is a list of the object prologue addresses that will be allowed.

As an example, the HP49 Font Browser only displays fonts, directories and backup objects. So, `Filer_Type` is specified as `{ DOFONT DORRP DOBAK }`.

If you want to browse all kinds of objects, then this parameter should be just `{ ZERO }`. Since this is very common, there is an entry that will supply this list as argument to `^FILER_MANAGERTYPE`. It is called `^FILER_MANAGER`. Using this entry, you only specify the other two arguments.

28.1.2 The Filer_Path Argument

This argument specifies the initial path. It can be:

Value	Meaning
`{ }`	start in `HOME`
`{ FOO.DIR }`	start in `HOME/FOO.DIR`

Value	Meaning
`:n:{ }`	start in port n
`:n:{ FOO }`	start in backup `FOO` in port n.

28.1.3 The Filer_List Argument

This argument specifies the menu keys and hard key assignments. It is a list with one element for each menu key. Each menu key is represented by a list with three to five arguments.

The general structure is like this:

```
{
  {
    (Item 1)
    Name_Item
    Location_Item
    Action_Item
    [ ExtraProgram_Item (if 16 <= Action_Item <= 23) ]
    [ Key_Shortcut ]
  }
  {
    (Item 2)
    ...
  }
  ..
}
```

Each of the elements in the sublists will be described now.

28.1.3.1 Name_Item

This specifies what will be displayed in the menu. It can be either a string, a grob or a program which when evaluated returns a string or a grob.

28.1.3.2 Location_Item

This allows you to control when the action bound to this menu can be run. This is either a bint or a program that returns a bint when evaluated.

There are five possible values, which are listed in the table below. The "Constant" column lists the name of a constant (defined in the `filer.h` file in the `includes` directory of the examples) that you should use when programming.

Value	Constant	Meaning
0	`fEverywhere`	The action can be run anywhere.
1	`fVar`	The action can be run only if the user is browsing the `HOME` directory or one of its subdirectories.
2	`fNoLib`	The action cannot be run if the user is browsing a library.
3	`fNoBackup`	The action cannot be run if the user is browsing a backup object in a port.
4	`fHomePort`	The action can only be run in the root of a port.

28.1.3.3 Action_Item

This will define what will happen when the user presses the softkey corresponding to that menu or the hardkey assignemnt (see section 28.1.3.5).

It is a bint, or a program that returns a bint when evaluated.

It is possible to call a built-in function of the Filer, or define your own. The table below lists the built-in actions available. Again, "Constant" is the name of a constant defined in `filer.h`.

Value	Constant	Action
0	`cBip`	Beeps.
1	`cInfo`	Not implemented in the HP49G.
2	`cHexa`	Not implemented in the HP49G.
3	`cView`	Views the object.
4	`cArbo`	Shows the directory tree.
5	`cUp`	Moves the highlight up.
6	`cMaxUp`	Moves the highlight to the first item.
7	`cDown`	Moves the highlight down.
8	`cMaxDown`	Moves the highlight to the last item.
9	`cSelect`	Marks the selected variable.
10	`cUpDir`	Goes to the parent directory.
11	`cDownDir`	Visits the highlighted directory.
12	`cPreviousMenu`	Displays the previous menu page.
13	`cNextMenu`	Displays the next menu page.
14	`cEVAL`	Evalutes the highlighted variable.
15	`cSwapHeader`	Toggles between the two available header lines.
24	`cDetails`	Toggles between showing information on the variables or just their names.
25	`cEDIT`	Edits the selected variable.
26	`cCOPY`	Copies the selected variable.

Value	Constant	Action
27	`cMOVE`	Moves the selected variable.
28	`cRCL`	Recalls the contents of the select variable.
29	`cPURGE`	Purges the selected variable.
30	`cRENAME`	Renames the selected variable.
31	`cCRDIR`	Creates a directory.
32	`cORDER`	Reorders the variables in the current directory.
33	`cSEND`	Sends the select variable using Kermit.
34	`cHALT`	Suspends the filer temporarily.
35	`cEDITB`	Edits the select variable in the most appropriate editor.
36	`cRECV`	Receives a variable using Kermit.
37	`cQUIT`	Exits the filer.
38	`cPageUp`	Scrolls the contents of the filer one screen up.
39	`cPageDown`	Scrolls the contents of the filer one screen down.
40	`cNewObject`	Creates a new variable.
41	`cSort`	Opens a dialog with several options for sorting the variables.

To run a custom program, the `Action_Item` argument will be in the range 16–23. Each of these seven values specifies what will be in the stack and how your program is going to be called.

The table below lists the calling methods:

Value	Description
16	Recalls only the current path. 1: Path
17	Recalls the name and the contents of the currently selected object. 3: Path 2: Object 1: Name
18	Equivalent to the above, but deals with multiple selected objects. $2n+2$: Path $\vdots$ 5: Object 2 4: Name 2 3: Object 1 2: Name 1 1: Number of objects (bint)

Value	Description
19	The program is called once for each object. For each object, puts the same that calling method #17 puts.
20	Recalls only the name of the current object. 2: Path 1: Name
21	Recalls all the selected names. $n+2$: Path $\vdots$ 3: Name 2 2: Name 1 1: Number of names (bint)
22	Recalls the current object only in a string of addresses. 2: Path 1: String
23	Recalls the selected objects in a string of addresses. 2: Path 1: String

Custom calls 22 and 23 will not be described here, as they are not very useful and somewhat more difficult to use.

When the program is called on a library at the root of a port, some special rules apply to the name:

- For calls 17 and 18, the name will be the title of the library. (Like "Emacs 1.09 CD&Pivo".)
- For call 19, the name will be an "L" plus the number of the library, for example, "L1790".
- For calls 20 and 21, the name will the library number as a real number.

28.1.3.4 ExtraProgram_Item

When using a custom program, this element holds the program to be called.

There are some additional features that can be useful:

- If you launch the program from VAR, your program will start in the current browsed directory.

- A program can only be called on a selected object, except for call 16 which can be run in an empty directory.
- By default, once a program has been run, the screen will be refreshed, the working directory will be parsed again and the current selection will be lost. You can prevent that by leaving `FALSE` in the stack. Example:

```
{ "INFO"
  fEverywhere
  BINT20
  :: SWAPDROP (Remove the path)
     "Name selected is:" DISPROW1
     ID>$ DISPROW2
     FALSE
  ;
}
```

- If you want to force the Filer to exit after the program is run, leave "`TakeOver`" in the stack. Example:
  `{ "QUIT" fEverywhere BINT16 :: DROP ' TakeOver ; }`
  which is equivalent to this, which uses the built-in call:
  `{ "QUIT" fEverywhere cQUIT }`

28.1.3.5 Key_Shortcut

Use this argument to assing a program to a key. This argument is a bint in the form `# axx`, where `a` is 0 or 1, meaning without alpha and with, respectively. `xx` is the key code plus optionally `#40` for the Left Shift or `#C0` for the right shift.

If you want to assign your program to LeftShift + TOOL, the number will be `#049`: `#09` representing the TOOL key, and `#40` for the LeftShift.

NOTE: This argument must be the fifth of the list. So, if you are using a built-in call you will have to define the entry as something like this:
`{ "TITLE" fEverywhere cQUIT TakeOver # 12F }`
This will assing the program to Alpha-ON.

28.2 Reference

Addr.	Name	Description
067004	ˆFiler	(→) Calls the standard filer.
06D004	ˆFILER_MANAGER	({path} {args} →) Customized Filer, browsing all object types.
06E004	ˆFILER_MANAGERTYPE	({types} {path} {args} →) {args} = { item1 item2 ... } item = {name loc action [prog] [key]} ... } Customized filer for selected types only.

Part III

Input and Output

Chapter 29
Checking for Arguments

In System RPL, it is very important to check if all arguments required by a program are present in the stack, and if they are of a valid type, when that program is directly accessible to the user. In User RPL, you do not have to worry about this: it is done automatically. In System RPL, very few commands do that, so this task is left for the programmer. This may seen at first a disadvantage, but it is in fact an advantage: you just need to check the arguments once, in the beginning of the program. This generates a fast code, differently from User RPL where the arguments are checked in every command.

29.1 Number of Arguments

To check for a specific number of arguments, use one of the following commands. They check if there are enough arguments in the stack, and produce a "Too Few Arguments" error if not.

Command	**When to use**
`CK0`, `CK0NOLASTWD`	No arguments required
`CK1`, `CK1NOLASTWD`	One argument required
`CK2`, `CK2NOLASTWD`	Two arguments required
`CK3`, `CK3NOLASTWD`	Three arguments required
`CK4`, `CK4NOLASTWD`	Four arguments required
`CK5`, `CK5NOLASTWD`	Five arguments required

The `CK<n>` commands save the name of the command in which they are executed, and if an error happens, that name is displayed. (For more details, see Chapter 22.) This means they should only be used in libraries, because if they are not part of a library and there is an error, the error will be shown as something like "`XLIB 1364 36 Error:`". In programs that are not a part of a library, use `CK<n>NOLASTWD,` which does not save the name of the command.

Besides checking for the specified number of arguments, these words also “mark” the stack in a way that, if an error happens, the objects that were pushed in the stack by your program can be removed, leaving no junk in the stack. This works by “marking” the stack above the n^{th} level, where n is the number of required arguments. For example, if your program uses `CK2` or `CK2NOLASTWD` and there are three arguments in the stack, you can image the stack like this:

3:	10.
2:	3.
1:	5.5

This mark is not fixed at this level; instead it moves as elements are pushed or popped. Here is the stack after the program pushes the bint 1:

4:	10.
3:	3.
2:	5.5
1:	¤ 1h

Now, if an error happens in the program, all objects “below” the mark are dropped. This removes all objects pushed by the program, and also the program arguments if they are still in the stack. This is the standard HP49G behavior.

Besides checking for a number of arguments and providing for error recovery, these words also save the arguments as the last arguments, recoverable via the `LASTARG` User command, provided this is enabled. If an error occours and it is enabled, then the arguments are automatically restored.

For user-acessible programs that take no arguments, you should nevertheless use `CK0` (or `CK0NOLASTWD` if it is not part of a library), to mark all the objects in the stack as of user ownership and mark the stack for error recovery.

If your program uses a stack-defined number of arguments (like `DROPN`), use the words `CKN` or `CKNNOLASTWD`. These words first check for a real number in level one, and then for the specified number of objects in the stack. The stack is marked at level two, but only the real number is saved in LAST ARG. The real is converted to a bint.

29.2 Argument Type

The words CK&DISPATCH1 and CK&DISPATCH0 are used to allow your program to do different actions based on the types of arguments given to it. They are used like this:

```
...
CK&DISPATCH1
  #type1 action1
  #type2 action2
  #type3 action3
  ...
  #type_n action_n
;
```

The type/action pairs are terminated by a SEMI (;). If after the dispatching you want to do some more actions for all argument types, you will need to enclose the whole CK&DISPATCH1 block in another secondary.

This is how CK&DISPATCH0 works: it checks if the stack matches the definitions in #type1. If it does, action1 is executed, after which program execution resumes after SEMI. (Each action must be a single object, so if you want to do more than one action, all of them must be included in a secondary, i.e., between :: and ;.) If the type definition does not match the stack, then #type2 is checked, and so on. If no match was found, a “Bad Argument Type” error is generated.

Even when your program accepts only one combination of arguments, this command is still useful for checking if the arguments are of the given type.

The difference between CK&DISPATCH0 and CK&DISPATCH1 is that the latter, after completing the first pass unsuccessfully, strips all the tags from the arguments, converts zints to reals, and does a second pass. Only after the second pass without a match the “Bad Argument Type” error is generated.

Each type definition is a bint like this: #nnnnn. Each n is an hexadecimal number representing the object in one position of the stack, according to the table below. The first n represents the object in level five, the second in level four, and so on. This way, #00201 represents a complex number in level three, any object in level two and a real number in level one; #000A6 represents a hxs in level two and an id in level one. There are also two-digit object type numbers, ending in F. Each time you use one of these, the number of arguments that can be checked is reduced. For example, #13F4F represents a real number in level three, an extended real in level 2 and an extended complex in level one.

Dispatch Code	User Type	Object type
0	n/a	Any object
1	0	Real number
2	1	Complex number
3	2	String
4	3, 4 or 29	Array or matrix
5	5	List
6	6	Identifier (global)
7	7	LAM (Temporary identifier)
8	8, 18 or 19	Secondary
9	9	Symbolic
A	n/a	Symbolic class
B	10	Hex string
C	11	Graphics object (GROB)
D	12	Tagged object
E	13	Unit object
0F	14	Rompointer (XLIB name)
1F	20	Bint
2F	15	Directory
3F	21	Extended real
4F	22	Extended complex
5F	23	Linked array
6F	24	Character
7F	25	Code object
8F	16	Library
9F	17	Backup object
AF	26	Library data
BF	27	Access pointer
CF	30	Font object
DF	27	Minifont object
EF	27	External object 4 (unused)
FF	28	ZINT

There are also the words `CK<n>&Dispatch`, where `<n>` is a number from one to five. These words combine `CK<n>` with `CK&DISPATCH1`. Because they use `CK<n>` (and thus save the last command name), they should only be used in library commands.

29.2.1 Examples

By disassembling and studying built-in words, you can learn a lot. Not only about argument checking, but also about many other things.

The TYPE command provides an example of dispatching. Here is its disassembly:

```
::
  CK1
  ::
    CK&DISPATCH0
    real       %0
    cmp        %1
    str        %2
    arry       XEQTYPEARRY
    list       %5
    id         %6
    lam        %7
    seco       TYPESEC (returns 8, 18 or 19)
    symb       %9
    hxs        %10
    grob       %11
    TAGGED     %12
    unitob     %13
    rompointer %14
    BINT31     %20  (#)
    BINT47     %15
    # 3F       %21  (%%)
    # 4F       %22  (C%%)
    # 5F       %23  (LNKARRAY)
    # 6F       %24  (CHR)
    # 7F       %25  (CODE)
    # 8F       %16
    # 9F       %17
    # AF       %26  (Library Data)
    # CF       % 30 (Font)
    # FF       % 28 (ZINT)
    any        %27  (External)
  ;
  SWAPDROP
;
```

In this case, it would have been possible to use CK&DISPATCH1 instead of CK&DISPATCH0, because tagged objects are explicitly listed on the table.

Since the last item on the list is any, type 27 is returned for any other object not listed.

Since TYPE is part of a library, the command CK1&Dispatch could have been used. The reason it did not is to save ROM space. The inner composite is actually the body of the System RPL command XEQTYPE. This way, System RPL programmers can call a function to return the type of the object, without the overhead of checking if *there is* an object, and without no need to duplicate the dispatching mechanism.

Note the object names. They are aliases for built-in bints. See Chapter 2 for a list of built-in bints.

29.3 Reference

Addr.	Name	Description
262B0	CK0	(→) Saves current command to LASTCKCMD. Marks stack below level 1 to STACKMARK.
262B5	CK1	(ob → ob) Saves current command to LASTCKCMD. Verifies that there is at least one object in the stack, if not generates a "Too Few Arguments" error. Saves stack mark to STACKMARK. If Last Arg is enabled then saves the argument.
262BA	CK2	(ob1 ob2 → ob1 ob2) Like CK1, but checks for at least two arguments.
262BF	CK3	(ob1...ob3 → ob1...ob3) Like CK1, but checks for at least three arguments.
262C4	CK4	(ob1...ob5 → ob1...ob5) Like CK1, but checks for at least four arguments.
262C9	CK5	(ob1...ob5 → ob1...ob5) Like CK1, but checks for at least five arguments.
262CE	CKN	(ob1...obn %n → ob1..obn #n) Checks for a real in level one. Then checks for that number of arguments. Finally, converts the real to a bint.
26292	CK0NOLASTWD	(→) Like CK0, but does not save current command.

Addr.	Name	Description
26297	CK1NOLASTWD	(ob $\to$ ob) Like CK1, but does not save current command.
2629C	CK2NOLASTWD	(ob1 ob2 $\to$ ob1 ob2) Like CK2, but does not save current command.
262A1	CK3NOLASTWD	(ob1...ob3 $\to$ ob1...ob3) Like CK3, but does not save current command.
262A6	CK4NOLASTWD	(ob1...ob4 $\to$ ob1...ob4) Like CK4, but does not save current command.
262AB	CK5NOLASTWD	(ob1...ob5 $\to$ ob1...ob5) Like CK5, but does not save current command.
25F25	CKNNOLASTWD	(ob1...obn %n $\to$ ob1..obn #n) Like CKN, but does not save current command.
2631E	CK&DISPATCH0	($\to$) Dispatches on stack argument.
26328	CK&DISPATCH1	($\to$) Dispatches on stack arguments, stripping tags and converting reals to ZINTS if necessary.
26323	CK&DISPATCH2	($\to$) Equivalent to CK&DISPATCH1.
26300	CK1&Dispatch	($\to$) Combines CK1 with CK&DISPATCH1.
26305	CK2&Dispatch	($\to$) Combines CK2 with CK&DISPATCH1.
2630A	CK3&Dispatch	($\to$) Combines CK3 with CK&DISPATCH1.
2630F	CK4&Dispatch	($\to$) Combines CK4 with CK&DISPATCH1.
26314	CK5&Dispatch	($\to$) Combines CK5 with CK&DISPATCH1.
25F9A	0LASTOWDOB!	($\to$) Clears command save by last CK<n> command. aka: 0LASTOWDOB!, 0LastRomWrd!
2EF6C	AtUserStack	($\to$) :: CK0NOLASTWD 0LASTOWDOB! ;
25E9E	CK1NoBlame	($\to$) :: 0LASTOWDOB! CK1NOLASTWD ;

Addr.	Name	Description
354CB	'RSAVEWORD	(→) Stores first object in the composite above the actual to LASTCKCMD. aka: 'RSaveRomWrd
26319	EvalNoCK	(comp → ?) Evaluates composite without saving as current command. If first command is CK<n>&Dispatch it is replaced by CK&DISPATCH1. If first command is CK<n> it is skipped. Any other first command is also skipped!
25F29	(EvalNoCK:)	Run Stream: (ob →) EvalNoCK with the next object in the runstream as argument.
2A9E9	RunRPN:	Run Stream: (ob →) Evaluate the next object in the runstream with RPN mode on (i.e. system flag 95 clear). After the evaluation, the system flag is restored to its old value.

29.3.1 Type Checking

Addr.	Name	Description
36B7B	CKREAL	(% → %) (Z → %) Checks for real. If a ZINT, convert to real. Else SETTYPEERR.
184006	^CK1Z	($/#/hxs → Z) CHecks for an integer. Converts strings, bints or hxs's to zints. Errors for other object types.
185006	^CK2Z	(ob ob' → Z Z') Like ^CK1Z, but for two objects.
186006	^CK3Z	(ob ob' ob'' → Z Z' Z'') Like ^CK1Z, but for three objects.

Addr.	Name	Description
3D2B4	CKSYMBTYPE	(→) Checks for quoted name (name as symbolic).
2EF07	nmetasyms	(meta → meta) Checks for meta containing %, C%, unit, id, lam or symb.
03C64	TYPE	(ob → #prolog) Returns address of prolog of object.
3BC43	XEQTYPE	(ob → ob %type) System version of user word TYPE, but this keeps the object.
3511D	TYPEREAL?	(ob → flag)
35118	DUPTYPEREAL?	(ob → ob flag) aka: DTYPEREAL?
3512C	TYPECMP?	(ob → flag)
35127	DUPTYPECMP?	(ob → ob flag)
3510E	TYPECSTR?	(ob → flag)
35109	DUPTYPECSTR?	(ob → ob flag) aka: DTYPECSTR?
35136	DUPTYPEARRY?	(ob → ob flag) aka: DTYPEARRY?
3513B	TYPEARRY?	(ob → flag ???)
35292	TYPERARRY?	(ob → flag)
352AD	TYPECARRY?	(ob → flag)
35195	TYPELIST?	(ob → flag)
35190	DUPTYPELIST?	(ob → ob flag) aka: DTYPELIST?
3504B	TYPEIDNT?	(ob → flag)
35046	DUPTYPEIDNT?	(ob → ob flag)
350E1	TYPELAM?	(ob → flag)
350DC	DUPTYPELAM?	(ob → ob flag)
194006	^TYPEIDNTLAM?	(ob → flag) Tests if ob is ID or lam.
2F0D4	(ILnot?)	(ob → ob flag) Tests if ob is neither an ID nor a LAM.
35168	TYPESYMB?	(ob → flag)
35163	DUPTYPESYMB?	(ob → ob flag)
350FF	TYPEHSTR?	(ob → flag)
350FA	DUPTYPEHSTR?	(ob → ob flag)
35186	TYPEGROB?	(ob → flag)

Addr.	Name	Description
35181	DUPTYPEGROB?	(ob → ob flag)
351A4	TYPETAGGED?	(ob → flag)
3519F	DUPTYPETAG?	(ob → ob flag)
351B3	TYPEEXT?	(ob → flag) Is ob a unit object?
351AE	DUPTYPEEXT?	(ob → ob flag) Is ob a unit object?
3514A	TYPEROMP?	(ob → flag)
35145	DUPTYPEROMP?	(ob → ob flag)
350F0	TYPEBINT?	(ob → flag)
350EB	DUPTYPEBINT?	(ob → ob flag)
35159	TYPERRP?	(ob → flag)
35154	DUPTYPERRP?	(ob → ob flag)
3503C	TYPECHAR?	(ob → flag)
35037	DUPTYPECHAR?	(ob → ob flag)
35177	TYPECOL?	(ob → flag) Is on a secondary?
35172	DUPTYPECOL?	(ob → ob flag) Is ob a secondary? aka: DTYPECOL?
350D2	TYPEAPLET?	(ob → flag)
350CD	DUPTYPEAPLET?	(ob → ob flag)
35087	TYPEFLASHPTR?	(ob → flag)
35082	DUPTYPEFLASHPTR?	(ob → ob flag)
350C3	TYPEFONT?	(ob → flag)
350BE	DUPTYPEFONT?	(ob → ob flag)
350B4	TYPELNGCMP?	(ob → flag)
350AF	DUPTYPELNGCMP?	(ob → ob flag)
350A5	TYPELNGREAL?	(ob → flag)
350A0	DUPTYPELNGREAL?	(ob → ob flag)
35096	TYPEZINT?	(ob → flag)
35091	DUPTYPEZINT?	(ob → ob flag)
182006	^TYPEZ?	(ob → flag)
183006	^DUPTYPEZ?	(ob → ob flag)
114007	^TYPEGAUSSINT?	(ob → flag) Checks if ob is Gaussian integer.
115007	^DTYPEGAUSSINT?	(ob → ob flag) Checks if ob is Gaussian integer.

Addr.	Name	Description
116007	^DUPTYPEGAUSSINT?	(ob → ob flag) Checks if ob is Gaussian integer.
187006	^CK1Cext	(ob → flag) Checks if object is integer or Gaussian integer.
181006	^CKALG	(ob → ob) Checks that an object is real/cmplx/unit or idnt/lam/symbolic.
25E77	?OKINALG	(ob → ob flag) Is object allowed in algebraics?
171006	^DTYPFMAT?	(ob → ob flag) Tests if object is a symbolic matrix.
191006	^IDNTLAM?	(ob → ob flag) Tests if ob is `idnt` or lam.
192006	^FLOAT?	(ob → ob flag) Tests if ob is real or complex.
195006	^REAL?	(ob → ob flag) Tests if ob is real, zint or hxs.
196006	^TYPEREALZINT?	(ob → flag) Tests if ob is real, zint or hxs.
193006	^CKSYMREALCMP	(ob → ob) Does "Bad Argument Type" error if ob is not a real, complex or symbolics.

Chapter 30
Keyboard Control

There are several ways a System RPL program can get input from the user:

- From the stack;
- Waiting keystrokes from the keyboard;
- Using the internal `INPUT`;
- Using the internal `INFORM`;
- Setting up a Parameterized Outer Loop;
- And other methods.

You have already seen how to get input directly from the stack. Using `InputLine`, `ParOuterLoop` and input forms will be seen on the following chapters. In this chapter, you will learn how to read keystrokes from the keyboard.

30.1 Key Locations

In User RPL, key representations have the form `%rc.p`. In System RPL, they are represented by two binary integers. The first, often called, `#KeyCode`, goes from one (F1 key) to 51 (ENTER key), and represents each key, in order, from left to right and top to bottom. The up arrow is code 10, being considered the fourth key of the second row. The left, down and right arrows have codes 14, 15, 16, respectively, being considered as part of the third row.

The second number, `#Plane`, represents the modifier states, according to the table below:

#Plane	Modifiers	#Plane	Modifiers
1	None	4	Alpha
2	Left-shift	5	Alpha, left-shift
3	Right-shift	6	Alpha, right-shift

You can convert from one representation to another using:

`Ck&DecKeyLoc` (`%rc.p` → `#KeyCode #Plane`)

`CodePl>%rc.p` (`#KeyCode #Plane` → `%rc.p`)

Sometimes, the shift keys are not being treated as modifiers for other keys, but as keys in their own right. Then they have the key codes 40h (left-shift), C0h (right-shift), and 80h (alpha).

On the HP48, only the six key planes listed above existed. The HP49 introduced five more planes, the shift-hold keys. These are shifted keypresses, where the shift key is being held down while the key is pressed. In User RPL, these keys are denoted by adding 0.01 to the %rc.p representation. For example, the keycode 11.21 means holding down left-shift while pressing the `F1` key.

In System RPL, shift-hold keys can be encoded in two ways. The first form (which we will call encoding A) leaves the keycode `#kc` unchanged, and uses new planes `#8...#C`. The second form (encoding B) uses planes in the range `#1...#6` and adds the keycode of the shift key to the keycode `#kc`.

The following table lists the different encodings for all possible ways to press the `F1` key on the HP49G.

		User RPL	A		B	
Plane	**Shift Keys**	**%rc.pl**	**#kc**	**#pl**	**#kc**	**#pl**
1	Unshifted	11.1	1h	1h	1h	1h
2	Left-shift	11.2	1h	2h	1h	2h
3	Right-shift	11.3	1h	3h	1h	3h
4	Alpha	11.4	1h	4h	1h	4h
5	Alpha, left-shift	11.5	1h	5h	1h	5h
6	Alpha, right-shift	11.6	1h	6h	1h	6h
7	Unused					
8	Left-shift-hold	11.21	1h	8h	41h	2h
9	Right-shift-hold	11.31	1h	9h	C1h	3h
10	Alpha-hold	11.41	1h	Ah	81h	4h
11	Alpha, left-shift-hold	11.51	1h	Bh	41h	5h
12	Alpha, right-shift-hold	11.61	1h	Ch	C1h	6h

Most, but not all, System RPL entries dealing with keys can handle shift-hold key presses. The reference section has information about this issue for each relevant entry. The System RPL entries which expect `#kc` and `#pl` as arguments (like `CodePl>%rc` or `Key>StdKeyOb`) accept both forms of encoding (A and B). Entries which return `#kc` or `#kc` and `#pl` (like `Ck&DecKeyLoc` and `GETTOUCH`) all use encoding B. Encoding A seems to be more convenient for dispatching. In order to convert encoding B into encoding A, you can use

```
:: SWAP 64 #/ ROTSWAP #0=?SKIP #6+ ;
```

30.2 Waiting for a Key

A convenient entry used to wait for a key is `WaitForKey`. This command puts the HP49 in a low-power state and waits until a key is pressed. It then returns the key code to level two and the plane to level one. There are other words, listed below, which are used in other circunstances.

Unfortunately, `WaitForKey` does not deal with the shift-hold keys. We therefore list below a short program which behaves just like `WaitForKey` but returns the extended keycode (encoding B). The program contains a code object accessing the key buffer, which we list without explanation. You will find this program in the `keyboard` directory in the examples directory.

```
::
  WaitForKey        (normal WaitForKey)
  CODE              (get extended keycode)
    GOSBVL =SAVPTR
    D0=    047DF
    A=DAT0 A
    D0=A
    A=0    A
    A=DAT0 1
    A=A-1  P
    A=A+A  A
    CD0EX
    C=C+A  A
    CD0EX
    D0=D0+ 2
    A=DAT0 B
    GOVLNG =PUSH#ALOOP
```

```
    ENDCODE
    ROTDROPSWAP  (replace keycode with extended value)
;
```

If you would like the program to return encoding A instead of encoding B, just replace `ROTDROPSWAP` with:

```
BINT63 #>case #6+
```

30.3 Reference

30.3.1 Converting Keycodes

Addr.	Name	Description
25EA7	Ck&DecKeyLoc	(%rc.p → #kc #p) Converts from user key representation format to system. Does handle shift-hold keys.
25EA9	CodePl>%rc.p	(#kc #p → %rc.p) Converts from system key representation format to user. Does handle shift-hold keys.
25EDC	H/W>KeyCode	(# → #') Converts the keycode offset for shift keys to the keycode of the shift key, i.e. 80h->32d, 40h->37d, C0h->42d
25EEA	ModifierKey?	(#kc #pl → flag) Is the key any of the three modifiers right-shift, left-shift, or alpha?

30.3.2 Waiting for Keys

Addr.	Name	Description
261CA	FLUSHKEYS	(→) Flushes the key buffer. aka: FLUSH

Addr.	Name	Description
04708	CHECKKEY	(→ #kc T) (→ F) Returns next key in the key buffer (if there is one), but does not pop it. Does handle shift-hold keys.
04714	GETTOUCH	(→ #kc T) (→ F) Pops next key from key buffer (if there is one). Does handle shift-hold keys.
25ED6	GETKEY	(→ #kc flag) Get a single keypress from the keybuffer, waits if necessary. The key is returned along with TRUE. If an exception happens, returns FALSE. The exception is not handled. Does handle shift-hold keys.
25ED7	GETKEY*	(→ #kc T) (→ F F) (→ {Alrmlist} T F) Get a single keypress from the keybuffer, waits if necessary. The key is returned along with TRUE. If an exception happens (error or alarm), the exceptions is handled and the entry returns FALSE. Does handle shift-hold keys.
25ED9	GetKeyOb	(→ ob) Wait for a single key and return the object associated with this key. Does handle shift-hold keys.
25EC5	DoKeyOb	(ob →) Execute ob as if it had been assigned to a key and the key had been pressed.
047C7	REPKEY?	(#kc → flag) Returns TRUE if the key is being pressed.
25EE3	KEYINBUFFER?	(→ flag) Returns TRUE if there is at least a key in the key buffer.
25F0B	WaitForKey	(→ #kc #flag) Returns next full key press. Does *not* handle shift-hold keys.

Addr.	Name	Description
2F268	Wait/GetKey	(% → ?) Internal WAIT command. Does *not* handle shift-hold keys.

30.3.3 The ATTN Flag

Addr.	Name	Description
25FAE	ATTN?	(→ flag) Returns TRUE if CANCEL has been pressed.
25E70	?ATTNQUIT	(→) If CANCEL has been pressed, ABORTs program. aka: ?ATTN_QUIT
25E9D	CK0ATTNABORT	(→) Executed by the UserRPL program delimiters x<< and x>> and by xUNTIL. Mainly just ?ATTNQUIT.
25EED	NoAttn?Semi	(→) If CANCEL has been not pressed, drops the rest of the stream.
05040	ATTNFLG@	(→ #) Recalls CANCEL key counter.
05068	ATTNFLGCLR	(→) Clears CANCEL key counter. Does not affect the key buffer.

30.3.4 Bad Keys

Addr.	Name	Description
25EBF	DoBadKey	(→) Beeps.
25ECD	DropBadKey	(ob →) Beeps.
25E6E	2DropBadKey	(ob ob' →) Beeps.

30.3.5 User Keys

If no keys are assigned, the internal key assignments list is an empty list. If there is one or more assignments, the list contains 51 sublists, each one representing one key. Each sublist is either empty, if that key has no assignments; or contains twelve elements: each representing the assignment of one plane. The planes are given in the table in section 30.1. For planes with no assignment, an empty list must be given. The seventh list is always empty.

Addr.	Name	Description
25F09	UserKeys?	(→ flag) Does BINT62 TestSysFlag.
25967	GetUserKeys	(→ {}) Returns user keys list (internal format).
2F3B3	(AsnKey)	(ob #kc #p →) Assigns an object to a key, specified in system format.
25621	(NonUsrKeyOK?)	(→ flag) Returns TRUE if the keys not defined do their normal actions.
25617	(SetNUsrKeyOK)	(→) Keys not defined do their normal actions.
2561C	(ClrNUsrKeyOK)	(→) Keys not defined just beep when pressed.
25EE5	Key>StdKeyOb	(#kc #pl → ob) Recalls the standard assignment of the key. This is the assignment which is active when USER mode is of.
25EE6	Key>U/SKeyOb	(#kc #pl → ob) If user mode is on, recalls the user object assigned to a key. If user mode is off, recalls the standard assignment instead.
255006	^KEYEVAL	(% → ?) Keystroke evaluation. If % is negative, the standard key is always evaluated.

Chapter 31
Using InputLine

The command `InputLine` is the system equivalent to the user command `INPUT`. Its use is similar, and does a similar thing:

- Displays a prompt in the top of the screen;
- Starts the keyboard entry modes;
- Initializes the edit line;
- Accepts input until ENTER is pressed;
- Parses, evaluates, or just returns the user input;
- Returns TRUE if the environment was exited by ENTER or FALSE if it was aborted by ON/CANCEL.

The stack must contain the following parameters:

Name	**Description**
$Prompt	The prompt to be displayed during input.
$EditLine	The initial edit line.
CursorPos	The initial cursor position. You can either specify the character position, in absolute terms, or as a two-element list with the row and column. In both cases, a `#0` represents the end of edit line, row or column. All numbers should be specified as bints, naturally.
#Ins/Rep	The initial insert/replace mode of the cursor: • `#0` current mode • `#1` insert mode • `#2` replace mode

Name	Description
#Entry	The initial entry mode: • `#0` current entry mode plus program entry mode • `#1` only program entry mode • `#2` program and algebraic entry modes
#Alphalock	The initial alpha mode: • `#0` current mode • `#1` alpha enabled • `#2` alpha disabled
ILMenu	The initial menu, in the format specified below. Normally, specified as `FALSE`, which means that the menu should not be changed.
#ILMenu	The initial menu row number (normally `BINT1`, to show the first page).
AttnAbort?	A flag: • `TRUE` CANCEL aborts the input • `FALSE` CANCEL just clears the edit line
#Parse	How to process the edit line: • `#0` return edit line as a string (unevaluated) • `#1` return edit line as a string and a parsed object • `#2` parse and evaluate edit line

If AttnAbort? is `TRUE` and the user presses CANCEL while `InputLine` is active, the edition is aborted. If it is `FALSE`, CANCEL just clears the edit line. If it was already empty, then it aborts `InputLine`, returning `FALSE`.

Depending on the value of #Parse, different values are returned, acording to the table:

#Parse	Stack	Description
`#0`	`$Editline TRUE`	Edit line only (unevaluated)
`#1`	`$Editline obs TRUE`	Edit line and parsed object(s)
`#2`	`ob1 ... obn TRUE`	Resulting object(s)
	`FALSE`	CANCEL pressed to abort

31.1 Menu Key Assignments

Any application can specify an initial menu via the ILMenu parameter. This menu will be displayed when the `InputLine` starts. All menu keys can have assignments to the unshifted, left-shifted and right-shifted planes. When the loop exits, the previous menu is restored intact.

The ILMenu parameter is a list (or, in rare cases, a program returning a list), in the format described in section 37.1. You can also supply just `FALSE` as this parameter, if you do not want the current menu to be changed.

Note that the actions must start with the word `TakeOver` to flag that they should be run with the command line active.

31.2 An Example

Here is an example of `InputLine`, which prompts for your name, and if the edition was not aborted, displays it.

```
::
  $ "Your name:"      (prompt)
  NULL$               (initial edit line)
  #ZERO#ONE           (cursor at end, insert mode)
  ONEONE              (prog mode, alpha enabled)
  NULL{}              (no menu)
  ONE                 (menu row)
  FALSE               (CANCEL clears)
  ZERO                (returns string)
  InputLine
  NOT?SEMI            (exit if FALSE)
  $ "Your name is "
  SWAP&$              (concatenate string & name)
  CLEARLCD            (clear display)
  DISPROW1            (display string on 1st line)
  SetDAsTemp          (freeze display)
;
```

31.3 Reference

Addr.	Name	Description
2EF5F	InputLine	(args → $ T) (args → $ ob1..obn T) (args → ob1..obn T) (args → F) args = $pr $line #pos #I/R #I/A #alph menu #row attn #parse
2F154	(input$)	($1 $2 → $3) This is what the User command INPUT does if level 1 is a string.
2F155	(input{})	($1 {} → $3) This is what the User command INPUT does if level 1 is a list.

Chapter 32
The Parameterized Outer Loop

The Parameterized Outer Loop is a System RPL structure that allows you to create a complete application, which receives keystrokes and does different actions based on the key that was pressed. This is repeated as many times as necessary, until an exit condition happens. Most of the time, there is a key that stops the loop, like CANCEL or DROP. Generally, it is used with programs that work with the display. Complex uses of the POL include input forms (Chapter 35) and the browser (Chapters 33 and 34). Note that POLs are a very general construct and for that reason they require elaborate arguments. Simple applications can sometimes be implemented more easily and compactly with a loop around `WaitForKey` (section 30.2) and direct display handling.

To set up a parameterized outer loop, nine parameters are necessary:

Parameter name	Description
AppDisplay	This object is evaluated before each key evaluation. It should handle display updating not handled by the keys themselves, and should also perform special handling of errors.
AppKeys	The hard key assignments, in the format described below.
NonAppKeyOK?	A flag: if `TRUE`, then the hard keys not assigned perform their normal actions. Otherwise, they just beep.
DoStdKeys?	A flag: if `TRUE`, then standard key definitions are used for non-application keys instead of default key processing.
AppMenu	Either the menu specification, in the format described in section 37.1, or `FALSE` to leave the current menu unchanged.
#AppMenuPage	The initial menu page. Normally `BINT1` to show the first page.

Parameter name	Description
SuspendOK?	A flag: if `TRUE`, any user command that would create a suspended environment and restart the system outer loop will instead generate an error.
ExitCond	This object is evaluated before each display update and key evaluation. If the result is `TRUE`, the loop is exited.
AppError	The error-handling object to be evaluated in an error occurs during key evaluation.

After setting up the arguments, call `ParOuterLoop`. This word does not generate any results itself, but any of the key assignments can return results to the stack or any other form desired.

32.1 Parameterized Outer Loop Words

The parameterized outer loop is formed by calls (with proper error handling) to the following words. None of them return anything, and the only one that takes arguments is `POLSetUI`: the same nine required by `ParOuterLoop`.

Word	Action
`POLSaveUI`	Saves the current user interface in a temporary environment.
`POLSetUI`	Sets the current user interface, according to the parameters given.
`POLKeyUI`	Displays, reads and evaluates keys. Handles errors, and exits according to the user interface specified by `POLSetUI`.
`POLRestoreUI`	Restores the user interface saved by `POLSaveUI` and abandons the temporary environment.
`POLResUI&Err`	Restores the user interface and errors. This is used when there is an error not handled within the parameterized outer loop.

The word `ParOuterLoop` decompiles to:

```
::
  POLSaveUI       (save current user interface)
  ERRSET ::       (start error trap)
    POLSetUI      (set new user interface)
```

```
      POLKeyUI      (handle keypresses)
    ;
    ERRTRAP
      POLResUI&Err  (if an error happened, restore)
                    (the saved interface and error)
    POLRestoreUI    (restore saved user interface)
  ;
```

If you use the words above instead of ParOuterLoop, you must provide the same level of error protection as the code above.

One note: the parameterized outer loop creates a temporary environment when it saves its current user interface, and it abandons it when it restores a saved user interface. This means that you cannot use words that operate on the topmost temporary environment, like 1GETLAM within the loop, unless the variable was created after calling POLSaveUI, and it is abandoned before calling POLRestoreUI. For temporary environments created before calling POLSaveUI, named temporary variables should be used.

32.2 The Display

In the parameterized outer loop, the user is responsible for setting up the display and updating it; there is no default display.

The display can be updated in two ways: with the parameter “AppDisplay” or with key assignments. For example, when the user presses a key to move the cursor, the key assignment can either pass information to “AppDisplay” (often implicitly), so that it handles the screen updating, or the key assignment object can handle the display itself. Which method is more efficient depends on the situation. In our example below, AppKeys just sets the position of the grob in lams, and AppDisplay draws the grob.

32.3 Error Handling

If an error occurs during the key processing, AppError is executed. This object is responsible for processing any errors generated while the parameterized outer loop is running. AppError should determine the specific error and act accordingly. Or you can just specify ERRJMP as AppError, which means your application does not handle any errors.

32.4 Hard Key Assignments

In the parameterized outer loop, any key in any of the six basic planes (see section 30.1) can be assigned a new function. The parameter AppKeys specifies which keys to assign and their actions.

If a key is not assigned by the application, and the NonAppKeyOK? parameter is TRUE, the standard key definition is executed if the DoStdKeys? parameter is TRUE, or, if available, the USER key assignment, if it is FALSE. If NonAppKeyOK? is FALSE, a warning beep is produced, and nothing else is done.

Most of the time, NonAppKeysOK? should be set to FALSE.

The AppKeys parameter is a secondary, which must take as argument the keycode and plane, and return either the desired key definition and TRUE, or FALSE if the application does not handle it. Specifically, the stack diagram is as follows:

(#KeyCode #Plane → KeyDef TRUE)
(#KeyCode #Plane → FALSE)

The suggested form for the key assignments is:

```
BINT1 #=casedrop :: (process unshifted plane) ;
BINT2 #=casedrop :: (process left-shifted plane) ;
...
2DROPFALSE
```

And each plane handler normally has the form

```
BINT7 ?CaseKeyDef :: TakeOver <process APPS key> ;
BINT9 ?CaseKeyDef :: TakeOver <process TOOL key> ;
...
DROPFALSE
```

The word ?CaseKeyDef is very handy in this case, because it is equivalent to #=casedrop :: ' <keydef> TRUE ;. Using this word, the code becomes shorter, and the definitions become more legible. ?CaseKeyDef is used in the form:

... #KeyCode #TestKeyCode ?CaseKeyDef <keydef> ...

If #TestKeyCode equals #KeyCode, ?CaseKeyDef drops both of them, pushes <KeyDef> and TRUE to the stack, and exits the secondary. Otherwise, it drops only #TestKeyCode, skips <KeyDef> and continues.

If you want to handle shift-hold keys, you can do so. The extended keycode (encoding B, see section 30.1) is provided to the AppKeys program on stack levels 5 and 6 All you need to do is to start AppKeys with the snippet

```
4DROP 2DUP 2DUP
```

and than dispatch normally.

32.5 Menu Key Assignments

You can specify a menu to be displayed when the parameterized outer loop starts. The format of the AppMenu parameter is essentially the same of the ILMenu parameter of `InputLine`, described in section 37.1.

The difference is that `TakeOver` is not necessary in this case, since the input line is not active.

Also, since hard key assignments have priority over menu key assignments, you should put this code in the AppKeys parameter, in each plane definition:

```
DUP#<7 casedrpfls
```

This will push `FALSE` when a key whose code is less than seven (that is, one of the softkeys) is pressed. The `FALSE` will force the standard assignment to be run, and this assignment runs the action defined by the AppMenu parameter.

For that to work, the NonAppKeysOK? parameter must be `TRUE`, so that the menu keys work normally, that is, doing the actions specified by the AppMenu parameter.

32.6 Preventing Suspended Environments

Your application may require the evaluation of arbitrary commands and user arguments, but it might not want the current environment to be suspended by `HALT` or `PROMPT` commands. The parameter SuspendOK?, when `FALSE`, will cancel these and any other commands that would suspend the environment and generate a “HALT Not Allowed” error, which AppError can handle. If the parameter is `TRUE`, the application must be prepared to handle the consequences. “The dangers here are many and severe”, as it is written in `RPLMAN.DOC`.

Almost all applications should set `FALSE` as the SuspendOK? parameter.

32.7 The Exit Condition

The parameter ExitCond is an object that is evaluated before each key evaluation. If it evaluates to `TRUE`, the loop is exited, otherwise it continues. You could define, for example, ExitCond as `' LAM exit`. When the “quit” key is pressed, you just have to use `TRUE ' LAM exit STO` and the loop will be exited. Naturally you must create the lam and initialize it with `FALSE` before.

32.8 An Example

The following program is an example of an application that uses a parameterized outer loop to create an environment where the user may move a little graphic over the screen. You can use the arrow keys to move, or the menu keys. In both cases, if you press left-shift before, the graphic moves ten steps instead of one. There is code to assure that the graphic does not go off the screen boundaries.

Figure 32.1 below displays this program running.

Figure 32.1: The POL example

```
::
* Defines names for used keys. Makes things easier and
* more readable
  DEFINE kpNoShift    BINT1
  DEFINE kpLeftShift  BINT2
  DEFINE kcUpArrow    BINT10
  DEFINE kcLeftArrow  BINT14
  DEFINE kcDownArrow  BINT15
```

```
  DEFINE kcRightArrow BINT16
  DEFINE kcLeftShift  BINT37
  DEFINE kcOn         BINT47

* Requires no arguments
  CK0NOLASTWD

* Prepare display
  RECLAIMDISP    (clear and resize display)
  ClrDA1IsStat   (temporarily disable clock)

* Smiling face grob. The below must be in one line only.
  GROB 7C 310003100008F000060300810C004000104000102000202
4012010004010004010004010004011044021042026032048F0104000
10810C0006030008F000
  FIFTYSIX      (initial x coordinate for box)
  EIGHTEEN      (initial y coordinate for box)
  FALSE         (initial exit condition)
  {
    LAM MrSmile
    LAM x
    LAM y
    LAM exit?
  } BIND        (binds local variables)

* The following composite is the display update object.
* It clears the screen and draws the smiling face grob.
’ ::
    CLEARVDISP       (clear display)
    LAM MrSmile      (recall smiling face grob)
    HARDBUFF         (recall current display)
    LAM x LAM y      (smile coordinates)
    GROB!            (REPL)
    DispMenu.1       (display menu)
  ;

* The following composite is the key action handler.
  ’ ::
    kpNoShift #=casedrop ::
      DUP#<7 casedrpfls  (enable softkeys)
      kcUpArrow ?CaseKeyDef
        ::
          LAM y DUP
          BINT1 #<ITE
            :: DROP ERRBEEP ;
```

```
          :: #1- ' LAM y STO ;
      ;
    kcDownArrow ?CaseKeyDef
      ::
        LAM y DUP
        BINT36 #>ITE
          :: DROP ERRBEEP ;
         :: #1+ ' LAM y STO ;
      ;
    kcLeftArrow ?CaseKeyDef
      ::
        LAM x DUP
        BINT1 #<ITE
          :: DROP ERRBEEP ;
         :: #1- ' LAM x STO ;
      ;
    kcRightArrow ?CaseKeyDef
      ::
        LAM x DUP
        BINT111 #>ITE
          :: DROP ERRBEEP ;
         :: #1+ ' LAM x STO ;
      ;
    kcOn ?CaseKeyDef
      :: TRUE ' LAM exit? STO ;
    kcLeftShift #=casedrpfls
    DROP 'DoBadKeyT
  ;
  kpLeftShift #=casedrop ::
    DUP#<7 casedrpfls  (enable softkeys)
    kcUpArrow ?CaseKeyDef
      ::
        LAM y DUP
        BINT10 #<ITE
          :: DROPZERO ERRBEEP ;
         :: BINT10 #- ;
        ' LAM y STO
      ;
    kcDownArrow ?CaseKeyDef
      ::
        LAM y DUP
        BINT27 #>ITE
          :: DROP BINT27 ERRBEEP ;
         #10+
        ' LAM y STO
```

```
        ;
      kcLeftArrow ?CaseKeyDef
        ::
          LAM x DUP
          BINT10 #<ITE
            :: DROPZERO ERRBEEP ;
           :: BINT10 #- ;
          ’ LAM x STO
        ;
      kcRightArrow ?CaseKeyDef
        ::
          LAM x DUP
          BINT102 #>ITE
          :: DROP BINT112 ERRBEEP ;
           #10+
          ’ LAM x STO
        ;
      kcLeftShift #=casedrpfls
      DROP ’DoBadKeyT
    ;
    2DROP ’DoBadKeyT
  ;

* Key definitions
  TrueTrue

* Menu specification
  { { "Up" {
               ::
                 LAM y DUP
                 BINT1 #<ITE
                   :: DROP ERRBEEP ;
                  :: #1- ’ LAM y STO ;
               ;
               ::
                 LAM y DUP
                 BINT10 #<ITE
                   :: DROPZERO ERRBEEP ;
                  :: BINT10 #- ;
                 ’ LAM y STO
               ;
           }
    }
    { "Down" { ::
                   LAM y DUP
```

```
                  BINT36 #>ITE
                    :: DROP ERRBEEP ;
                   :: #1+ ’ LAM y STO ;
                ;
                ::
                  LAM y DUP
                  BINT37 #>ITE
                    :: DROP BINT37 ERRBEEP ;
                   #10+
                 ’ LAM y STO
                ;
              }
     }
     { "Left" { ::
                  LAM x DUP
                  BINT1 #<ITE
                    :: DROP ERRBEEP ;
                   :: #1- ’ LAM x STO ;
                ;
                ::
                  LAM x DUP
                  BINT10 #<ITE
                    :: DROPZERO ERRBEEP ;
                   :: BINT10 #- ;
                  ’ LAM x STO
                ;
              }
     }
     { "Right" { ::
                   LAM x DUP
                   BINT111 #>ITE
                     :: DROP ERRBEEP ;
                    :: #1+ ’ LAM x STO ;
                 ;
                 ::
                   LAM x DUP
                   BINT102 #>ITE
                     :: DROP BINT112 ERRBEEP ;
                    #10+
                   ’ LAM x STO
                 ;
               }
     }
     NullMenuKey
     { "Quit" :: TRUE ’ LAM exit? STO ; }
```

```
    }
    ONEFALSE     (first menu row, no suspended envs)
    ' LAM exit?  (exit condition)
    'ERRJMP      (error handler)
    ParOuterLoop (run the par outer loop)
    RECLAIMDISP  (resize and clear display)
    ClrDAsOK     (redraw display)
  ;
```

32.9 Reference

Addr.	Name	Description
2B475	ParOuterLoop	(Disp Keys NonAppKeys? DoStdKeys? menu #row suspendOK? ExitCond AppErr →)
2B4AC	POLSaveUI	(Disp Keys NonAppKeys? DoStdKeys? menu #row suspendOK? ExitCond AppErr →) Saves current UI to LAMSavedUI.
2B542	POLSetUI	<see>ParOuterLoop Sets new UI, same arguments as to ParOuterLoop.
2B628	POLKeyUI	(→) Displays, reads and evaluates keys according to set UI.
2B6CD	POLRestoreUI	(→) Restores saved UI from LAMSavedUI.
2B6B4	POLResUI&Err	(→) Restores saved UI and executes ERRJMP.
29F25	AppDisplay!	(ob →)
29F35	AppDisplay@	(→)
29F55	AppKeys!	(ob →)
29F75	AppKeys0	???
2A055	AppExitCond!	(ob →)
2A065	AppExitCond@	(→ ob)
2A145	AppError!	(ob →)
2A158	AppError@	(→ ob)

Addr.	Name	Description
25690	AppMode?	(→ flag) Is currently a POL active?
25695	SetAppMode	(→)
2569A	ClrAppMode	(→)
2564D	SetNAppKeyOK	(→)
2565A	DoStdKeys?	(→ flag)
2565F	SetDoStdKeys	(→)
25F04	SuspendOK?	(→ flag) Does the current user interface allow suspension?
27E72	nohalt	(→ ob) :: LAM 'nohalt ;
25671	SetAppSuspOK	(→)
25676	ClrAppSuspOK	(→)

Chapter 33
Using the HP49 Browser

The browser is the engine behind the selection boxes created by the User RPL command `CHOOSE`. However, it can do much more than what that command does.

There are two browser engines in the HP49G calculator: the old one, which was present since the HP48G series, and a new one, only present in the HP49G model. This chapter will describe the new engine, which is easier to use. It has some features the old one does not have, but the old one also has some important features that this one does not, such as full screen mode and selecting multiple items. The next chapter will describe the old engine.

There are several flashpointers which can be use to access the browser engine. These flashpointers are not officially supported, but are very likely stable.

The main difference to User RPL `CHOOSE` command is that you can specify a message handler, which can be used to provide a custom the menu, to handle key presses and some other things.

The main entry is `FPTR 2 72` (`^Choose3`). It has the following stack diagram:

(`meta $title #initial ::message` → `ob TRUE`) or
(`meta $title #initial ::message` → `FALSE`)
depending on whether the user selects something or cancels.

As an alternative, you can replalce `FPTR 2 72` with `FPTR 2 74`. The differences are that the entry does not save a copy of the original meta on the virtual stack and that instead of the selected object, the index is returned. The indices start at zero, not one.

33.1 The Choose Items meta

`meta` is a meta object (see Chapter 12) that contains the items which should be shown in the selection box. All object types are allowed, and they will be decompiled for display.

33.2 The Title String

`$title` is the title. It will be shown in a small box on top of the choose box. No title will be shown if this is the empty string. This can be useful when the the contents of the choose box do not need a further explanation. Omitting the title makes space for an additional item line.

33.3 The Initially Selected Item

When the choose engine starts, an item is already highlighted. Usually this is the first item, but you can select another one with the `#initial` parameter. The numbering starts with zero, not one.

33.4 The Message Handler

`::message` is a program, the message handler. A message handler is a general way to pass a variable number of optional parameters to an application. The application will call the message handler program with different "messages" (normally a bint) in stack level 1, and maybe additional arguments in other stack levels. The handler can decide to handle this message. If it does handle it, it should do its work and return `TRUE`. If it decides to ignore the message, it should just drop the bint and return `FALSE`. The empty message handler therefore is the command `DROPFALSE` (which you can conveniently push in the stack with `'DROPFALSE`). When you use the user command `xCHOOSE`, it just supplies `DROPFALSE` and hands over to the more general engine.

The message handler can handle the following messages:

Message	**Message name and meaning**
`BINT1`	MsgDispBox This message has to do with the display of the choose box. It is currently not well understood. The stack diagram of the message handler for this message seems to be (#1 → `::prog TRUE`) (#1 → `FALSE`)
`BINT2`	MsgDispTitle This should display the title. If not handled, the title is drawn using the supplied `$title` argument. (#2 → `TRUE`) (#2 → `FALSE`)
`BINT3`	MsgEndInit This message is executed after the initialization of the choose box, but before control is handed over to the POL. (#3 → `TRUE`) (#3 → `FALSE`)
`BINT4`	MsgKeyPress This is a key handler, similar to the ones used by a POL. When the user presses a key, the message handler is called with the keycode and plane (see section 30.1), and the message `BINT4` on the stack. It should return the key definition (an address or a secondary), `TRUE` and `TRUE` again. If the key is not handled, `FALSE` must be returned. Here is the stack diagram for the message handler regarding this message: (`#kc #pl #4` → `KeyDef TRUE TRUE`) — yes, `TRUE` twice! (`#kc #pl #4` → `FALSE`)
`BINT5`	MsgMenu This must return the menu which is shown to the user during the selection. The return value for this message is evaluated to get the menu. The menu is not automatically updated when you move the selection, but message #6 can be used to enforce an update. If the menu has more than one page, you must handle the NXT and PREV keys in the keyhandler — they are not handled by default. (#5 → `{ menu list } TRUE`) (#5 → `::prog_returning_list TRUE`) (#5 → `FALSE`)

Message	Message name and meaning
`BINT6`	MsgEndEndDisp This message is called after the redisplay of the choose box finishes (because you changed the selected item). You can use this to force an update of the menu display by setting 24LAM to `FALSE`. See the example below. (`#6` → `TRUE`) (`#6` → `FALSE`)

33.5 The Browser and Lams

The browser POL uses 24 unnamed local variables, so maybe you should not rely on unnamed locals yourself. Better use named locals for this purpose. A few important unnamed LAMs used by the browser engine are:

LAM	Contents
`1LAM`	Quit. Set this to `TRUE` if you want the POL to exit.
`2LAM`	DispOffset. Index of selected item with respect to DispTop.
`3LAM`	DispTop. Index of the first choose item currently visible on the screen.
`17LAM`	The message handler.
`14LAM`	The redisplay program.
`24LAM`	DisplayMenu. Set this to `FALSE` in order to enforce a redisplay of the menu.

33.6 Accessing the Selected Item

To use the browser for more than just selecting an item, you must write programs which will be accessible with the key handler or with the menu. On of the most important tasks in these programs is to find out what the current item is. The choose box engine keeps two copies of the choose list on the Virtual Stack, and you can use these to get the current item. On level one of the Virtual Stack, the list is inverted, and the items which have already been shown in the CHOOSE box are converted to strings (sensitive to flag -85). On level three of the Virtual Stack there is a copy of the original list. The index of the current item is available with this code snippet:

```
:: 2GETLAM 3GETLAM #+ ;
```

Indexes start at 0.

To access the current item use one of these methods:

1. Get the decompiled string. This is very fast and only 8 bytes.

   ```
   :: 2GETLAM 3GETLAM #+ GetElemBotVStack ;
   ```

2. Get the original item. There is no supported way to get to the third level of the Virtual Stack directly, so you have to dig it out and restore the stack afterwards. Here is a way to do it (35 bytes):

   ```
   ::
     GetVStackProtectWord PopMetaVStack
     GetVStackProtectWord PopMetaVStack
     2GETLAM 3GETLAM #+
     GelElemTopVStack
     1LAMBIND
     PushMetaVStack&Drop SetVStackProtectWord
     PushMetaVStack&Drop SetVStackProtectWord
     1GETABND
   ;
   ```

   This looks complicated, but it is also quite fast and actually used in the ROM for the Help key of the catalog.

3. If you find 2 too long, you can keep a copy of your original list, for example in a named LAM “mylist”. If you did that before calling the browser flashpointer, you get the current item with

   ```
   :: LAM mylist 2GETLAM 3GETLAM #+ #1+ NTHCOMPDROP ;
   ```

33.7 Saving and Restoring the Screen

If you want to use a menu key or another key to do an excursion from the choose box which uses the display, you must save and restore the current screen around it. This is because the browser POL only updates as little as possible on the display, so when you return from your excursion, the display will look bad and not recover. There are two simple flashpointers which can be used to save and restore the display:

`FPTR 2 88` Save the current isplay
`FPTR 2 89` Restore the saved display

Note that these commands use a specific storage place, so they cannot be used by stacked choose boxes (a choose box creating another choose box which needs to save its screen for an excursion). In such cases, you need to save and restore copies of `HARDBUFF` and `HARDBUFF2`.

33.8 An Example

Below follows an example for the application of the browser engine. This program displays the numbers 1-100 for multiple selection and returns a list of all selected values. Pressing the Squareroot key displays the square root of the current number in a message box. In the menu, pressing F1 adds the decompiled version of the currently selected number (a string!) to the return list. Pressing F2 will show some help text about the choose box. There is another menu button F3 which does not do anything, but which shows if the selected number is even or odd. Since this display changes, we need message six to force a menu update. F5 and F6 are the usual CANCL and OK actions.

Figure 33.1 shows this program while running.

Figure 33.1: The ’49 browser example

```
::
  101 ONE_DO
    INDEX@ UNCOERCE
  LOOP
  100 P{}N                      (Make list with 1-100)
  DUP
```

```
  NULL{}                          (Empty list to collect)
  { LAM mylist LAM res } BIND     (Save a copy of the list)
  INNERCOMP                       (Explode for FPTR 2 72)
  "REALS"                         (Title)
  0                               (Initial position)
  '
  ::                              (The key handler)
    4 OVER#=case
    ::
      DROP DUP#1= 3PICK 23 #=
      ANDcase                     (SQRT key pressed)
      ::
        2DROP                     (DROP the keycodes)
        '
        ::
          LAM mylist
          2GETLAM 3GETLAM
          #+ #1+
          NTHCOMPDROP             (Get current value)
          %SQRT DO>STR            (Compute SQRT)
          FlashWarning            (Display)
        ;
        TrueTrue                  (Yes, we handle this key)
      ;
      FALSE                       (Other keys not handled)
    ;
    5 OVER#=case                  (Provide a menu)
    ::
      DROP
      ' ::
        NoExitAction              (Do not save as LastMenu)
        { { "->{}"                ("Add to list" menu key)
            :: TakeOver
               LAM res            (Get current list)
               2GETLAM            (Get element as string)
               3GETLAM #+
               GetElemBotVStack
               >TCOMP             (Add to list)
               ' LAM res STO      (STO current list)
            ; }
          { "?"                   ("Help" menu entry)
            :: TakeOver
               FPTR 2 88          (Save current screen)
               DOCLLCD            (Clear screen)
               ZEROZERO           (Next is the help text)
```

```
"->{}   ADD
?      HELP
SQRT   DISP ROOT"
               $>GROBCR
               XYGROBDISP       (Display help text)
               WaitForKey       (Wait for any key)
               2DROP
               FPTR 2 89        (Restore the screen)
            ;
          }
          { ::                  (Button to show)
               TakeOver           ("even" or "odd")
               LAM mylist       (The list)
               2GETLAM
               3GETLAM #+ #1+   (Get current element)
               NTHCOMPDROP
               DUP              (Test if even)
               %2 %/
               %FLOOR
               %2 %* %= ITE
               "even" "odd"     (Return correct label)
            ;
            NOP                 (No action when pressed)
          }
          NullMenuKey           (4th key is empty)
          { "CANCL"             (Default CANCL action)
            FPTR 2 77 }
          { "OK"                (Default OK action)
            FPTR 2 76 }
        }
      ;
      TRUE                      (Yes, we provide a menu)
    ;
    6 OVER#=case                (Enforce menu update)
    :: DROP FalseFalse
       24 PUTLAM ;
    DROPFALSE                   (Other messages)
  ;                               (are not handled)
  FPTR 2 72                     (Run the CHOOSE engine)
  ITE
    ::
      DROP                      (DROP current value)
      LAM res                   (Return list)
      TRUE                      (Push TRUE)
    ;
```

```
    FALSE                         (CANCL: return FALSE)
   ABND                           (Free local variables)
 ;
```

33.9 Reference

Addr.	Name	Description
072002	(^Choose3)	(meta $title #pos ::handler → ob T) (meta $title #pos ::handler → F) The main choose engine.
074002	(^Choose3Index)	(meta $title #pos ::handler → #idx T) (meta $title #pos ::handler → F) Same as ^Choose3, but returns the index of the selected item instead of the item itself. #idx starts at zero.
070002	(^Choose2)	(meta $title #pos → ob T) (meta $title #pos → F) Call Choose3Index with empty message handler. This is just :: 'DROPFALSE FPTR2 ^Choose3Index ;
073002	(^Choose3Save)	(meta $title #pos ::handler → ob T) (meta $title #pos ::handler → F) Save and restore HARDBUFF/2 around a Choose3 call.
005002	(^sysCHOOSE)	($title {} %sel → ob %1) ($title {} %sel → %0) Equivalent to User RPL CHOOSE command.
075002	(^ChooseDefHandler)	(→ ::handler) Pushed the default message handler (the one used by the CAT key) on the stack.

Addr.	Name	Description
088002	(^SaveHARDBUFF)	($\rightarrow$) Save HARDBUFF and HARDBUFF2 is a safe place.
089002	(^RestoreHARDBUFF)	($\rightarrow$) Restore HARDBUFF and HARDBUFF2 saved with SaveHARDBUFF.
077002	(^Choose3OK)	($\rightarrow$) The OK action executed by Choose3 if OK or ENTER is pressed.
076002	(^Choose3CANCL)	($\rightarrow$) The CANCEL action executed by Choose3 if CANCL or ON is pressed.

Chapter 34
Using the HP48 Browser

The HP48 browser (which is still present in the HP49) allows you to do many things. Basically, it displays a list of entries, from which you can select one or many (unlike the new HP49 browser, which only allows one item to be selected), and you can act on those entries by means of menu keys or hard key assignments.

This “old” engine has a few features that the HP49 one does not have, such as a full-screen mode. It is, however, more complicated to use. Just like the Input Form engine (see Chapter 35), it has thousands of features, and generally there are several ways to accomplish the same thing.

The browser is called by the entry `~Choose`. It expects five parameters in the stack. It will the return the results and `TRUE`, or just `FALSE`, depending on the way it was exited (more on that later). Here are the stack diagrams:

(`::Appl $Title ::Converter {}Items Init` → `result TRUE`) or
(`::Appl $Title ::Converter {}Items Init` → `FALSE`)

Here, `result` is either a list or a single object, depending on whether check marks and multiple selections are enabled.

34.1 The ::Appl Parameter

This is a program that allows configuration of several aspects of the browser. It works as other message handlers do: it is called with a bint in the stack, representing the code of the message. If the message is handled, the program should return any data required by the message and `TRUE`, otherwise it returns `FALSE`. Which means that `DROPFALSE` (which can be pushed in the stack with the command `'DROPFALSE`) is a valid value for this parameter, meaning that no messages are handled, and that default values should be used at all the times.

Here are the descriptions of some of the messages:

Code (Decimal)	Description and Stack
57	Number of lines the browser will display on the screen. The default depends on the current font, and on system flag 90. ($\rightarrow$ #)
58	Height of browser line. Probably this does not need to be changed. ($\rightarrow$ #
59	Width of browser line. Leave space for the display of arrows if the number of elements may be grater than the page size. ($\rightarrow$ #)
60	Should return `TRUE` if the browser will be full-screen, or `FALSE` if windowed. The default is windowed. ($\rightarrow$ `flag`)
61	Should return `TRUE` if check marks are allowed, thus supporting the selection of multiple items, or `FALSE` if not. The default is not to allow check marks. ($\rightarrow$ `flag`)
62	Returns the number of elements. If your program changes the number of elements during execution, you must handle this message. ($\rightarrow$ #)
63	Should return the coordinates of the upper left corner of the browser selection box. You probably do not need to change the default value. ($\rightarrow$ `#x #y`)
64	This message should return the initial difference between the marked selection and the top of page. Be sure that the difference is less than the current selection and less than the page size, otherwise the calculator may crash. ($\rightarrow$ #)
65	This message is called when the background needs to be painted. Its action can be used to draw something else on the background. ($\rightarrow$)

Code (Decimal)	Description and Stack
66	This message is called when the title needs to be painted. Its action should draw the title in `HARDBUFF`. Most of the times, this is not handled, and the title is drawn from the `$Title` parameter. (→)
67	Returns title as a grob. Most of the times, this is not handled, and the title is drawn from the `$Title` parameter. (→ `grob`)
68	If message 67 is not defined, this is called to return the title as a grob, but only for full-screen mode. (→ `grob`)
69	If message 67 is not defined, this is called to return the title as a grob, but only for windowed mode. (→ `grob`)
70	If the `$Title` parameter is not a null string, this entry is called to return a title string. This overrides the `$Title` parameter. (→ `$`)
74	This message should draw all visible lines of the browser. (→)
79	This message should display one line of the browser. If this is the selected line, this message should draw this line in inverse video or mark that it's the selected one in another way. (# →)
80	This message is an alternative to supplying the items as the `{}Items` parameter. It supplies the number of the item, and this message should returns the item. Any object can be returned; `::Converter` will be called to convert this into a string. If you want to have dynamically-changing items in the browser, this message allows that. But message 82 is probably better in this case. (# → `ob`)
81	This message converts one element into a grob. (This overrides the `::Converter` parameter.) If should return a grob with dimensions 7NULLLAMx8NULLLAM. If check marks are enabled, you must incorporate the check mark in the grob if the item is checked. (# → `grob`)

Code (Decimal)	Description and Stack
82	This message is like message 80, but the object is already returned as a string. `::Converter` is *not* called afterwards. If this message is used, you do not need to write a `::Converter`. (# → `$`)
83	Returns a list describing the menu. The format of the list is the same of `InputLine` and Input Forms, see section 37.1. (→ `{}`)
85	This message is called when the browser is started, after everything has been set. (→)
86	This is called when an item is checked or unchecked. The default action handles checking and unchecking of items pretty fine, so you probably do not need to handle this message. (# →)
87	This message is called before the browser exits. (→)
91	This is called after the ON key is pressed, or the CANCL menu key. If `TRUE` is returned, the browser exits. If `FALSE` is returned, the browser continues. (→ `flag`)
96	This is called after the ENTER key is pressed, or the OK menu key. If `TRUE` is returned, the browser exits. If `FALSE` is returned, the browser continues. (→ `flag`)

34.2 The $Title Parameter

This parameter specifies the title. There are messages that can override this parameter: 66, 67, 68, 69 and 70.

34.3 The ::Converter Parameter

This is a secondary that converts whichever kind of object is used as a list into a string for display. The stack diagram for this secondary is

(ob → $)

If you handle messages 81 or 82, you do not need to write this program to do the conversion. However, the browser allows the user to press Alpha followed by a letter to search for an object that starts with that letter and jump to it. This requires the `::Converter` parameter, even if those messages are provided. So you should ensure this parameter someway returns a string. The `DO>STR` entry can be of great use here.

34.4 The {}Items Parameter

You can specify a list of objects here, or you can specify an empty list, and use messages 80, 81 or 82 to provide the elements.

34.5 The Init Parameter

This can be either a binary integer or a list. If it is the bint 0, the browser works as a viewer, disallowing selections. If it is any other bint, it is the initially selected element.

If multiple selections are enabled, you can specify instead a list of bints, representing the initially checked elements.

34.6 Typical Browser Usage

By reading the description of the messages and the parameters above, you have probably noted that there are several ways to provide the element that will form the browseable list, and you may have been confused by that. Here, two ways to do that will be listed.

- You can provide the elements using the `{}Items` parameter, and provide a `::Converter` that will convert one of those elements into a string. You do not need to worry about messages 80, 81 or 82. This method is good if the list of elements will not change while the program is running.

- You can leave the {}Items list empty, and store the list of elements somewhere else (most likely in a lam). Then, use messages 80, 81 or 82 to return the elements. If you use messages 81 or 82, you will return elements already as a grob or as a string, and ::Converter can be a null secondary. Or you can use 80 to return some object, and then use ::Converter to make a string out of it. This method is good if the elements change while the program is running. If you use this technique, you must also handle message 62.

When the number of elements in the browser changes, run this code to adapt the browser to the changes:

```
::
  ROMPTR 0B3 03E        (Re-read # of elements)
  ROMPTR 0B3 026        (Re-read width)
  18GETLAM              (#Index)
  12GETLAM              (#NumOfElements)
  DUP#0=IT
    DROPONE
  #MIN                  (Reduce #index if #NumOfElements)
                        (was reduced)
  18PUTLAM
  FALSE ROMPTR 0B3 019  (Recalculate offset)
;
```

34.7 An Example

This example uses the browser to allow the user to enter a list of equations (inspired by the Y= window, but considerably different). Initially, the list is empty. The user then adds equations to the list. Equations can also be edited or deleted.

This program handles messages 62 and 82 to return the number of elements and an equation already converted to a string when asked for it. The equations are stored in a named LAM. Some other messages are also handled to configure other aspects of the browser.

Figure 34.1 shows this program while running.

```
::
  NULL{}                          (start with empty list)
  ' LAM EQS
```

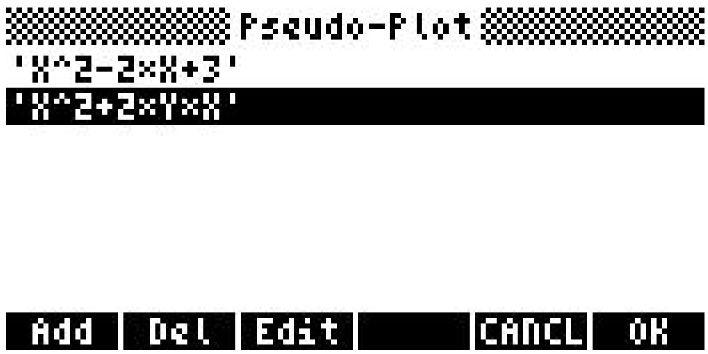

Figure 34.1: The ’48 browser example

```
  1 DOBIND

  ’ ::                          (the ::Appl parameter)
    60 #=casedrop TrueTrue      (use full screen)
    62 #=casedrop ::            (number of elements)
      LAM EQS LENCOMP
      DUP#0=IT
        #1+
      TRUE
    ;
    82 #=casedrop ::            (return nth element as str)
      LAM EQS SWAP
      NTHELCOMP
      ITE
        ::                      (convert to string)
          setStdWid
          FPTR2 ^FSTR7
        ;
       "No equations"
      TRUE
    ;
    83 #=casedrop ::            (the menu)
      {
        { "Add"
          ::
            PushVStack&Clear    (save stack)
            DoNewEqw
```

```
          DEPTH
          #0<> IT           (add eqaution)
            ::
              LAM EQS SWAP
              >TCOMP
              ' LAM EQS
              STO
              ROMPTR B3 3E  (re-read # elements)
            ;
          PopMetaVStackDROP
        ;
      }
      { "Del"
        ::
          LAM EQS
          INNERDUP
          #0=case DROP      (quit if empty)
          PushVStack&Keep   (save stack contents)
          reversym
          DROP
          18GETLAM
          ROLL
          DROP
          18GETLAM #1-
          UNROLL
          DEPTH
          {}N
          ' LAM EQS STO
          PopMetaVStackDROP (restore stack)
          ROMPTR B3 3E      (re-read # elements)
          18GETLAM          (change selected element)
          12GETLAM          (if necessary)
          #MIN
          18PUTLAM
          FALSE ROMPTR B3 19
        ;
      }
      { "Edit"
        ::
          LAM EQS           (get element)
```

```
            18GETLAM
            NTHELCOMP
            NOT?SEMI          (quit if empty)
            FPTR2 ^EQW3Edit   (edit)
            NOT?SEMI          (quit if not changed)
            18GETLAM
            LAM EQS
            PUTLIST           (replace)
            ' LAM EQS STO
          ;
        }
        NullMenuKey
        { "CANCL" FPTR2 ^DoCKeyCancel }
        { "OK" FPTR2 ^DoCKeyOK }
      }
      TRUE
    ;
    DROPFALSE
  ;

  "Pseudo-Plot"                 (title)
  ' NULL::                      (converter)
  NULL{}                        (no items - msgs are used)
  BINT1                         (initially selected elt)

  ROMPTR2 ~Choose               (run browser)

  ABND
;
```

34.8 Reference

Addr.	Name	Description
0000B3	˜Choose	(::Appl $Title ::Convert {} offset → {}' T) (::Appl $Title ::Convert {} offset → ob T) (::Appl $Title ::Convert {} offset → F) The return value is a list if checkfields are enabled, otherwise it is just the selected object. Only FALSE is returned when the user presses CANCEL.
0050B3	˜ChooseMenu0	(→ {}) Menus with "OK".
0060B3	˜ChooseMenu1	(→ {}) Menus with "CANCL", "OK".
0070B3	˜ChooseMenu2	(→ {}) Menus with "CHK", "CANCL", "OK".
0630B3	˜ChooseSimple	($title {items} → ob T) ($title {items} → F) Simple interface to the HP48 choose engine. On the HP49G, calls ˆRunChooseSimple.
004002	ˆRunChooseSimple	($title {items} → ob T) ($title {items} → F) Simple interface to the HP48 choose engine.
09F002	ˆDoCKeyCheck	(→) Toggle check on current item.
0A0002	ˆDoCKeyChAll	(→) Check all elements.
0B0002	ˆDoCKeyUnChAll	(→) Uncheck all items.
09E002	ˆDoCKeyCancel	(→) Simulate Cancel.
09D002	ˆDoCKeyOK	(→) Simulate OK.

Addr.	Name	Description
0B3002	^LEDispPrompt	($\rightarrow$) Redraw title.
0B2002	^LEDispList	($\rightarrow$) Redraw browser lines.
0B1002	^LEDispItem	(# $\rightarrow$) Redraw one line.
0150B3	(~BBMoveTo)	(# $\rightarrow$) Moves selection to line and updates display.
0190B3	(~BBRecalOff&Disp)	(flag $\rightarrow$) Recalculates offset of selected item in page, and redraws lines if the flag is TRUE.
0220B3	(~BBRunEntryProc)	($\rightarrow$) Sends message 85 to ::Appl, thus running the user-defined start-up procedure.
0230B3	(~BBReReadPageSize)	($\rightarrow$) Re-reads the size of the page (message 57).
0240B3	(~BBReReadHeight)	($\rightarrow$) Re-reads the height of the browser line (message 58).
0250B3	(~BBReReadCoords)	($\rightarrow$) Re-reads the coordinates of the browser box (message 63).
0260B3	(~BBReReadWidth)	($\rightarrow$) Re-reads the width of the browser line (message 59).
0280B3	(~BBRunENTERAction)	($\rightarrow$) Sends message 96 to ::Appl, thus running the OK action. It does not check the value returned and never exits.
0290B3	(~BBRunCanclAction)	($\rightarrow$) Sends message 91 to ::Appl, thus running the CANCEL action. It does not check the value returned and never exits.
02F0B3	(~BBReDrawBackgr)	($\rightarrow$) Redraws the background.
0370B3	(~BBGetNGrob)	(#n $\rightarrow$ grob) Returns nth element as a grob.

Addr.	Name	Description
0380B3	(~BBGetNStr)	(#n → $) Returns nth element as a string.
03B0B3	(~BBRereadChkEnbl)	(→) Re-reads whether checkmarks are enabled. (Message 61).
03C0B3	(~BBRereadFullScr)	(→) Re-reads whether to use full-screen mode. (Message 60).
03D0B3	(~BReReadMenus)	(→) Re-reads the menu. (Message 83).
03E0B3	(~BBReReadNElems)	(→) Re-reads the number of elements. (Message 62).
03F0B3	(~BBGetN)	(#n → ob) Returns nth element.
04B0B3	(~BBIsChecked?)	(#n → flag) Returns whether the given element is checked.
0520B3	(~BBUpArrow)	(→ grob) Returns up arrow as grob
0530B3	(~BBDownArrow)	(→ grob) Returns down arrow as grob
0540B3	(~BBSpace)	(→ grob) Returns a space as grob.
0590B3	(~BBPgDown)	(→) Go down one page.
05A0B3	(~BBPgUp)	(→) Go up one page.
05B0B3	(~BBEmpty?)	(→ flag) Returns TRUE if the browser has no elements.
05C0B3	(~BBGetDefltHeight)	(→ #) Returns height of lines based on the font that will be used. This value is the default height of the browser. Equivalent to FPTR 2 64.
0190E0	~BRRclC1	(→) :: LAM 'BR5 ;

34.8.1 NULLLAMs Used by the Browser

The browser uses a great number of unnamed lams to store its information. Here is a description of them:

Lam	Description	Type
1	Used by `CACHE`	n/a
2	POL exit condition	`flag`
3	Initial display status. This is a list in this format: `{ DA1IsStatFlag DA2bEditFlag DA1BadFlag DA2aBadFlag DA2bBadFlag DA3BadFlag }`	`{}`
4	Menu before browser was run	`grob 131x8`
5	Screen before browser was run	`grob 131x56`
6	Offset in page	#
7	Height of browser line	#
9	x coordinate of upper left corner of browser in `HARDBUFF`	#
10	y coordinate of upper left corner of browser in `HARDBUFF`	#
11	Page size	#
12	Number of elements	#
13	Menu	`{}`
14	Full screen?	`flag`
15	List of indexes of checked items	`flag`
16	Check marks enabled?	`flag`
17	`TRUE` if is a browser, `FALSE` if it is a viewer	`flag`
18	Current selected index	#
19	{}Items	`{}`
20	::Converter	`::`
21	$Title	`$`
22	::Appl	`::`

Chapter 35
Creating Input Forms

Input forms provide a graphical interface for entering data required by a program. Data is entered by means of several fieds, which are "spaces" that the user can fill with the apropriate data. Input forms are used in many places in the HP49. You can see one by pressing the MODE key.

It is possible to create input forms in User RPL, with the `INFORM` command, but this is not one of the easiest tasks. In System RPL, it is even more difficult. But there are several advantages: in User RPL, you can only have text fiels, in System RPL you can have check boxes or choose fields. You can also restrict the valid inputs, and make fields appear or disappear during the execution. Finally, in System RPL the input forms are considerably faster.

Input forms are created with the `^IfMain` command, which is a flashpointer. It needs lots of arguments. They are divided in three categories: label definitions, field definitions, and general information. Each label and field definition is composed of several arguments.

The `^IfMain` command referes to the new input form engine present in the HP49. The old HP48 engine is still present (and has had some speed improvements); the old `DoInputForm` command is still present and the forms created based on that command will still work. The arguments are the same for both entries, but the message handling (see section 35.4 below) has changed. There are a few incompatibilities between both engines.

The table below shows the general argument structure for the `^IfMain` command:

Parameter	**Description**
`label_1`	
`...`	Label definitions
`label_n`	
`field_1`	
`...`	Field definitions
`field_n`	
`#labels`	Number of labels

Parameter	Description
#fields	Number of fields
MessageHander	See section 35.4 below
Title	Title to be shown on top of screen

35.1 Label Definitions

Each label definition consists of three arguments:

Parameter	Description
label_text	Text to be displayed
#x_offset	X coordinate
#y_offset	Y coordinate

label_text is a string, that will be converted to a grob using the minifont. This text will be displayed at the specified coordinates. These are two bints representing the x and y positions of the label in the screen. The top-left corner has coordinates (0, 0), and coordinates increase down- and right-wards.

The new input form engine also supports a grob as argument, this grob will be directly displayed at the given coordinates.

35.2 Field Definitions

Each field definition consists of thirteen arguments:

Parameter	Description
MessageHandler	See section 35.4 below
#x_offset	X coordinate
#y_offset	Y coordinate (normally label Y coordinate - 1)
#Length	Length of field
#Height	Height of field (usually 8)
#FieldType	Type of field, see below for valid values.
#AllowedTypes	List of valid object types
Decompile	See below
"HelpString"	Help string
ChooseData	See below
ChooseDecompile	See below

Parameter	Description
`ResetValue`	Reset value
`InitValue`	Initial value

The message handler will be described below.

The x and y positions specify where the field will appear. They work similarly to the x and y positions of label definitions. Then length and height are also two bints, which specify the size of the field.

The field type is a bint which defines the type of the field:

Decimal value	Field Type
1	Text field: user can enter anything.
23	Extended text field (DoInputForm engine only): The user can enter anything, or select a variable using the filer.
12	Choose field: user must select from a list of valid values.
2	Combo field: user can select from a list of values or enter another.
32	Checkbox field.

The allowed types parameter is used in the text, extended text and combo fields: it is a list of bints, representing the allowed types of objects that can be entered in that field. You can find the object types in the table of section 29.2. You have to use the values in the "User Type" column, as bints. Other fields should specify `MINUSONE`. You can also specify `MINUSONE` for text and combo fields, this means that all kinds of objects are accepted. In the extended text field, the list of types is also used to limit the variables displayed in the filer.

Decompile is a bint that specifies how the entered objects should be displayed in the screen. Its meaning depends on the bits that are set. First, you should start with `BINT2` or `BINT4`: the former tells that numbers will be decompiled using the current mode, the latter specifies that STD mode should be used. If the field will not hold numbers, it does not make much difference in which value you choose.

After you have specified the basic way to decompile objects, you can also set some flags to configure it further. If you want to use the minifont when displaying the field valued, add 1 to the value. If you add 8 to the value, then only the first character of the string will be displayed. If the object the field holds is a list (or another composite, as a matter of fact), you can add 16 or 32, to get the first or second object of this composite, respectively, and display this object according to the rules defined by the other values. This option is sometimes useful when using choose fields, but not for normal text fields.

Note: `DoInputForm` does not support fiels decompiled with the minifont.

You can also specify the Decompile paramater as `BINT0`. If this is done, no decompilation is done: you can only use strings in the field, and they will be displayed, without the quotes, in the normal font.

The next parameter specifies the help string that will be shown in the last line of the display when that field has the focus. Enter anything you want.

The ChooseData parameter is only used in list and combo fields. Other types should have `MINUSONE` as this parameter. This paramer is the list of values that will be presented to the user for selecting. When you use a decompile value that includes the value 16, you can use a list like this:

```
{ { "label1" <foo> } { "label2" <bar> } { ... } ... }
```

This way, only the first objects will be shown, but the entire list will be returned. (Like the `INFORM` User command does.)

When using `DoInputForm` (but not `^IfMain`, you can also specify a string in the ChooseData parameter of text fields. This means that the text field will allow the user to browse the variables stored in memory and use the contents of some variable as the value of the field.

Apparently, `^IfMain` ignores the ChooseDecompile parameter. Just specify it with the same value you used for the Decompile parameter.

The reset and initial values are the contents of the field that are shown when the form is initially displayed, and when it is reset. It should be an object of the types allowed for that field, for list fields it will be one of the elements of ChooseData list. For check fields, use `TRUE` or `FALSE`. You can leave text or combo fields empty by specifying `MINUSONE` as one or both of this parameters.

35.3 Label and Field Counts

These are two bints, representing the number of labels and fields defined. Note that since they are different values, you can have labels which just show some kind of information to the user, or fields without any label definition.

35.4 Message Handlers

As with other input/output applications of the HP49, input forms use message handlers to allow the programmer to have more control over the input form. There is one message handler for each field, and one for the input form itself. The messages are passed whenever something “interessant” happens to a field or the input form, and during the initialization of the input form.

As with other message handlers, the program you provide is called with a message number (a bint) in level one, and sometimes other parameters. If the program handles the message, then it should return whatever is required by the specific (sometimes nothing). If the message is not handled, it should drop the message number and push `FALSE` in the stack, leaving any other arguments there. So, a message handler that handles no messages is simply `DROPFALSE`, which, as you know, can be conveniently pushed in the stack with `'DROPFALSE`.

In the message handling, the entries listed in the reference section below can be used to retrive information from the input form or to modify it.

Section 35.8.2 will describe each of the available messages *in* `^IfMain`*. The messages of* `DoInputForm` *are different.*

Here is a template message handler program if only one message is handled:

```
' ::
  IfMsgGetFocus     (or any other message)
  #=case
  ::
* Here is the message handling code
    TRUE (to tell the system the message was handled)
  ;
  FALSE (indicate that other messages were not handled)
;
```

And this is a template message handler for two or more messages:

```
' ::
  IfMsgOK OVER#=case
  ::
* Code. Do not forget to return TRUE.
  ;

  IfMsgType OVER#=case
```

```
    ::
  * Code for message.
    ;

  * And possibly more.

    DROPFALSE  (other messages are not handled)
  ;
```

35.5 The Title

This is a string that will be shown on the top of the display, with the small font. If it is longer than 32 characters (the width of the screen), it will be truncated and "..." will be appended.

With `^IfMain`, instead of a string you can provide your own grob to be displayed. It should have the size of 131x7 pixels.

35.6 Results Of The Input Form

The stack output, if the user exited the input form by ENTER is:

N+1: field_1
N: field_2
...
2: field_n
1: `TRUE`

If CANCEL was used to exit the form, then just `FALSE` is returned.

The value of each field depends on the types allowed for that field, and on the way the possible values of list fields are specified. If a field is empty, `xNOVAL` is returned.

35.7 An Example

This example imitates the HP49 tranfer dialog, but far from completely. There are many differences, and this example has, naturally, no functionality beyond displaying an Input Form.

The code defines all the labels and fields, and the input form has a simple message handler that handles two messages: one message to set the field that will start with the focus, and one to configure the last three softkeys to look like the ones in the Tranfer dialog. (Our keys, however, only beep when pressed...)

Figure 35.1 below displays the screen when this program is run.

Figure 35.1: The Input Form example

```
::
* Label definitons
  "Port:"   1   10
  "Type:"   70  10
  "Name:"   1   19
  "Fmt:"    1   28
  "Xlat:"   49  28
  "Chk:"    104 28
  "Baud:"   1   37
  "Parity:" 49  37
  "OvrW"    111 37

* Field definitions
  'DROPFALSE               (Message handler)
  26 9 24 8                (Position & size)
  BINT12                   (Field type: choose)
  MINUSONE                 (Types, does not apply here)
  BINT0                    (No decompilation)
  "Choose transfer port"   (Help text)
  { "Wire" }               (Possible options)
  BINT0                    (ChooseDecompile - ignored)
  "Wire" DUP               (Initial & reset values)
```

```
'DROPFALSE
92 9 36 8
BINT12
MINUSONE
BINT0
"Choose type of transfer"
{ "Kermit" "XModem" }
BINT0
"Kermit" DUP

'DROPFALSE                (Message handler)
25 18 103 8               (Position & size)
BINT1                     (Field type: text field)
{ BINT5 BINT6 }           (Allows ids and lists)
BINT2                     (Decompile with stack appearance)
"Enter names of vars to transfer"    (Help text)
MINUSONE                  (ChooseDate - n/a)
MINUSONE                  (ChooseDecompile - ignored)
MINUSONE DUP              (Initially empty)

'DROPFALSE
20 27 18 8
BINT12
MINUSONE
BINT0
"Choose transfer format"
{ "Bin" "ASC" }
BINT0
"Bin" DUP

'DROPFALSE
74 27 24 8
BINT12
MINUSONE
BINT0
"Choose character translations"
{ "None" "Newl" "\8D159" "\8D255" }
BINT0
"\8D255" DUP

'DROPFALSE
122 27 7 8
BINT12
MINUSONE
BINT0
```

```
"Choose checksum type"
{ "1" "2" "3" }
BINT0
"3" DUP

'DROPFALSE
20 36 24 8
BINT12
MINUSONE
BINT0
"Choose baud rate"
{ "1200" "2400" "4800" "9600" "15300" }
BINT0
"9600" DUP

'DROPFALSE
74 36 24 8
BINT12
MINUSONE
BINT0
"Choose parity"
{ "None" "Odd" "Even" "Mark" "Spc" }
BINT0
"None" DUP

'DROPFALSE
104 36 ZEROZERO
BINT32
MINUSONE
DUP
"Overwrite existing variables?"
MINUSONE
DUP
TrueTrue

9 9                      (Number of labels & fields)
' ::                     (InputForm message handler)
  BINT7 OVER#=case ::    (Sets initially focused field)
    DROP
    TWO
    TRUE
  ;
  BINT12 OVER#=case ::   (Configures menu softkeys)
    DROP
    { { "RECV" DoBadKey }
```

```
        { "KGET" DoBadKey }
        { "SEND" DoBadKey }
      }
      TRUE
    ;
    DROPFALSE
  ;
  "TRANSFER"                (Title)

  FPTR2 ^IfMain             (Run it)
;
```

35.8 Reference

35.8.1 Inputform

Addr.	Name	Description

020004	^IfMain	(l1..ln f1..fm #n #m msg $ → ob1..obn T)
		(l1..ln f1..fm #n #m msg $ → F)
		l = $ #x #y
		f = msg #x #y #w #h #type legal dec $hlp ChDat ChDec res init
		Starts an input form using the new engine.

2C371	DoInputForm	(l1..ln f1..fm #n #m msg $ → ob1..obn T)
		(l1..ln f1..fm #n #m msg $ → F)
		l = $ #x #y
		f = msg #x #y #w #h #type legal dec $hlp ChDat ChDec res init
		Starts an input form using the old engine.

0050B0	~IFMenuRow1	(→ {}) Returns the menu for the first menu row of an InputForm.

Addr.	Name	Description
0060B0	~IFMenuRow2	(→ {}) Returns the menu for the second menu row of an InputForm.
021004	^IfSetFieldVisible	(# T/F(fld/lbl) T/F(val) →) (# T/F(fld/blb) #0 → T/F(val)) Toggles the field or label visible or invisible. Second argument specifies if # means a field or a label. Third argument is the value to set. ZERO as third argument means to retrieve the current setting.
022004	^IfSetSelected	(# T/F(fld/lbl) T/F(val) →) (# T/F(fld/blb) #0 → T/F(val)) Toggles the field or label selected or not selected (appears in inverse video on the screen).
023004	^IfSetGrob	(# T/F(fld/lbl) grb →) Sets the grob of a field or a label (modifies the data saved in the data string).
024004	^IfSetFieldValue	(val # →) Sets the value of a field (full handling, including GROB setting).
026004	^IfGetFieldValue	(# → val) Gets the value of the Nth field.
027004	^IfGetCurrentFieldValue	(→) Gets the value of the current field.
025004	^IfSetCurrentFieldValue	(val →) Sets the value of the current field.
028004	^IfGetFieldMessageHandler	(# → prg) Retrieves a field message handler.
029004	^IfGetFieldType	(# → #type) Retrieves the field type.
02A004	^IfGetFieldObjectsType	(# → {}) Retrieves the field object type list.
02B004	^IfGetFieldDecompObject	(# → val) Retrieves the field decomp value.

Addr.	Name	Description
02C004	^IfGetFieldChooseData	(# → {}) Retrieves the field data for choose.
02D004	^IfGetFieldChooseDecomp	(# → val) Retrieves the field decomp value in case of choose.
02E004	^IfGetFieldResetValue	(# → val) Retrieves the field reset value.
02F004	^IfSetFieldResetValue	(val # →) Changes the field reset value.
030004	^IfGetFieldInternalValue	(# → val) Retrieves the field internal value.
031004	^IfDisplayFromData	(→) Displays the datastring on the screen. Takes care of the command line size.
032004	^IfGetNbFields	(→ #n) Recalls the number of fields from the data string.
033004	^IfCheckSetValue	(# val →) Checks or uncheck a check field.
034004	^IfCheckFieldtype	(ob → ob flag) Checks if an object meets the current field type requirements.
04C004	^IfGetPrlgFromTypes	({} → {}') (#FFFFF → #0) Generates a list of the allowed prologs for a field.
035004	^IfReset	(→) Resets all fields, set as the current value their reset value. Used to explode the datalist on the stack to work on it.
036004	^IfSetField	(# →) Makes a different field "current".
037004	^IfKeyChoose	(→ val) (→) If the current field is a choose field, displays the posibilities and let the user choose. A value is returned only if the user does not press CANCEL.

Addr.	Name	Description
038004	^IfKeyEdit	(→ (cmd line)) Edits the current field value if possible. You cannot edit a choose and a label choose field.
039004	^IfKeyTypes	(→ (cmd line)) (→) Displays a Choose box with all the possible types for this field. A command line is opened only if the user replies with OK.
03A004	^IfKeyCalc	(→ val) Puts the value of the field on the stack and HALT. Allows to the user to compute a new value.
03B004	^IfKeyInvertCheck	(→) Inverts the current check field value.
03C004	^IfONKeyPress	(→) On Key handler. Gives the oportunity to the user to perform his own program. Asks to the IF if we can leave. If Yes, puts a FALSE (quit with ON (if canceled)) and sets the 'Quit LAM to TRUE.
03D004	^IfEnterKeyPress	(→) Enter Key management. Gives the oportunity to the user to perform his own program. Asks to the IF if we can leave. If yes, puts the fields values on the stack put a TRUE (if validated) and sets the 'Quit LAM to TRUE.
03F004	^IfSetHelpString	($dat #n $/# → $dat') Sets the help string associated with a field. This is used by the automatic IF generator program and should not be use in other ways.

Addr.	Name	Description
040004	^IfSetTitle	($dat grb/$/# → $dat') Alters a DataString modifying the Title part. This is used by automatic IF generator program ans should not be use in other ways.
04A004	^IfInitDepth	(→) Initializes the internal depth counter. This has to be used when running a command modifying the stack
042004	^IfMain2	($dat handl {} → F) ($dat handl {} → ob1...obn T) Internal Inform Box main program. Alters a DataString modifying the Title part. This is used by automatic IF generator program ans should not be used in a different way.
043004	^IfPutFieldsOnStack	(→ ob1...obn) Puts on the stack the external value of each field.
044004	^IfSetFieldPos	(# T/F(fld/lbl) #x #y #w #h →) Changes the size and position of an object Note: You can not change the size or the X position of a label or a check field.
045004	^IfGetFieldPos	(# T/F(fld/lbl) → #x #y #w #h) Gets the size and position of an object.
047004	^IfSetAllLabelsMessages	($dat bmsg #n → $dat) Sets the text of a set of labels.
048004	^IfSetAllHelpStrings	($dat bmsg #n → $dat) Sets the Help String of all fields.
04D004	^IsUncompressDataString	($dc → $dat) Uncompresses a compressed data string.

35.8.2 Input Form Messages

The names of the messages are `DEFINE`s for the numbers. You will find this `DEFINE`s in the `inputform.h` file in the `include` subdirectory.

35.8.2.1 IfMsgKeyPress — 0

This message is sent after each keypress, first to the active field, then to the input form. If the field handles the message, the normal input form key handling is *not* executed.

Input	2: #KeyPlane
	1: #KeyCode
Output (if handled)	2: ::Key_Handler_Program
	1: `TRUE`
Output (if not handled)	3: #KeyPlane
	2: #KeyCode
	1: `FALSE`

35.8.2.2 IfMsgLooseFocus — 1

This is sent to a field when it is about to loose the focus. You can do anything here, including taking back the focus. If this is done, then no IfMsgGetFocus message will be sent to this field.

Input	1: #Field_That_Will_Get_Focus
Output	2: #Field_That_Will_Get_Focus
	1: `TRUE` or `FALSE`

35.8.2.3 IfMsgNewField — 2

This message is sent to the IF just before a new field receives the focus. There is no input, and the output can be either `TRUE` or `FALSE`.

35.8.2.4 IfMsgGetFocus — 3

This message is sent to the field that has just received the focus. There is no input, and the output can be either `TRUE` or `FALSE`.

35.8.2.5 IfMsgGetFieldValue — 4

This message is sent to the current field. It has as input the internal data of the field, and this message can be used to return the external value (which is displayed in the screen). Using this and the IfMsgSetFieldValue messages, it is possible, for example, to store only an offset to the current element when you have a list of fixed values, instead of the actual element.

Input	1: Internal value
Output (if handled)	2: External value 1: TRUE
Output (if not handled)	2: Internal value 1: FALSE

35.8.2.6 IfMsgSetFieldValue — 5

The complimentary message of IfMsgSetFieldValue: it gives as input the "external" (or user) value, and the internal value should be returned. If you want a message to be called after each change in the value of a field, this is the one. You can leave the value given as input unchanged, naturally.

Input	1: External value
Output (if handled)	2: Internal value 1: TRUE
Output (if not handled)	2: External value 1: FALSE

35.8.2.7 IfMsgGetFieldGrob — 6

This message is sent to the current field. If you decide to handle it, you will have to set the grob that is displayed in the field (you can use the `^IfSetGrob` entry for this). If you do so, then the standard code of the Input Form that would do this is not called.

Input	2: #Field 1: Value
Output	2: #Field 1: TRUE or FALSE

Here is an example of handling this message:

```
::
  OVER TRUE ROT SWAP (Number, number, TRUE, value)
  $>grob
  FPTR2 ^IfSetGrob
;
```

35.8.2.8 IfMsgSetFirstField — 7

This message is sent during initialization to the input form handler, to get the number of the first field that will be selected. It makes no difference wheter you return TRUE or FALSE, just change the number if desired.

Input	1: #Field
Output	2: #Field
	1: TRUE or FALSE

35.8.2.9 IfMsgFieldReset — 10

This message is sent to a field that is going to be reset. It is possible to modify the value of the field, if desired.

Input	1: Value
Output	2: Value, possibly modified
	1: TRUE or FALSE

35.8.2.10 IfMsgGetMenu — 11

This message is sent to the input form handler during initialization, and can be used to provide a menu for the input form. The menu is in the format described in section 37.1.

Input	1: Menu
Output (if handled)	3: Original menu
	2: New menu
	1: TRUE
Output (if not handled)	2: Original menu
	1: FALSE

35.8.2.11 IfMsgGet3KeysMenu — 12

This message can be used to change the last three softkeys of the first row of the standard input form menu. If handled, it should return a list with three sub-lists, each being a key definition.

Input	None
Output (if handled)	2: List
	1: TRUE
Output (if not handled)	1: FALSE

35.8.2.12 IfMsgCancel — 13

This allows the user to replace the default quit handler. This message is called when the ON key or the CANCL softkey are pressed. If it is handled, then no standard code is run. The user should alter the value of LAM `'Quit` to indicate the POL that the input form should be ended.

Input	None
Output (if handled)	1: TRUE
Output (if not handled)	1: FALSE

35.8.2.13 IfMsgCancelKey — 14

This message is sent to the input form handler when the user requests the input form to end via the CANCEL key. The programmer can prevent the input form to end if there is invalid input, for example.

Input	None
Output (if handled)	2: TRUE or FALSE
	1: TRUE
Output (if not handled)	1: FALSE

When the message is handled, a TRUE in level two means that the input form should end, FALSE means it should continue.

35.8.2.14 IfMsgOK — 15

This is similar to the IfMsgCancel message, but for the OK softkey or ENTER key.

35.8.2.15 IfMsgKeyOK — 16

This message is sent to the input form handler when the user requests the input form to end via the OK key. The programmer can prevent the input form to end if there is invalid input, for example.

Input	None
Output (if handled)	2: TRUE or FALSE
	1: TRUE
Output (if not handled)	1: FALSE

When the message is handled, a TRUE in level two means that the input form should end, FALSE means it should continue.

35.8.2.16 IfMsgChoose — 17

When the user presses the CHOOS softkey in a choose field, this message is sent, first to the field, and then to the input form (if it was not handled by the field). If it is handled by either, then no standard code is run, and you have to display the choose box yourself.

There are no arguments, and you should return TRUE to prevent the standard code to be executed if you desire that, after having displayed your choose box.

35.8.2.17 IfMsgType — 18

This message is sent to the input form when the TYPES softkey is pressed. If it is handled, no standard code is executed.

There are no arguments, and you should return TRUE to prevent the standard code to be executed if you desire that, after having displayed your choose box.

35.8.2.18 IfMsgCalc — 19

This message is sent to the form when the CALC softkey is pressed. If it is handled, no standard code is executed.

There are no arguments, and you should return TRUE to prevent the standard code to be executed if you desire that, after having displayed your choose box.

35.8.2.19 IfMsgNewCommandLine — 20

This message is sent to the input form when a new command line is created. The system does not care if the message is handled or not. It is just to give the programmer the opportunity to perform anything he needs. There are no inputs, and the output is just TRUE or FALSE, without any difference.

35.8.2.20 IfMsgOldCommandLine — 21

This message is sent to the input form when a command line is cancelled. See message IfMsgNewCommandLine above for more details.

35.8.2.21 IfMsgCommandLineValid — 22

This is sent to a field when the command line is validated.

Input	None; a command line is present
Output (if handled)	No command line; elements in the stack and TRUE
Output (if not handled)	1: FALSE

35.8.2.22 IfMsgDecompEdit — 23

This is sent to a field when an object needs to be decompiled for editing.

Input	1: Object
Output (if handled)	2: String 1: TRUE
Output (if not handled)	1: FALSE

35.8.2.23 IfMsgNextChoose — 24

This message is sent to a choose field when the +/- key is pressed. If it is handled, then the default action is not run. There are no inputs and no outputs, except for the TRUE/FALSE.

35.8.2.24 IfMsgEdit — 25

This is sent to a field when the EDIT softkey is pressed. The input is the current value of the field, the output can be nothing, a modified command line, something in the stack, or a modified field. If this message is handled, then the default code is not run.

Chapter 36
The Display

There are two screens available to the programmer while programming in System RPL: the graphics screen, which is visible, for example, in the Plot application (and referred as `PICT` in User RPL), and the text screen, which is the graphic visible in the standard stack environment. Whenever possible, the latter should be used, leaving the graphics screen untouched, because that is supposedly a user resource, which should not be changed by programs.

36.1 Display Organization

The HP49 system RAM contains three dedicated graphic objects (subsequently called grobs) used for display purposes. The commands below return each of this grobs:

Command	Grob
`ABUFF`	Text grob (stack)
`GBUFF`	Graphics grob (`PICT`)
`HARDBUFF`	Either the text or gaphics grob, whichever is active.
`HARDBUFF2`	Menu labels

One thing to note is that the words above return just pointer to the grob, so if you alter the grob, the display will also be altered automatically. Most of the times that is the desired behavior, but if you do not want that, call `TOTEMPOB` after using any of the words above to make a unique copy in temporary memory. See section 24.1.4 for more information on temporary memory and object references.

The text and graphic grobs may be enlarged, and may be scrolled. The menu label grob has a fixed size of 131x8 pixels.

The command `TOADISP` makes the text grob visible, and the command `TOGDISP` makes the graphic grob visible.

The text grob is divided in three regions. The display areas are numbered one, two and three. In many words you will find "DA", which means "Display Area". Figure 36.1 shows each of this areas.

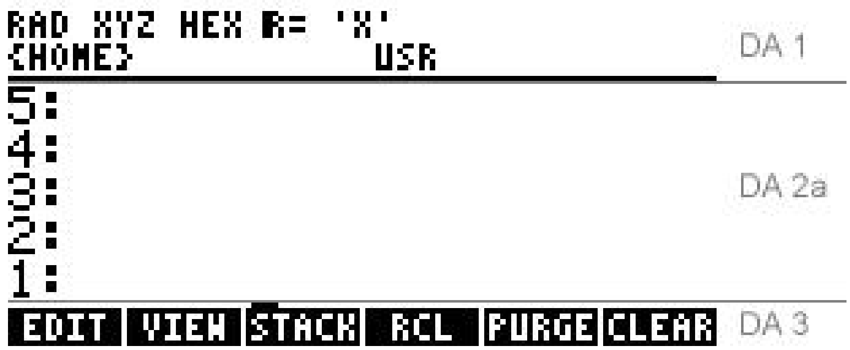

Figure 36.1: The Display Areas

Display area 2 is actually divided in two areas: 2a and 2b. Normally, only area 2a is visible, and it occupies the whole DA 2.

36.2 Preparing the Display

Two words establish control over the text display: `RECLAIMDISP` and `ClrDA1IsStat`. The first does the following:

- Assures the current display is the text one;
- Clears the text display;
- If necessary, resizes the text display to the default size of 131x56 pixels.

This word works very similarly to the user word `CLLCD`, the difference is that `CLLCD` never resizes the text display.

The word `ClrDA1IsStat` is optional, but most of the time it should be used. It suspends the ticking clock display temporarily. Most graphical programs would not want to have that clock displayed.

When the menu is not necessary, use the word `TURNMENUOFF` to hide the menu and enlarge the text grob to 131x64 pixels. It is turned on again with `TURNMENUON`. For more details on the menu, see Chapter 37.

The suggested template for an application that uses the text display is:

```
::
  ClrDA1IsStat     (suspend clock)
  RECLAIMDISP      (set, clear and resize text display)
  TURNMENUOFF      (turn off menu if desired)

  <application>

  ClrDAsOK         (redraw LCD)
    -or-
  SetDAsTemp       (freeze the whole display)
;
```

36.3 Controlling Display Refresh

In some programs, it is desired that, after the application ends, the screen is not redrawn, but continues frozen so that the user can see the results, like the User RPL the command `FREEZE` does. Other times, it is desired that the display is returned back to normal. In System RPL, several words serve those purposes. The most used ones are listed below; the whole list is in the reference section below.

Word	Action
`SetDA1Temp`	Freezes display area 1.
`SetDA2OKTemp`	Freezes display area 2.
`SetDA3Temp`	Freezes display area 3.
`SetDA12Temp`	Freezes display areas 1 and 2.
`SetDAsTemp`	Freezes the whole display.
`ClrDA1OK`	Redraws display area 1.
`ClrDA2OK`	Redraws display area 2.
`ClrDA3OK`	Redraws display area 3.
`ClrDAsOK`	Redraws the whole display.

36.4 Clearing the Display

The following words clear `HARDBUFF`, entirely or in part. Remember that `HARDBUFF` refers to the currently displayed grob, either the text or the graph display. Except from `BLANKIT`, no words take or return arguments.

Word	Action
CLEARVDISP	Clears entire HARDBUFF.
BlankDA1	Clears display area 1.
BlankDA2	Clears display area 2.
BlankDA12	Clears display areas 1 and 2.
Clr16	Clears top 16 rows.
Clr8	Clears top 8 rows.
Clr8-15	Clears rows 8 to 15 (second status line).
CLCD10	Clears status and stack area.
CLEARLCD	Clears entire display.
BLANKIT	(#start_row #rows →) Clears #rows from HARDBUFF.

36.5 Displaying Text

There are two fonts in the HP49: the "system font" and the "minifont". Both can be changed by the user, but it is only possible to access two fonts at each time. The height of the system font (or of its characters, to be precise) can vary, but its characters are always five pixels wide. The size of the minifont is fixed: each character is 3x5 pixels.

There are commands to display text in the system font directly, but not for the minifont. In the latter case, it is necessary to convert the text into a grob and display the grob. The list below only describes the most used ones, for a complete list see the reference section below.

36.5.1 System Font

To display text using the system font, use the commands DISPROW1, DISPROW2... to DISPROW10, which take a string as argument and display it in the specified line of the display. Note that, depending on the size of selected system font and wheter the menu is displayed, some of these commands may not be used. You can always safely display text on the first seven lines, even with the largest system font.

36.5.2 Minifont

As said above, displaying text with the minifont is more complicated. First, put a string in the stack and run the command `$>grob`. This will return a grob representing with the string in the minifont. You now need to display this grob on the screen. You can use `GROB!` or `XYGROBDISP` for that. For more information on these words and for a general treatment of grobs, turn to Chapter 15. In this same chapter, you will find some other commands which might be more convenient for displaying text with the minifont.

36.5.3 Displaying Warnings

The word `FlashWarning` is used to display a warning message. It beeps, and then displays the given string in a message box. The user must press OK in order to continue.

Instead of `FlashWarning`, one can use `FlashMsg`, which displays the text in the status line, and does not beep. To display a message in the status area, it uses the word `DISPSTATUS2`, which takes a string with a line break in it, and displays it using the two lines of the status area. After a short pause, the display is returned to the state it was before and the program continues.

36.6 Reference

36.6.1 Display Organization

Addr.	Name	Description
26166	TOADISP	(→) Sets the text display as the active.
2616B	TOGDISP	(→) Sets the graphic display as the active.
25FA4	ABUFF	(→ textgrob) Returns the text grob to the stack.
26076	GBUFF	(→ graphgrob) Returns the graphic grob to the stack. The HP49 extable address for `ExitAction!` is the same, but this must be a bug.

Addr.	Name	Description
2608F	HARDBUFF	(→ dispgrob) Returns the current grob to the stack.
26094	HARDBUFF2	(→ menugrob) Returns the menu grob to the stack.
25EDE	HARDHEIGHT	(→ #height) Returns the height of HARDBUFF.
25ED5	GBUFFGROBDIM	(→ #height #width) Returns dimensions of graphic grob.

36.6.2 Preparing the Display

Addr.	Name	Description
25EF4	RECLAIMDISP	(→) Activates the text grob, clears it and sets the default size.
2EE7D	ClrDA1IsStat	(→) Suspends clock display.
2EEFD	MENUOFF?	(→ flag) Returns TRUE if the menu grob is off.
2F034	TURNMENUOFF	(→) Turns off menu display, enlarges ABUFF to fill screen.
2F031	TURNMENUON	(→) Turns menu grob on.
2EEFC	MENUOFF	(→)
26247	GetHeader	(→ #) Gets header size in lines (0-2).
26283	SetHeader	(# →) Sets header size in lines (0-2).
26099	HEIGHTENGROB	(grob #rows →) Heightens graph or text grob.
260A3	KILLGDISP	(→) Clears graph display by setting it to NULLGROB. See DOERASE.
2EEF9	DOERASE	(→) Erases the graphics display grob without changing its size.

36.6.3 Immediate Refresh

Addr.	Name	Description
2EF67	SysDisplay	($\rightarrow$) Redisplays all required areas. Does it immediately, without waiting for the current command to finish.
2F19F	?DispCommandLine	($\rightarrow$) Redisplays the command line now if necessary.
2F19E	DispCommandLine	($\rightarrow$) Redisplays the command line now.
2EE5A	DispEditLine	($\rightarrow$) Just calls DispCommandLine.
2DFCC	?DispMenu	($\rightarrow$) Redisplays the menu now if no key is waiting in the buffer. Even better is this: :: DA3OK?NOTIT ?DispMenu ;
2DFF4	DispMenu.1	($\rightarrow$) Displays menu now.
2DFE0	DispMenu	($\rightarrow$) :: DispMenu.1 SetDAsValid ;
2C341	?DispStack	($\rightarrow$) Redisplays the stack now if necessary.
2C311	?DispStatus	($\rightarrow$) Redisplays the status area now if necessary.
2C305	DispStatus	($\rightarrow$) Displays the status area now.
2C2F9	DispStsBound	($\rightarrow$) Displays a horizontal line at y=14, normally the separation between header and stack.
2A7F7	DispTimeReq?	($\rightarrow$ flag) Is time display required? Checks system flag 40 and something else.
2F300	DispILPrompt	($\rightarrow$) Redisplays the InputLine prompt, i.e. refreshes the region between the command line and the header during InputLine. Requires a string (the prompt) in 4LAM.

Addr.	Name	Description
26260	nDISPSTACK	($prompt #height #header flag flag →) Used by DispILPrompt.

36.6.4 Controlling Display Refresh

Addr.	Name	Description
2EE8D	ClrDA1OK	(→)
2EE8E	ClrDA2aOK	(→)
2EE8F	ClrDA2bOK	(→)
2EE90	ClrDA2OK	(→)
2EE6E	ClrDA3OK	(→)
2EE6D	ClrDAsOK	(→)
2EE62	DA1OK?	(→ flag)
2EE63	DA3OK?	(→ flag)
2EE66	DA2aLess1OK?	(→ flag)
2BF3A	DA1OK?NOTIT	(→) Does DA1OK?, NOT then IT.
2BF53	DA2aOK?NOTIT	(→) DA2aOK?, NOT then IT.
2BF6C	DA2bOK?NOTIT	(→) DA2bOK?, NOT then IT.
2BF85	DA3OK?NOTIT	(→) Does DA3OK?, NOT then IT.
2EE69	SetDA1Temp	(→)
2EE8A	SetDA2aTemp	(→)
2EE6A	SetDA2bTemp	(→)
2EEA7	ClrDA2bTemp	(→)
2F37A	SetDA2OKTemp	(→)
2EE6B	SetDA3Temp	(→)
2EE71	SetDA12Temp	(→)
2EE64	SetDAsTemp	(→)
2EEA5	SetDA2bTempF	(→)
2EE67	SetDA1Valid	(→)
2EF98	SetDA2aValid	(→)
2EE68	SetDA2bValid	(→)
2EE91	SetDA2Valid	(→)
2EF99	SetDA3Valid	(→)

Addr.	Name	Description
2EEA0	SetDA3ValidF	($\rightarrow$)
2EE78	SetDA1Bad	($\rightarrow$)
2EE74	ClrDA1Bad	($\rightarrow$)
2EEB0	DA1Bad?	($\rightarrow$ flag)
2EE79	SetDA2aBad	($\rightarrow$)
2EE75	ClrDA2aBad	($\rightarrow$)
2EEB1	DA2aBad?	($\rightarrow$ flag)
2EE7A	SetDA2bBad	($\rightarrow$)
2EEB3	ClrDA2bBad	($\rightarrow$)
2EEB2	DA2bBad?	($\rightarrow$ flag)
2EE7B	SetDA3Bad	($\rightarrow$)
2EEB5	ClrDA3Bad	($\rightarrow$)
2EEB4	DA3Bad?	($\rightarrow$ flag)
2EE72	SetDA1NoCh	($\rightarrow$)
2EE73	SetDA2aNoCh	($\rightarrow$)
2EE76	SetDA2bNoCh	($\rightarrow$)
2EE81	ClrDA2bNoCh	($\rightarrow$)
2EEB7	DA2bNoCh?	($\rightarrow$ flag)
2EE93	SetDA2NoCh	($\rightarrow$)
2EE6F	SetDA12NoCh	($\rightarrow$)
2EE77	SetDA3NoCh	($\rightarrow$)
2EE70	SetDA13NoCh	($\rightarrow$)
2EE94	SetDA23NoCh	($\rightarrow$)
2EE65	SetDA12a3NCh	($\rightarrow$) aka: SetDA12a3NoCh
2F379	SetDA123NoCh	($\rightarrow$)
2EE7C	SetDAsNoCh	($\rightarrow$)
2EE6C	SetDA2aEcho	($\rightarrow$)
2EEAC	SetDA1IsStat	($\rightarrow$)
2EEAE	SetNoRollDA2	($\rightarrow$)
2EEAF	ClrNoRollDA2	($\rightarrow$)
2EEAB	DA1IsStatus?	($\rightarrow$ flag)
2EE7F	SetDA2bIsEdL	($\rightarrow$)
2EE7E	DA2bIsEdL?	($\rightarrow$ flag)
2EE80	ClrDA2bIsEdL	($\rightarrow$)

36.6.5 Clearing the Display

Addr.	Name	Description
25E7E	BLANKIT	(#startrow #rows →) Clears #rows from HARDBUFF, starting at #startrow.
26021	CLEARVDISP	(→) Clears HARDBUFF.
2EED4	Clr8	(→) Clears top eight rows (first status line).
2EED5	Clr8-15	(→) Clears 2nd status line.
2F15E	Clr16	(→) Clears top 16 rows.
2EF5E	BlankDA1	(→) Clears status area from HARDBUFF.
2F31C	BlankDA2a	(→) Clears display area DA2a.
2F31B	BlankDA2	(→) Clears display areas DA2a and DA2b.
2EE5C	BlankDA12	(→) Clears display areas DA1 and DA2
261C0	CLCD10	(→) Clears status and stack areas.
261C5	CLEARLCD	(→) Clears whole display.
2EF05	DOCLLCD	(→) Like user word CLLCD.

36.6.6 Annunciator and Modes Control

Addr.	Name	Description
2613E	SetLeftAnn	(→) Sets left-shift annunciator.
2603A	ClrLeftAnn	(→) Clears left-shift annunciator.
26148	SetRightAnn	(→) Sets right-shift annunciator.

Addr.	Name	Description
2603F	ClrRightAnn	(→) Clears right-shift annunciator.
26139	SetAlphaAnn	(→) Sets alpha annunciator.
26035	ClrAlphaAnn	(→) Clears alpha annunciator.
25EE9	LockAlpha	(→) Sets alpha mode, annunciators, etc.
25F08	UnLockAlpha	(→) Clears alpha mode, annunciators, etc.
2649F	(ClrBusyAnn)	(→) Clears the busy annunciator.
26143	SetPrgmEntry	(→) Sets program-entry mode.
2610C	PrgmEntry?	(→ flag) Is program-entry mode set?
25EBE	Do1st/2nd+:	(→ :: <ob1> ; (PRG mode)) (→ :: <ob2> <rest> ; (no PRG mode)) If in program mode, executes the next object after it. If not in program mode, executes the rest of the stream starting at the second object after it.
25719	SetAlgEntry	(→) Sets algebraic-entry mode.
2571E	ClrAlgEntry	(→) Clears algebraic-entry mode.
256EA	AlgEntry?	(→ flag) Is algebraic-entry mode set?
25EDF	ImmedEntry?	(→ flag) Returns TRUE if immediate-entry mode (program and algebraic-entry modes cleared).
25E74	?ClrAlg	(→) Clears AlgEntry mode if set.
25E75	?ClrAlgSetPr	(→) Clears AlgEntry mode if set and sets ProgramEntry mode.

36.6.7 Window Coordinates

Addr.	Name	Description
2F384	TOP8	(→ HBgrob #x1 #y #x1+131 #y1+8) Returns coordinates of first status line.
2F36C	Rows8-15	(→ HBgrob #x1 #y1+8 #x1+131 #y1+16) Returns coordinates of second status line.
2F383	TOP16	(→ HBgrob #x1 #y1 #x1+131 #y1+16) Returns coordinates of status area.
2617F	WINDOWCORNER	(→ #x #y) Gets coordinates of corner of window.
2EED6	HBUFF_X_Y	(→ HBgrob #x #y) Returns current grob and window coordinates.
2F352	LEFTCOL	(→ #x) Gets x-coordinate of left column.
2F36B	RIGHTCOL	(→ #x) Gets x-coordinate of right column.
2F385	TOPROW	(→ #y) Gets y-coordinate of top row.
2F31D	BOTROW	(→ #y) Gets y-coordinate of bottom row.
26198	WINDOWXY	(#x #y →) Sets corner coordinates.

36.6.8 Scrolling the Display

Addr.	Name	Description
26193	WINDOWUP	(→) Moves display one pixel up.
26184	WINDOWDOWN	(→) Moves display one pixel down.
26189	WINDOWLEFT	(→) Moves display one pixel left.
2618E	WINDOWRIGHT	(→) Moves display one pixel right.
2F370	SCROLLUP	(→) Moves display one pixel up, checks for corresponding key being pressed.

Addr.	Name	Description
2F36D	SCROLLDOWN	(→) Moves display one pixel down, checks for corresponding key being pressed.
2F36E	SCROLLLEFT	(→) Moves display one pixel left, checks for corresponding key being pressed.
2F36F	SCROLLRIGHT	(→) Moves display one pixel right, checks for corresponding key being pressed.
2F34A	JUMPTOP	(→) Jumps to top of display.
2F347	JUMPBOT	(→) Jumps to bottom of display.
2F348	JUMPLEFT	(→) Jumps to left of display.
2F349	JUMPRIGHT	(→) Jumps to right of display.
2F38D	WINDOWTOP?	(→ flag) Is window at the top?
2F38A	WINDOWBOT?	(→ flag) Is window at the bottom?
2F38B	WINDOWLEFT?	(→ flag) Is window at the left?
2F38C	WINDOWRIGHT?	(→ flag) Is window at the right?

36.6.9 Displaying Objects

Addr.	Name	Description
2F21D	ViewObject	(ob →)
2F21E	ViewStrObject	(flag $ → F) Flag decides if it should be possible to toggle TEXT/GRAPH.
2F21F	ViewGrobObject	(flag grob → F) Flag decides if it should be possible to toggle TEXT/GRAPH.

Addr.	Name	Description
25F12	sstDISP	(ob →) Displays ob in status line. Used for single stepping during debugging.
0C1007	^SCROLLext	(grob →) Launches PICT environment.
2EF61	WINDOW#	(#x #y →) Internal PVIEW, displays PICT starting at the given coordinates.

36.6.10 Displaying Text

Addr.	Name	Description
25EB4	DODISP	(ob %row →) Displays any object in specified row.
25FB8	DISPROW1	($ →) aka: DISP@01, BIGDISPROW1
25EAB	DISPROW1*	($ →) Displays relative to window corner.
25FBD	DISPROW2	($ →) aka: DISP@09, BIGDISPROW2
25EAC	DISPROW2*	($ →) Displays relative to window corner.
25FC2	DISPROW3	($ →) aka: DISP@17, BIGDISPROW3
25FC7	DISPROW4	($ →) aka: DISP@25, BIGDISPROW4
25FCC	DISPROW5	($ →)
261F7	DISPROW6	($ →)
25FD1	DISPROW7	($ →)
25FD6	DISPROW8	($ →) May not be possible depending on the size of the font and whether the menu is on or off.
25FDB	DISPROW9	($ →) May not be possible depending on the size of the font and whether the menu is on or off.
25FE0	DISPROW10	($ →) May not be possible depending on the size of the font and whether the menu is on or off.

Addr.	Name	Description
25FB3	DISPN	($ #row →) aka: BIGDISPN
25EBC	Disp5x7	($ #start #max →) Displays string on multiple lines, starting at #start and no using more than #max rows. New lines must be manually specified. Segments longer than 22 characters are truncated and appended with "...".
25EAD	DISPSTATUS2	($ →) Displays message in status area using two lines.
38C00	(DISPST2&FREEZE)	($ →) DISPSTATUS2 and freeze status area.
2EEFF	DispCoord1	($ →) Displays $ in menu grob using minifont.
2F32B	DISPCOORD2	($ →) Displays $ in menu grob using minifont and waits for a key. Then refreshes menu display.
25FE5	DISPLASTROW	($ →) Displays $ in the last stack display row, just above the menu.
25FEA	DISPLASTROWBUT1	($ →) Displays $ in the last stack display row. If menu is turned on it can cover displayed text.
25ED4	FlashMsg	($ →) Displays message in status area, then restores it to normal.
2EE61	FlashWarning	($ →) Displays message in a message box and beeps. Waits for OK to be pressed.
2F1A5	AskQuestion	($ → flag) Use the string to aks the user a question with yes/no in a choose box.
02E002	^DoAlert	($ →) Displays alert messagebox.
2EE60	DoWarning	($ →) Displays message, beeps and freezes status area.

Addr.	Name	Description
007002	ˆCk&DoMsgBox	($ #x #y grob menu → T) Displays a message box with a grob in the upper left corner and the specified menu. The meaning of #x and #y is unclear.
0040B1	˜MsgBoxMenu	(→ {}) The messsage box menu, with just the OK key.

36.6.11 Fonts

Addr.	Name	Description
2621A	FONT>	(→ font) Recalls system font.
2625B	MINIFONT>	(→ minifont) Recalls the current minifont.
25F15	>FONT	(font →) Sets system font.
2620B	>MINIFONT	(minifont →) Sets the current minifont.
26288	StackLineHeight	(→ #) Returns height of text grob minus size of header and menu.
26242	GetFontStkHeight	(→ #) Returns stack font height (used for display stack rows). aka: StackFontHeight

Chapter 37
The Menu

The menu line is divided in six parts, one for each key, each eight pixels high and 21 pixels wide. The starting columns for each menu key label in `HARBDUFF2` are:

Hex	Dec	Softkey	Hex	Dec	Softkey
0	0	First softkey (F1)	42	66	Fourth softkey (F4)
16	22	Second softkey (F2)	58	88	Fifth softkey (F5)
2C	44	Third softkey (F3)	6E	110	Sixth softkey (F6)

The command `DispMenu.1` redisplays the current menu; and the command `DispMenu` redisplays the current menu and then calls `SetDA3Valid` to freeze the menu display area (display area 3).

The words below convert several kinds of objects to menu labels and display them at the specified column:

Word	Stack and action
`Str>Menu`	(`#col $` →) Makes and displays a standard menu label.
`Id>Menu`	(`#col id` →) Recalls id and displays standard or directory label, depending on the contents.
`Grob>Menu`	(`#col grob` →) Displays a grob as a menu label.
`Seco>Menu`	(`#col ::` →) Evaluates secondary and uses results to create and display appropriate menu label.

The words below convert strings to the four different kinds of grobs available. All of them take a string and return a grob as arguments

Word	Action
`MakeStdLabel`	Makes a black label (standard).
`MakeBoxLabel`	Makes label with a box inside.
`MakeDirLabel`	Makes directory label (bar above).
`MakeInvLabel`	Makes white label (like in Solver).

37.1 Menu Format

A menu is either a list

`{ MenuKey1 MenuKey2 ... MenuKeyN }`

or a program

`:: <Settings> { MenuKey1 MenuKey2 ... MenuKeyN } ;`

which returns such a list and optionally changes of the default menu properties installed by `InitMenu`.

Each menu key can be any of the following:

- `NullMenuKey`
- `KeyObj`
- `{ LabelObj KeyProcNS }`
- `{ LabelObj { KeyProcNS KeyProcLS } }`
- `{ LabelObj { KeyProcNS KeyProcLS KeyProcRS } }`

`LabelObj` is the object to be displayed as the label. If it is a program with `TakeOver` as the first command, it is evaluated with the x-position of the label on the stack and must return the argument(s) for the `LabelDef` progam (normally the x-position of the label and the object to display as a label).

If you do not override the `LabelDef` command (most of the times you will not), then `LabelObj` can be any object, but genereally it is a string or a 21x8 grob.

`KeyProc` is the action taken upon key press. It will be executed by a special executor which takes appropriate actions depending upon the object type. If `KeyProc` is a program with `TakeOver` as the first command, it will override the normal executor. NS here means this is the action when the menu key is pressed unshifted (think of No-Shift). Similarly, LS and RS means the actions run when the key is pressed left- or right-shifted, respectively.

37.2 Menu Properties

The menu system of the HP49 provides an amazing flexibility. Besides the normal actions, a menu has many properties which define the appearance of labels and the specific actions taken upon keypresses, actions to take when the context changes or a different menu is installed etc.

The properties a menu carries are:

Word	**Stack and action**
`MenuDef`	The current menu.
`MenuKeys`	The menu keys in a list.
`MenuRow`	The menu page.
`LabelDef`	The label builder for menu.
`MenuRowAct`	Action taken when menu row changes or when LastMenu is reinstalled.
`ExitAction`	Action taken when menu changes. Normally this action saves the current menu as `LastMenu`.
`TrackAct`	Action taken when the context (the current directory) changes.
`ReviewKey`	Action taken when REVIEW key (Rightshift DOWN) is pressed.
`MenuKeysNS`	Action taken when menu key is pressed.
`MenuKeysLS`	Action taken when menu key is pressed left-shifted.
`MenuKeysRS`	Action taken when menu key is pressed right-shifted.
`BadMenu?`	Must the menu be be redrawn?
`Rebuild?`	Has the menu row changed?
`Track?`	If context has changed is there a prg to execute?

Examples for the `TrackAct` property are:

- SolverMenu has `DoSolveMenu` as `TrackAct`, because there might be another EQ variable to use.
- The Custom menu just restarts itself because the value of the CST variable may have changed. (CstTrack = `:: NoExitAction MenuDef@ InitMenu ;`)

Most menu properties can be modified using supported entry points. Here is an example for doing so. The following program sets a modified VAR menu, which allows variables to be protected against being overwritten with a left-shifted menukey.

```
::
  MenuMaker
  ::
    ROMPTR A9 2                   (the builtin VAR menu)
    '
    ::
      DUP
      DUPTYPECSTR? NOT_IT DECOMP$  (make a string)
      SWAP
      ID prtct
      ITE                         (select label type)
        MakeInvLabel
       MakeStdLabel
      Grob>Menu                   (display label)
    ;
    LabelDef!
    '
    ::
      ID prtct
      NOTcase xSTO
      DECOMP$ "\0A is protected" &$
      FlashWarning
    ;
    MenuKeyLS!
  ;
  DoMenuKey
;
```

This does several things:

1. Gets the normal VAR menu in order to pass it to `DoMenuKey`.
2. It modifies the `LabelDef` property in a way that the protected variables will have an inverted label in the menu.
3. It modifies the `MenuKeyLS` property in a way that it exits with an error message if the relevant variable is protected.

The program in `ID prtct` is an ID selector which has the stack diagram (`id` → `flag`) and must decide if a given ID should be protected. Here are some possibilities:

1. `:: DROPTRUE ;` — All variables are protected

2. `:: DROPFALSE ;` — No variables are protected

3. `:: ID>$ CAR$ CHR_% EQUAL ;` — Variables with names starting with "%" are protected.

4.
```
::                       (all variables in the list)
  ID PROTECTED           (stored in the variable)
  OVER EQUALPOSCOMP      ('PROTECTED' are protected)
  #0<>
;
```

Note that if the variable `prtct` does not exist or does not follow the required stack diagram, the calculator may crash. You might want to modify the program to put in better protection against user errors. Also note that this only protects against storing using left-shift and a menukey. It will not protect against using the `STO` command or the filer, naturally.

37.3 Reference

37.3.1 Menu Properties

Addr.	Name	Description
04A41	GETDF	(#menukey → ob) Gets the definition of a menu key from THOUCHTAB. #menukey = #1..#6
04A0B	GETPROC	(#menukey → ob) Gets the definition of a menu key from THOUCHTAB. #menukey = #1..#6. With #7, get the executor.
2580E	SetRebuild	(→) Sets the flag that the menu needs to be rebuild.
260B7	MenuRow!	(#n →) Sets the menu row. #n is not the row, but the index of the first menu key in that row, i.e. 1,7,13,...
260BC	MenuRow@	(→ #n) Recalls the index of the first menu key in the current menu page. Returns 1 for the first page, 7 for the second page, 13 for the third and so on.

Addr.	Name	Description
260A8	LastMenuRow!	(#n →) Sets the row of the last menu. #n is not the row, but the index of the first menu key in that row, i.e. 1,7,13,...
260AD	LastMenuRow@	(→ #n) Recalls the index to the first menu key in the current row of the last menu. Returns 1 for the first page, 7 for the second page, 13 for the third and so on.
25845	MenuDef@	(→ menu) Recalls the current menu definition. menu is a MenuList or a program, or a Rompointer.
25908	LastMenuDef!	(menu →) Sets the definition of the last menu. menu is a MenuList or a program, or a Rompointer.
2590D	LastMenuDef@	(→ menu) Recalls the definition of the last menu. menu is a MenuList or a program, or a Rompointer.
25EFB	SaveLastMenu	(→) Stores row and definition of current menu as the last menu.
25EDA	GetMenu%	(→ %)
25863	MenuRowAct!	(ob →) Stores ob as the RowAct menu property.
25EE2	InitTrack:	(→) Execute the program which is next in the runstream if the directory changes. Used by the VAR menu to set first menurow when diretory changes, or by the CST menu to rebuild it.

Addr.	Name	Description

25877	`LabelDef!`	`( ob → )`
		Store a program which displays a menu label. Prg has the stack diagram
		`( #col ob → )`
		For example, the LIBS command uses the following program to make all menu label look like directories:
		`:: DUPNULL$?  ITE`
		`   MakeStdLabel MakeDirLabel`
		`   Grob>Menu ;`
		During execution, `INDEX@` will contain the menu key number.

2589F	`MenuKeyLS!`	`( ob → ob )`
		Set the action for left-shifted menu keys. The program receives the action part of the menu item as an argument, i.e.
		`{ob-NS ob-LS ob-RS}.`

258B3	`MenuKeyRS!`	`( ob → ob )`
		Set the action for right-shifted menu keys. The program receives the action part of the menu item as an argument, i.e.
		`{ob-NS ob-LS ob-RS}.`

2588B	`MenuKeyNS!`	`( og → ob )`
		Set the action for unshifted menu keys. The program receives the action part of the menu item as an argument, i.e. ob-NS or
		`{ob-NS ob-LS ob-RS}.`

25890	`MenuKeyNS@`	`( → ob )` Recall the action for unshifted menu keys.
25EFC	`SetKeysNS`	`( ob → )` Sets ob as MenuKeysNS, `DoBadKey` to LS & RS.
25F02	`StdMenuKeyLS`	`( {ob-NS ob-LS ob-RS} → ? )` The content of MenuKeyLS for standard menus.

25F03	`StdMenuKeyNS`	`( ob-NS → ? )`
		`( {ob-NS ob-LS ob-RS} → ? )`
		The content of MenuKeyNS for standard menus.

Addr.	Name	Description
27FED	`NullMenuKey`	($\rightarrow$) A placeholder for an empty menu key when defining menu lists.
258C7	`ReviewKey!`	(ob $\rightarrow$) Store a program which is called with the review key (RS DOWN). The program has the stack diagram ($\rightarrow$)
258EF	`(ExitAction!)`	(ob $\rightarrow$) Store ob as exit action.
25EEF	`NoExitAction`	($\rightarrow$) Sets `NOP` as ExitAction. Mostly used to avoid that the menu is saved as the previous menu when a new Menu gets installed.

37.3.2 Building Menus

Addr.	Name	Description
275C6	`TakeOver`	($\rightarrow$) Override the default menu key executer. If this is the first entry in a program, the program can be used in edit mode. When the first in a program in the label slot of a menu key, the program is evaluated to get the label object (most likely a grob).
275EE	`Modifier`	($\rightarrow$) `:: TakeOver ;`

27620	`MenuMaker`	($\rightarrow$ ob)
		Quotes next object, and also provides `TakeOver`. The disassembly is
		`:: TakeOver 'R ;`
		Normally this is used like this:
		`:: MenuMaker menu InitMenu ;`

25EE0	`InitMenu`	(menu $\rightarrow$) menu is {} or `::` settings {} ; Settings override the default settings installed by `InitMenu`.
25EC6	`DoMenuKey`	(menu $\rightarrow$) `:: SetDA12NoCh InitMenu ;`

Addr.	Name	Description
25EE1	InitMenu%	(%mnu.pg →) (%0 →)
25F00	StartMenu	(menu #n →) #n is the index of the first menu key on the page, use 1 for the first page, 7 for the second etc. StartMenu does ExitAction (Previous menu!), sets the default menu properties and page. Then it evaluates menu, stores result to MenuKeys and executes SetThisRow.
25EFE	SetThisRow	(→) Builds a new TOUCHTAB, SetBadMenu.
25EE8	LoadTouchTbl	(MenuKey1 .. MenuKeyN #n →) Builds new TOUCHTAB from menukeys.

37.3.3 Menu Display

Addr.	Name	Description
2EF66	SysMenuCheck	(→) Checks menu validity. If DA3NoCh? then nothing. If Track? then ?DoTrackAct@. If Rebuild? then SetThisRow.
2DFCC	?DispMenu	(→) Redisplay the menu now if no key is waiting in the buffer. Even better is this: `:: DA3OK?NOTIT ?DispMenu ;`
2DFF4	DispMenu.1	(→) Displays the menu immediately.
2DFE0	DispMenu	(→) `:: DispMenu.1 SetDAsValid ;`

37.3.4 Displaying Menu Labels

Addr.	Name	Description
2E0D5	Grob>Menu	(#col grob →) Displays grob as menu label.
2E0F3	Str>Menu	(#col $ →) Displays string as menu label.
2E11B	Id>Menu	(#col id →) Displays id as menu label.
2E107	Seco>Menu	(#col :: →) Does EVAL then DoLabel.
25886	DoLabel	(#col ob →) If ob is of one of the supported types, displays a menu label. If not, generates a "Bad Argument Type" error.
2E2AA	MakeLabel	($ #w #x grob → grob') Inserts $ into grob using CENTER$3x5 with y=5.
08E007	^WRITEMENU	($6...$1 →) Displays the six strings as menu keys.

37.3.5 General Entries

Addr.	Name	Description
25EA6	CheckMenuRow	(# → # #')
25EFD	SetSomeRow	(#n →) with Mod(n,FFFFFh)= 0.
2589A	DoMenuKeyNS	(#n →)
275FD	MenuKey	(→) Takes NOB from Runstream.
2F15B	CLEARMENU	(→)
25F2B	CHECKMENU	(→)
3EA01	(CST)	(→ ob) Evaluates ID CST.
2C2C0	nCustomMenu	(→) Installs the CST menu.
25EFF	SolvMenuInit	(→) Sets MenuKeyNS/LS/RS, ReviewKey and LabelDef properties needed by the Solver menu.

Addr.	Name	Description
25EC3	DoFirstRow	($\rightarrow$) Sets the first row of the current menu.

Chapter 38
Programming the HP49 Editor

The HP49G has a builtin editor which is much faster and nicer than the editor on the HP48. However, it is a general-purpose editor, and it would be useful for specific applications to add some features without having to write a whole new editor. The HP49G ROM contains a number of supported entry points which can be used to manipulate the editor from programs. These can be used to write editor extensions.

38.1 Terminology

The terms below will appear often in this chapter.

Term	Meaning
EditLine	The string which is currently being edited. Also called "Buffer" and "Command line". In the stack diagrams, we will use `$buf` for it.
Cursor position	The position of the cursor in the Editline. Represented by a bint. In stack diagrams, is written as `#cpos`.
Current line	The current line in the editor, i.e. the substring after the NEWLINE before the cursor up to the next NEWLINE character.
Editor window	When the text being edited is too long and/or wide, the screen of the HP49G shows only a part of the text: the window. When the cursor is moved, the window must be re-positioned to show the new position.
Selection	A region in the buffer can be selected when the begin marker and the end marker are active. The selected substring is called `$sel` in the stack diagrams.

Term	Meaning
Word-start	The beginning of a word, a position in a string where the char before is SPACE or NEWLINE, and the char after is a non-white character. Several commands deal with word-start positions, called `#ws` in the stack diagrams below.
Invisible chars	The HP49G can show text in different fonts and styles. In order to switch between fonts and styles, special markers are inserted into the text to indicates a change in font or style. These 3-character sequences are not visible, but they count in string length and in cursor position. Some Editor commands are aware of these strings and do complicated computations to cut and paste text with attributes. This of course makes these commands slower than they could be. If you do not use fonts and styles, you need not to worry about all this.

38.2 Examples

For information on the specific entries used in the examples below, consult the Reference section below.

1. Select the current line and copy it onto the clipboard.

```
::
  TakeOver
  CMD_END_LINE        (goto end of line)
  RCL_CMD_POS         (recall position)
  CMD_STO_FIN         (store as marker)
  CMD_DEB_LINE        (beginning of line)
  RCL_CMD_POS         (recall position)
  CMD_STO_DEBUT       (store as marker)
  CMD_COPY            (copy to clipboard)
;
```

This can be done shorter by using the builtin command `SELECT.LINE` command. The following is equivalent to the above.

```
::
  TakeOver
```

```
    SELECT.LINE
    CMD_COPY
  ;
```

2. Insert a ":: ;" template on a single line and position the cursor between "::" and ";".

```
  ::
    TakeOver
    "::  ;"
    CMD_PLUS
    CMD_BAK
    CMD_BAK
  ;
```

3. Insert a multi-line ":: ;" template and position the cursor with extra indentation on the second line.

```
  ::
    TakeOver
    "::\0A\0A;"
    CMD_PLUS
    CMD_UP
    SPACE$         (fix indentation to 2 extra spaces)
    CMD_PLUS
  ;
```

4. Go to next label. Labels are lines starting with "*".

```
  ::
    TakeOver
    "\0A*"             (newline followed by star)
    FindStrInCmd       (find that)
    IT
      ::               (if successful)
        DROP           (drop #end)
        #1+            (correct to move over NL)
        STO_CURS_POS   (set new cursor position)
      ;
    DROP               (drop the search string)
  ;
```

5. The RPLCPL command of the Emacs library (see section A.6) does completion of names in the Editor. It needs to find the word fragment before the cursor. Here is how this can be done:

```
::
  RCL_CMD        (recall EditLine)
  RCL_CMD_POS    (current position)
  DUP            (arg needed by GET.W<-)
  GET.W<-        (position of word start)
  #1+SWAP        (prepare args for SUB$)
  SUB$           (get the substring)
;
```

6. Change the indentation of the current line to #N spaces. #N is a bint expected on stack level 1. The command leaves empty lines and lines starting with a "*" alone.

```
::
  Blank$             (make the indentation str.)
  CMD_DEB_LINE       (goto beginning of line)
  RCL_CMD
  RCL_CMD_POS
  #1+ SUB$1          (look at first char in line)
  BINT42             (ASCII code of '*')
  OVER#=case         (line starts with '*'?)
    :: 2DROP         (cleanup,         )
       CMD_DOWN      ( next line & exit)
    ;
  BINT32 >#?SKIP     (line starts with nonwhite ch)
    ::
      CMD_END_LINE   (line starts with whitespace:)
      RCL_CMD_POS    (remember end of line position)
      CMD_DEB_LINE   (back to beginning of line)
      DO>Skip        (jump to next word)
      RCL_CMD_POS
      #<ITE
        DROPRDROP    (if already in next line: Exit)
        DoFarBS      (kill whitespc before 1st word)
    ;
  CMD_PLUS           (insert spaces)
  CMD_DEB_LINE       (back to beginning of line)
  CMD_DOWN           (next line)
;
```

38.3 Executing External Commands in the Editor

In order to use the new commands in the editor, you must bind them to a key or put them into a menu. Note that each command you write needs a `TakeOver` as the first entry in the secondary or the command will not execute in the editor.

Here is a simple example for an `InputLine` environment which defines an initial menu with two commands to select the current line and to clear the EditLine. For more information on `InputLine`, see Chapter 31.

```
::
  "Edit this!"        (prompt)
  ""                  (initial string)
  zero                (cursor position)
  zerozerozero        (modes)
  {
    {
      "SLINE"
      ::              (program to select line)
        TakeOver
        SELECT.LINE
      ;
    }
    {
      "CLEAR"
      ::              (program to clear EditLine)
        TakeOver
        DEL_CMD
      ;
    }
  }
  ONE                 (initial menu line)
  TRUE                (abort flag)
  ZERO                (parse)
  InputLine           (and GO!)
;
```

38.4 Reference

38.4.1 Status

Addr.	Name	Description
257A2	EditLExists?	(→ flag) Does an EditLine exist?
2EEED	NoEditLine?	(→ flag) Does no EditLine exist?
2F196	RCL_CMD	(→ $) Returns a copy of the current command line to the stack. Same as EDITLINE$.
2EEEB	EDITLINE$	(→ $) Returns a copy of the current command line to the stack. Same as RCL_CMD.
2F197	RCL_CMD2	(→ $) Similar to RCL_CMD, but if there is not enough memory to copy the EditLine to the stack, it will move the current EditLine into TEMPOB. Of course, this will delete the current EditLine.
2EF87	RCL_CMD_POS	(→ #) Recalls the current cursor position.
26585	CURSOR@	(→ #) Recalls the current cursor position.
26594	(CURSOR_PART)	(→ #) Recalls the current cursor row (line).
2F158	(THISCHAR)	(→ chr) Returns the character under the cursor. At the end of the file, returns CHR_00.
2EEEA	CURSOR_END?	(→ flag) Checks if the cursor is at the end of a line or at the end of the file. Works by checking the current character against newline and CHR_00.
264CC	FIRSTC@	(→ #) Column of the left display window edge.
26030	CURSOR_OFF	(→ #) Cursor column relative to left edge of display window.

Addr.	Name	Description
2EF91	CAL_CURS_POS	(#l #c → #) Computes a position in the current EditLine from line and column number. The result can be used by STO_CURS_POS to move the cursor to that location. If #line is larger than the number of lines in the EditLine, computes the position of the last line.
2EF90	CAL_CURS_POS_VIS	(#l #c → #) Similar to CAL_CURS_POS, but will ignore invisible characters. The result can be used by STO_CURS_POS_VIS to move the cursor to that location.
2F199	RCL_CMD_MODE	(→ $) Recalls a string with current editor settings. Can be used together with STO_CMD_MODE to save and restore the state of the EditLine, when temporarily leaving the editor with HALT or when calling a program which must temporarily change settings.
2F198	STO_CMD_MODE	($ →) Stores a mode string similar to the one obtained by RCL_CMD_MODE.

38.4.2 Inserting Text

Addr.	Name	Description
2EF74	CMD_PLUS	($ →) Inserts string at current cursor position in EditLine.
2F194	CMD_PLUS2	($ →) Replaces entire current EditLine with new string. When there is not enough memory to copy the string on stack level 1, moves the string out of TEMPOB. You must be careful that the string is not referenced in any way. The cursor is moved to the end of the new string.

Addr.	Name	Description
2F195	CMD_PLUS3	($ →) Same as CMD_PLUS2, but the cursor position is not changed. Useful when restoring a command line context after HALT.
2EF97	InsertEcho	($ →) Inserts string at current cursor position in Edit-Line.
2EEE4	Echo$Key	($/chr →) Same as CMD_PLUS.
2F11C	Echo$NoChr00	($ →) Inserts string at current cursor position in Edit-Line.
25EC1	DoDelim	(→) Takes a character or string from the runstream and inserts it.
25EC2	DoDelims	(→) Takes a character or a string from the run-stream, inserts it and moves the cursor back by one character.
25795	INSERT_MODE	(→) Turns insert mode on. In insert mode, new characters do not overwrite old ones.
2577F	(TogInsert)	(→) Toggles the insert/overwrite flag.
25790	INSERT?	(→ flag) Returns TRUE if insert mode is active.

38.4.3 Deleting Text

Addr.	Name	Description
2EF82	CMD_DEL	(→) Deletes next char in Editor. Same as LS+DEL. If you hold down BS while this entry is executed, the HP49G will think you have pressed the key and want to repeat it.

Addr.	Name	Description
2EF81	`CMD_DROP`	(→) Backspace in Editor. Deletes char before cursor. Same as BS key. If you hold down BS while this entry is executed, the HP49G will think you have pressed the key and want to repeat it.
2EF95	`DEL_CMD`	(→) Clears the entire EditLine.
2EEE7	`InitEdLine`	(→) `:: DEL_CMD ;`
2F2F0	`DO<Del`	(→) Deletes left to beginning of word. Same as the ←DEL button in the editor TOOL menu.
2F2F1	`DO>Del`	(→) Deletes right to beginning of next word, Same as the DEL→ button in the editor TOOL menu.
2F2F9	`DODEL.L`	(→) Deletes all chars in the current line. If the line is already empty, delete the NEWLINE. Same as the DEL.L button in the editor TOOL menu.
2F2DD	`DoFarBS`	(→) Deletes to beginning of line. Same as the RS+←DEL in the editor TOOL menu.
2F2DE	`DoFarDel`	(→) Deletes to end of line. Same as RS+Del→ in the editor TOOL menu.

38.4.4 Moving the Cursor

Addr.	Name	Description
2EF8B	STO_CURS_POS	(# →) Stores cursor position. Moves cursor to specified position and if necessary repositions the editor window to make sure the cursor position is visible. If it is necessary to scroll the window horizontally, this command sets the left edge of the window to the cursor column and shows as much text as possible to the right of the cursor. However, if the cursor is also visible when the window edge is moved to column zero, this position takes precedence.
2EF8C	STO_CURS_POS2	(# →) Same as STO_CURS_POS, but moves the right edge of the editor window to the cursor column.
2EF8D	STO_CURS_POS3	(# →) Same as STO_CURS_POS, but without checking for style/font switch sequences. So while STO_CURS_POS always makes sure the cursor ends up right before a visible character, this command allows you to position it within the invisible escape sequences.
2EF8E	STO_CURS_POS4	(# →) Behaves with respect to editor window positioning like STO_CURS_POS2, but with respect to invisible chars like STO_CURS_POS3.
2EF8F	STO_CURS_POS_VIS	(# →) Like STO_CURS_POS, but ignores the invisible characters. So if you look at your string and say, I want to go to what I see as the 5th character, use this entry.
2F378	SetCursor	(# →) ({# #'} →) Sets the cursor to the given position. For the list argument, the numbers are row and column.

Addr.	Name	Description
2EF7C	CMD_NXT	(→) Moves cursor to next char, like Right Arrow.
2EF7B	CMD_BAK	(→) Moves cursor to the left. Same as as Left Arrow.
2EF80	CMD_DOWN	(→) Moves cursor to the next line. Same as Down Arrow.
2EF7F	CMD_UP	(→) Moves cursor to the previous line, like Up Arrow.
2EF7D	CMD_DEB_LINE	(→) Moves cursor to the beginning of line. Same as RS+LEFT.
2EF7E	CMD_END_LINE	(→) Moves cursor to the end of line. Same as RS+RIGHT.
2EF7A	CMD_PAGED	(→) Moves cursor one page down, like LS+DOWN.
2EF77	CMD_PAGEL	(→) Moves cursor one page left, like LS+LEFT.
2EF78	CMD_PAGER	(→) Moves cursor one page right, like LS+RIGHT.
2EF79	CMD_PAGEU	(→) Moves cursor one page up, like LS+UP.
2F2EE	DO<Skip	(→) Skips left to beginning of word. Same as the ←SKIP button in the editor TOOL menu.
2F2EF	DO>Skip	(→) Skips right to the beginning of the next word. Same as the SKIP→ button in the editor TOOL menu.
2F2E4	DO>BEG	(→) Goes to begin of selection (if active) or to beginning of EditLine. Same as →BEG button in the editor TOOL menu.

Addr.	Name	Description
2F2E5	DO>END	(→) Goes to end of selection. Same as the →END button in the editor TOOL menu. When there is no selection, does not move.
2F2E6	GOTOLABEL	(→) Brings up the CHOOSE-box with labels in the EditLine. Same as the LABEL button in the editor TOOL/GOTO menu.

38.4.5 Selection, Cut and Paste, the Clipboard

Addr.	Name	Description
2EF83	CMD_STO_DEBUT	(# →) Sets begin marker, like RS+BEGIN, but takes position from stack.
2EF84	CMD_STO_FIN	(# →) Sets end marker, like RS+END, but takes position from stack.
2EF85	RCL_CMD_DEB	(→ #) (→ #0) Recalls the position of the BEGIN marker. If the selection has been cleared, returns ZERO.
2EF86	RCL_CMD_FIN	(→ #) (→ #0) Recalls the position of the END marker. If the selection has been cleared, returns ZERO.
2F2DC	ClearSelection	(→) Unselects the selected text without changing the contents of the editor. Sets both begin and end marker to ZERO.
2EF93	VERIF_SELECTION	(→ flag) Returns TRUE when the END marker is not ZERO, indicating that the selection is active. Use this command as a check before doing anything with the selection.
2EF8A	CMD_COPY	(→) Copies selected string, like RS+COPY.

Addr.	Name	Description
2EF88	CMD_CUT	(→) Cuts string. Really is "delete", does not copy to kill buffer. So a "normal" CUT would be `:: CMD_COPY CMD_CUT ;`

2F2FA	CMD_COPY.SBR	(→ $)
		Puts the selection as a string on the stack. This command is font/style aware. It is recommended not to use it because it may get the wrong text style if the cursor is not repositioned to the beginning of the selection first. If you don't use fonts,
		`:: RCL_CMD`
		`    RCL_CMD_DEB RCL_CMD_FIN`
		`    SUB$ ;`
		does something similar.

2EF94	PASTE.EXT	($ →) Pastes from stack with treatment of fonts and styles. Inserts the string on stack level at the cursor position. It can insert normal text right in the middle of bold test etc. If you don't use styles or different fonts, CMD_PLUS is probably faster.
2F2E1	SELECT.LINE	(→) Selects current line, position cursor at beginning of line. Selection does not include the NEWLINE char at the end of the line.
2F2E2	SELECT.LINEEND	(→) Selects current line, position cursor at end of line. Selection does not include the NEWLINE char at the end of the line.
2A085	(Clipboard!)	($ →) Stores string to Clipboard.
2A095	(Clipboard@)	(→ $) Recalls Clipboard contents to stack.
2A0A5	(Clipboard0)	(→) Clears the Clipboard.
2A0B5	(Clipboard?)	(→ flag) Is there anything on the Clipboard?

38.4.6 Search and Replace

Addr.	Name	Description
2F2F3	GET.W->	(→ #) Returns the position of the next word-start to the right of the current cursor position. Note the asymmetry of this command and GET.W<-.
2F2F4	GET.W<-	(# → #') Takes a position from the stack and return the position if the nearest word-start to the left of that position. Note the asymmetry of this command and GET.W->.
2F2F2	FindStrInCmd	($find → $find $start $end T) ($find → $find F) Finds a string in the EditLine, starting from the current cursor position. The search string remains on the stack, presumably in order to do repeated searches. Returns the start and end positions of the match and a flag. This function respects the setting of the internal flag for case-sensitive search.
2F2E8	DOFIND	(→) Same as the FIND menu button in the editor TOOL/SEARCH menu. Pops up the FIND input form.
2F2EA	DONEXT	(→) Finds next. Same as the NEXT button in the editor TOOL/SEARCH menu.
2F2E9	DOREPL	(→) Same as the REP button in the editor TOOL/SEARCH menu. Pops up the REPLACE input form.
2F2EB	DOREPLACE	(→) Replaces current match. Same as the R button in the editor TOOL/SEARCH menu.

Addr.	Name	Description
2F2EC	DOREPLACE/NEXT	($\rightarrow$) Replaces current match and move to next match. Same as the R/N button in the editor TOOL/SEARCH menu.
2F2ED	REPLACEALL	($\rightarrow$) Replaces all matches in buffer. Same as the ALL button in the editor TOOL/SEARCH menu.
2F2FC	REPLACEALLNOSCREEN	($\rightarrow$) Like `REPLACEALL`, but does not update the screen. Much faster this way.

38.4.7 Evaluation

Addr.	Name	Description
2F2DF	EditSelect	($\rightarrow$) Edits the current selection. Opens the editor with the selection only. You can then edit the selection. After pressing ENTER the edited text is inserted back into the previous editing environment.
2F2E3	EVAL.LINE	($\rightarrow$) Evaluates the current line and replace it with the result of the evaluation. Similar to `EVAL.SELECTION`, but without the need to select the line first.
2F2FB	EVAL.SELECTION	($\rightarrow$) Evaluates the current selection and replace it with the result of the evaluation. Same as the EXEC button in the editor TOOL menu.

Addr.	Name	Description
2F2F8	EXEC_CMD	(cmd algflag → obsel) Runs a command on the selection in the Editline. Takes two arguments: the command to run and a flag which says how to compile the selection before the command is applied. If the flag is TRUE, and ALG mode in on, the ALG compiler is used and the DOTAG :: xEVAL prologue of the result is removed. Use this if the result is to be edited by another editor. The selection is left on stack level 1 as an object.
0B954	(RunInNewContext)	(ob →) Saves current user interface, evaluate ob and restore the user interface. Can be used to run applications from inside another application.

38.4.8 Starting the Editor

Addr.	Name	Description
2EEE9	EditString	($ →) Starts editing the string when the current program exits. This is the entry to use if a program should exit with the editor activated. Use InitEdLine before this entry to clear the editline (if desired) - if not, the string is inserted into the current editline. All code after this entry will be executed *before* control is handed to the editor application. For example: `:: "SOME STRING" DUPLEN$ SWAP (get length) InitEdLine (clear the editline) EditString (string to editline) STO_CURS_POS2 (cursor at end) "Starting editor..." FlashMsg (display before edit) ;`

Addr.	Name	Description
2F19A	ViewLevel1	(ob → ob') Edits the object in level 1
2F1AF	AlgObEdit	(ob → ob') Used instead of ViewLevel1 if in Algebraic mode. Does not execute STARTED and EXITED.
2B2F2	(CallEditCmd:)	(ob → ob') Evaluates the next object in the runstream, which usually in an editing command like ObEdit. When the evaluation returns FALSE, the original object which was saved in a temporary variable is restored to the stack. When the evaluation returns TRUE, the TRUE is removed from the stack.
2EEE5	EditLevel1	(ob → ob')
2F1AE	ObEdit	(ob → ob' T) (ob → F) Edits object. When the user cancels, only FALSE is returned. Otherwise the changed object along with TRUE is returned.
011004	ˆEQW3Edit	(symb → symb' T) (symb → F) Opens the equation editor to edit the expression. If exited by ENTER, returns new expression and TRUE. If exited by CANCEL, returns just FALSE.

38.4.9 Miscellaneous

Addr.	Name	Description
25ED2	EditMenu	(→ {}) Returns the Editor menu.
2EF73	?Space/Go>	(→) Inserts a SPACE character unless there is already one before the cursor position. Use this if you want to make sure the next stuff echoed is separated by at least one space from the word preceding it.

Addr.	**Name**	**Description**

2EF76	AddLeadingSpace	($ → $')
		Adds a leading space to the string on level1 if it does not start with a space *and* if the cursor in the editor is after a non-white character. So
		`:: "DUP" AddLeadingSpace`
		`   AddTrailingSpace CMD_PLUS ;`
		inserts DUP and makes sure it will be surrounded by spaces.

2EF75	AddTrailingSpace	($ → $') Adds a trailing space to the string on level1 unless the string already ends with a space.
2EF9A	CommandLineHeight	(→ #pix) Returns the number pixel rows occupied by visible part of the EditLine.
2F2DB	DOTEXTINFO	(→) Displays the info screen about the Editline. Same as the INFO button in the editor TOOL menu.
2F2F6	GET_CUR_FONT.EXT	(→ #) Returns the ID (as a system binary) of the font used for the character under the cursor.
2EF96	NO_AFFCMD	(→) Tells the next CMD_PLUS call not to update the display. For speed, if you want to do more insertion before the user needs to see it.
2F19E	DispCommandLine	(→) Redisplays the command line.
2F19F	?DispCommandLine	(→) Redisplays the command line if necessary.

Addr.	Name	Description

2F2F7	PUT_STYLE	(# →)
		Changes the style at point. If the selection is active, changes the style of the text in the selection. Otherwise changes the style of text typed subsequently. Takes a BINT from the stack which is the number of the style. In think the ITALI button in the editor TOOL/STYLE menu could be implemented with the following program:
		`:: ERRSET PUT_STYLE`
		`   ERRTRAP ERRJMP ;`
		`PUT_STYLE` does not `ABND` its temporary environment, so you need the `ERRTRAP` construction to work around this bug.

2F2F5	PUT_FONTE	(# →) Changes the font at point. Works similar to the `PUT_STYLE` command.
2F2E7	SELECT.FONT	(→) Pops up the CHOOSE box to select a font. Same as the FONT button in the editor TOOL/STYLE menu.
2F2E0	ViewEditGrob	(→) at cursor Views the grob currently edited in the Editline near the cursor. If the EditLine contains `GROB 10 10 FFFFFF...` move the cursor to the "1" of the first "10". Then this entry point will display the grob.

Addr.	Name	Description
2EF92	XLINE_SIZE?	(ob → flag) Checks if the cursor is outside the current line. In the HP49G editor, you can move the cursor further to the right than the line length, without actually making the line longer. The line gets extended only if you actually insert text or use CMD_DEL to catch to following line to the position. This entry returns TRUE if it is not on or before the newline. Note that it takes an arbitrary object from the stack first - so put something there before calling it.
27F47	<DelKey	(→ {}) Returns the ←DEL menu key.
27F9A	>DelKey	(→ {}) Returns the DEL→ menu key.
27EAF	<SkipKey	(→ {}) Returns the ←SKIP menu key.
27EFB	>SkipKey	(→ {}) Returns the SKIP→ menu key.
2EEE6	InitEd&Modes	(→) :: InitEdLine InitEdModes ;
2EEE7	InitEdLine	(→) :: DEL_CMD ;
2EEE8	InitEdModes	(→)
2F05E	SaveLastEdit	($ →) Calls CMD_STO if history is on.
2F326	CMDSTO	($ →) Adds string to the list of the last 4 commands, accessible with the CMD key.

Chapter 39
Plotting

The commands in this chapter deal with aspects related to plotting. Entries here deal primarily with the PPAR variable, that contains the parameters used in plotting. This variable is a list with the following parameters:

{ $(x_{\min}, y_{\min})$ $(x_{\max}, y_{\max})$ *indep res axes type depend*}

This is the meaning of each of the parameters:

Parameter	**Description**	**Default value**
$(x_{\min}, y_{\min})$	A complex number representing the coordinates of the lower left viewing range.	$(-6.5, -3.1)$
$(x_{\max}, y_{\max})$	A complex number representing the coordinates of the upper right viewing range.	$(6.5, 3.2)$
indep	The independent variable.	X
res	Resolution. A number that represents the interval between the plotted points.	0
axes	A complex number that represents the coordinates of the intersection of the axes. It can also be a list representing this coordinate and many other details, which are not described in this book.	$(0, 0)$
type	The name of the (user) command that specifies the plot type.	`FUNCTION`
depend	Dependent variable.	Y

39.1 Reference

Addr.	Name	Description
2F162	CHECKPICT	(→) Checks size of GBUFF. If it is smaller than 131x64 sets GBUFF back to its default size (131x64).
2EF06	CKPICT	(xPICT →) Checks for user word xPICT on level 1. Errors (SETTYPEERR) if there is another object.
2F258	PICTRCL	(xPICT → grob) Does CKPICT, then recalls GBUFF and does TOTEMPOB.
2F355	MAKEPVARS	(→ {}) Creates the default PPAR variable in the current directory and returns its value.
2F163	CHECKPVARS	(→ {}) Recalls contents of PPAR in current path to stack. Creates PPAR in current directory if non-existent. Errors "Invalid PPAR" if existing PPAR is invalid.
2F33D	GETPARAM	(# → ob) Extracts the #th item from PPAR. No error checking!
2F0FF	GETXMIN	(→ %) Recalls XMIN from the PPAR list if existent. If not, the default PPAR is created in the current directory.
2F366	PUTXMIN	(% →) Sets a new value for XMIN. PPAR is created if necessary.
2F0FE	GETXMAX	(→ %) Recalls XMAX from the PPAR list if existent. If not, the default PPAR is created in the current directory.
2F365	PUTXMAX	(% →) Sets a new value for XMAX. PPAR is created if necessary.

Addr.	Name	Description
2F100	GETYMIN	(→ %) Recalls YMIN from the PPAR list if existent. If not, the default PPAR is created in the current directory.
2F368	PUTYMIN	(% →) Sets a new value for YMIN. PPAR is created if necessary.
2F10E	GETYMAX	(→ %) Recalls YMAX from the PPAR list if existent. If not, the default PPAR is created in the current directory.
2F367	PUTYMAX	(% →) Sets a new value for YMAX. PPAR is created if necessary.
2F107	GETPMIN&MAX	(→ C% C%) Returns PMIN and PMAX.
2EEF2	PUTINDEP	(ID →) Internal xINDEP if the arg is an ID.
2EEF3	PUTINDEPLIST	({} →) Internal xINDEP if the arg is a list.
2F0E8	INDEPVAR	(→ id) Recalls the independent variable. If a list, extract first element. :: GETINDEP DUPTYPELIST? ?CARCOMP ;
2F106	GETINDEP	(→ id) (→ {}) Recalls the independent variable field in PPAR.
2EEF5	GETPTYPE	(→ name) Recalls the plot type using GETPARAM.
2EEF6	PUTPTYPE	(name →) Sets a new plot type. PPAR is created if necessary.
2F10D	GETRES	(→ %) Recalls the plot resolution using GETPARAM.
2EEF4	PUTRES	(% →) Set new plot resolution. PPAR is created if necessary.
2F33E	GETSCALE	(→ % %') Recalls the plot scale parameters.

Addr.	Name	Description
2EEF1	PUTSCALE	(% %' →) Set new plot scale. PPAR is created if necessary.
2EEEF	AUTOSCALE	(→) Internal AUTO.
2EF60	DOGRAPHIC	(→) Sets the scroll mode of PICTURE and is essentially the same as { } PVIEW.
25ECF	EQUATION	(→ ob) Recall the current equation, stored in the 'EQ' variable.
2F339	GetEqN	(#n → ob T) (#n → NULL$ F) Get the #nth equation, if EQ is a list of equations.
25EB5	DORCLE	(→ ob) Recalls the contents of the EQ variable, errors if it does not exist.
25EB6	DOSTOE	(ob →) Stores ob into the variable EQ.
2F297	XEQPURGEPICT	(xPICT →) If object in level one is xPICT, erases the graphic display. Otherwise, errors.
2F105	GDISPCENTER	(→) Moves to center of graphics display
2EF01	DOPX>C	({ hxs hxs' } → C%) Converts a list of two hex strings into a complex number. Used for plotting coordinates. Inverse operation is DOC>PX.
2EF02	DOC>PX	(C% → { hxs hxs' }) Converts a complex coordinate point into list of two HXS numbers. Inverse operation is DOPX>C.

Part IV

The HP49 CAS

Chapter 40
Introduction to the HP49 CAS

One of the major innovations in the HP49G is the powerful built-in Computer Algebra System (CAS). The HP49G CAS is derived mainly from the ALG48 and ERABLE libraries, originally written for the HP48 calculators. But on the HP49G, the CAS is fully integrated into the operating system, so that User RPL operators transparently access the CAS if the arguments require it. A huge number of supported entry points give access to the internal commands of the CAS, enabling users to write their own programs and commands dealing with symbolic objects, matrices and infinite precision integers.

40.1 Problems with These Chapters

The initial version of the reference listing of CAS commands for this book was derived from the source files[1] of ALG48 and ERABLE. The problem with this approach is that in the source, the different entries are not fully ordered according to functionality. Rather, each source file handles a certain area of CAS commands, and many utility routines are included inside the same file. For this reason there are several different locations where for example meta-object handling routines may be found. There are even similar such routines (which seem to do the same thing) in different files. We have made a significant effort to reorder the entries by functionality, but we realize that we have only partially succeeded. A deeper knowledge of the CAS and its internals is needed to complete this work.

A full documentation of the CAS should also contain extensive material about the internal representation of CAS objects, and many examples how to use these commands. Let us hope that Bernard Parisse will one day find time to fully document the HP49G CAS internals. For the time being we include a slightly edited version of a document he provided to us, which introduces some important aspects of the CAS. The reminder of this part will then just be a reference list of entries.

[1] actually, from a condensed version of the routine headers provided to us by Bernard Parisse

40.2 Symbolic Objects

The CAS manipulates symbolic scalars and vectors or matrices of these objects. Symbolic scalars have 3 representations, which we show in the following table using '2X' as an example expression.

user representation	a SYMBOL object which is the composite object. For the example expression it looks like this: `SYMBOL Z2 ID X x* ;`
meta representation	the SYMBOL object exploded onto the stack. For `'2*X'`, these are the 4 objects `Z2 ID X x*` #4 on stack levels 4 to 1.
list representation	polynomial coefficients (in the example: `{ 2 0 }`) with respect to the list of variables (`{ X }`).

Conversion from user to meta representation is done by `SYMBINCOMP` (a generalized `INNERCOMP` to handle non symbolic objects like integers).

Meta representation is used to handle operations when rational normal form is not relevant. It is more efficient than symbolic representation because you do not have to explode and rebuild the symbolic objects, everything is done on the stack. Stack operations on metas are described in Chapter 12. Unary and binary operators are often the operator name prefixed by `addt` (e.g. `addtSIN`). An example for a complex routine working on meta objects is `CASCOMPEVAL`. It does a `COMPEVAL`-like loop but with metas on the stack instead of symbolics.

The list representation is used when the rational normal form is important. This is the case for integration of rational fractions, rational simplifications, Laplace transformations, series expansions and similar operations. The first step for the conversion is to find the *list of variables* with respect to which the expression is rational. For example,

$$\frac{\sin(x)+y}{\cos(x)+y} \tag{40.1}$$

is rational with respect to `{ sin(x) cos(x) y }`. Given a symbolic or a list/array of symbolic objects, the user word `LVAR`, or the System RPL command `LVARext`, returns this list of variables. The conversion is then done as a quotient of 2 multivariate polynomials with respect to this list of variables, with this ordering.

Gaussian integers are represented as secondaries with two elements:

`:: imaginary_part real_part ;`. The imaginary and real parts must be integers.

Square roots are represented as *irrquad*: `:: x« a b c x» ;` represents `a+b*sqrt(c)`.

Polynomials are defined as a list of coefficients that are polynomials themselves, constants (integer or Gaussian integer) or irrquads. *Rational fractions* built over these polynomials are represented as `SYMBOL num deno x/ ;` where `num` and `deno` are polynomials that are prime together (in exact mode).

The main conversion routine to the list format is `VXXLext`. The main back conversion routine is `R2SYM`. There are several specialized routines to convert a list or meta of symbolic objects, or to convert a symbolic object into meta-representation, or from list format to the meta-representation of a symbolic object. These specialized routines are more efficient but more difficult to use.

Rational operators on list objects are implemented (`QAdd`, `QSub`, `QDiv`, `QMul`, `QNeg`, `RPext`), as well as Euclidean divisions with specializations e.g. for integers or Gaussian integers.

40.3 A Few Examples

In the following examples, the comments in each line represent the objects on the stack after the current command.

Rational simplification of a symbolic object might be coded as

```
::                      (symb)
  FPTR2 ^LVARext        (symb lvar)
  FPTR2 ^VXXLext        (lvar n/d)
  FPTR2 ^R2SYM          (symb)
;
```

The scalar product of 2 symbolic vectors in “list form”

```
::                            (x y)
  INNERCOMP #1+ROLL INNERCOMP
  DUP#1= casedrop FPTR2 ^QMul
  get1                        (y1,...,yn-1,x1,...,xn,#n,yn)
  ROTSWAP PTR2 ^QMul          (y1,...,yn-1,x1,...,xn-1,#n,xn*yn)
  OVER ONE_DO
   ROT 3PICK #2+PICK
```

```
     FPTR2 ^QMul
     FPTR2 ^QAdd
    LOOP
                                (y1, ..., yn-1, #n, X.Y)
    OVER #1+UNROLL #1- NDROP
  ;
```

Chapter 41
Type Checking and Conversion

The entries in this chapter are used to check for the special CAS objects described in Chapter 40, and to convert between this different kinds of objects.

41.1 Reference

Addr.	Name	Description

157006	`^SYMBINCOMP`	`( symb → ob1 .. obN #n )`
		`( ob → ob #1 )`
		`( {} → {} #1 )`
		Explodes symbolic object into meta. Other objects are converted into one-object metas by pushing #1 into the stack.

12A006	`^2SYMBINCOMP`	`( ob1 ob2 → meta1 meta2 )` Does `^SYMBINCOMP` for 2 objects.
4D7006	`^VXXLext`	`( ob Lvar → Q )` Converts object to internal form. The object can be a symbolic, a symbolic vector or a symbolic matrix. If the conversion was not successfull, vxxxlflag is cleared.
400006	`^R2SYM`	`( lvar ob → ob )` Back conversion of a scalar object.
4D8006	`^METALISTVXXL`	`( Meta → Meta )` Conversion of all elements of a meta object with respect to the variables in LAM1.
4D9006	`^VXXLFext`	`( n/d → Z1/Z2 )` Conversion of a fraction which does not depend on any variables.

Addr.	Name	Description
4DA006	^VXXL1ext	(n → Z) Conversion of an object which does not depend on any variables.
4DB006	^VXXL0	(ob → Q) Conversion of object with respect to Lvar in LAM1.
4DC006	^VXXL2NR	(Meta → Q) Converts symbolic meta to internal form (LAM1=Lvar). Set nocareflag to avoid square root problems.
4DD006	^VXXL2	(Meta → Q) Converts symbolic meta to internal form (LAM1=Lvar).
167006	^TYPEIRRQ?	(ob → flag) Is ob an irrquad?
168006	^DTYPEIRRQ?	(ob → ob flag) DUP, then ^TYPEIRRQ?.
177006	^CKMATRIXELEM	(ob → ob) Checks that ob is a valid internal matrix element. Look for CK[]NCK for user matrix element.
18F006	^CKFPOLYext	(ob → ob) Errors if list contains secondaries or empty lists.
190006	^CK2FPOLY	(ob ob → ob ob) Does CKFPOLYext on two objects.
19E006	^CLEANIDLAM	(ob → ob) Suppresses SYMB if not needed.

Chapter 42
Integers

This chapter lists the functions that deal with Arbitrary Precision Integers, a new number type provided by the HP49 CAS. For a description of that type, see Chapter 5.

You will notice that there are no entries for basic arithmetic operations on integers. This is because there are no specific such entries for integers. Instead, use the polynomial entries like ^QAdd, ^QMul, etc. listed in Chapter 46.

42.1 Reference

42.1.1 Built-in Integers

Addr.	Name	Description
2E0006	^DROPZ0	(ob → z0)
2DF006	^DROPZ1	(ob → z1)
392006	^2DROPZ0	(2 1 → z0)
3B3006	^NDROPZ0	(obn...ob1 #n → z0) Replaces meta with Z0.
3B4006	^NDROPZ1	(obn...ob1 #n → z1) Replaces meta with Z1.

42.1.2 Conversion Functions

Addr.	Name	Description
0EE006	^#>Z	(# → Z) Converts bint to zint.
0F5006	^R>Z	(% → z) Converts real to zint. Do not call this entry if the number if not an integer.

Addr.	Name	Description
18D006	^R2Zext	(% → %%/Z) Converts real to zint, or to long real if the number is not an integer. mode if number is not an integer.
0ED006	^H>Z	(HXS → Z / Error) Checks if HXS is a proper zint number and trims it.
0F2006	^S>Z	($ → z) Converts decimal in a string into a zint.
0F3006	^S>Z?	($ → z T) ($ → $ F) If possible, converts string into a zint and returns TRUE. If not, keeps the original string and returns FALSE.
184006	^CK1Z	($/#/hxs → Z) Checks for an integer. Converts strings, bints or hxs's to zints. Errors for other object types.
185006	^CK2Z	(ob ob' → Z Z') Like ^CK1Z, but for two objects.
186006	^CK3Z	(ob ob' ob'' → Z Z' Z'') Like ^CK1Z, but for three objects.
202006	^CK&CONVINT	(symb → zint) (symb → :: zint zint' ;) Check that a sym is a zint or Gauss integer, convert it.
203006	^CK&CONV2INT	(symb symb' → zint zint') (symb symb' → :: zint1 zint2 ; :: zint3 zint4 ;) Check that 2 sym are zint or Gauss integer, convert them.
205006	^CONVBACKINT	(zint|c → symb)
204006	^CONVBACK2INT	(zint|c zint|c → symb symb)
0F4006	^Z>ZH	(Z → Z') Converts decimal Z to hex Z.
18E006	^Z2Sext	(Z → '$Z') Converts Z to string number. The number is embedded in a symbolic to enable using it in algebraics.

42.1.3 General Integer Operations

Addr.	Name	Description
101006	^ZTrim	(Z → Z') Strips Z from unnecessary leading nibbles. Counts nibbles required for representation. If that equals used nibbles then quick exit. Else allocates new object, copies significant mantissa nibbles and apends original sign.
102006	^ZAbs	(Z → |Z|) Takes the absolute value of Z. If Z is already positive then does nothing. Else duplicate object and change sign.
50B006	^ZABS	(Z → Z') Absolute value.
0E0006	^ZSQRT	(Z → Z' flag) Calculates integer part of square root. If the number was a square, then flag is TRUE to indicate that the returned result is exact.
3D0006	^Mod	(Z Zn → Z') Make Z modulo N.
0DD006	^ZMod	(Z1 Z2 → Z')
105006	^ZNMax	(Z1 Z2 → NormMax[Z1,Z2]) Returns the integer with the greatest absolute value. (Returns Z1 if |Z1|≥|Z2|; returns Z2 if |Z1|<|Z2|).
106006	^ZNMin	(Z1 Z2 → NormMin[Z1,Z2]) Returns the integer with the smallest absolute value. (Returns Z1 if |Z1|≤|Z2|; returns Z2 if |Z1|>|Z2|).
10D006	^ZBits	(Z → Z #bits) Calculates number of bits used in Z.
10E006	^ZBit?	(Z #bit → Z flag) Tests if a bit in Z is set. Count starts from zero, as opposed to ZBits.
2B7006	^ZGCDext	(Z2 Z1 → Z) Integer GCD.
2B8006	^ZGcd	(Z2 Z1 → Z) This is the same entry as ZGCDext.

Addr.	Name	Description
3D6006	^IEGCDext	(a b → d u v) Bezout for integers. d=au+bv=gcd(a,b).
3D9006	^INEGCD	(a b → d u v)
07C007	^#FACT	(# → Z) Calculates the factorial of an integer. Works fine for all numbers #0 - #FFFFF, although at some point you will get an out of memory error.
576006	^factzint	(z → z!) Factorial for long integers.
215006	^PA2B2	(z/% → a+bi) Internal PA2B2.

42.1.4 Integer Factorization and Prime Numbers

Addr.	Name	Description
0C9006	^ZFactor	(Zs → Lf) Factors signed long integer.
0CA006	^NFactor	(z → {}) Factors positive long integer.
0CB006	^NFactorSpc	(z → {}) Semi-factors positive long integer. This is regular factorization with an extra 'hopeless?' test.
0CD006	^SFactor	(S → Lf) Factors short integer. Pollard Rho, with the assumption that trial division has been done already. Thus any factor less than 4012009 is known to be a prime, for greater factors a primality test is used before calling the actual Pollard Rho. Pollard Rho does not find the factors in order of magnitude, thus the results will be sorted after full factorization has been achieved.

Addr.	Name	Description
0CE006	^SPollard	(S → S1 S2) Factors short integer into 2 parts using Pollard Rho algorithm. Trial division and primality tests should be done prior to calling this subroutine, otherwise an eternal loop is risked. The random number generator is modeled after the user level RAND command, although the starting value is different.
0CF006	^BFactor	(N → Lf) Factors long integer. Brent-Pollard, with the assumption that trial division has been done already. When a small factor is found SFactor is called to get full short factorization. Since the factorization can potentially take a very long time, an execution time test is used to abort factoring very long integers (limit is 60s for each composite). The factors are sorted at exit.
0D0006	^BrentPow	(Za Z1 Z2 Zn #k → Z) Modular * + ^ mod for Brent-Pollard factorization. Output is Z1*Z2+Za mod Zn repeated k times Note that k=0 and k=1 give the same result. Also Z1≠Z2 makes no sense for k≠0. All arguments are assumed to be positive. Za is assumed to be < 16. In some instances k can be a very high number, thus it might make sense to use Montgomery multiplication.
0D1006	^ZPrime?	(Z → flag) Primality test for a positive integer. According to Pinch commercial software packages use only about 5-10 bases by default, maximum around 25. The latest versions usually implement a deterministic.
0D2006	^ZIsPrime?	(Z → flag) Probabilistic primality test for a positive integer.

Addr.	Name	Description
0D3006	ˆSIsPrime?	(S → flag) Tests if positive short Z is prime. M-R test fails for integers ≤ 3, so we just test them separately at the start. For convenience lets define 0 and 1 to be primes also.
0D4006	ˆBIsPrime?	(S → flag) Test if positive long Z is prime.
0D5006	ˆBRabin	(Z #base → Z flag) Performs Miller-Rabin test for long positive integer. Returns TRUE if base witnesses composite. Else returns FALSE.
0D6006	ˆZTrialDiv2	(Z → Z' #n) Remove factors of 2 from integer. #n is the power of two extracted from the number. The sign is also handled correctly, even though it is never required in ALG48 (absolute Z).
0D7006	ˆZTrialPrime?	(Z → flag) Trial division primality test for a positive integer. works for $Z \geq 3$ (return false for Z=2).
0D8006	ˆZTrialDiv	(Z → Mf Z') Trial division of a positive integer. If Z' is one then full factorization was achieved. The long trial division is not too slow, since division by short integer is quite fast. The quotient is also checked so that a final factor less than 2000ˆ2 will also be automatically detected.
0C7006	ˆPrime+	(Z → Z') Returns next prime (Z' > Z).
0C8006	ˆPrime-	(Z → Z') Returns previous prime (Z' < Z).

42.1.5 Gaussian Integers

Addr.	Name	Description
114007	^TYPEGAUSSINT?	(ob → flag) Checks if ob is Gaussian integer.
115007	^DTYPEGAUSSINT?	(ob → ob flag) Checks if ob is Gaussian integer.
116007	^DUPTYPEGAUSSINT?	(ob → ob flag) Checks if ob is Gaussian integer.
187006	^CK1Cext	(ob → flag) Checks if object is integer or Gaussian integer.
15D006	^CXRIext	(C → Zre Zim) Returns real and imaginary part of Gaussian integer.
2B5006	^CGCDext	(C2 C1 → C) GCD for Gauss integers.
4D5006	^CSQFFext	(C → { factor1 mult1 ... factn multn }) Factorization of Gauss integers. This is not the complete factorization of C over Gauss integers since the GCD of the real part and imaginary part of c is factored only over R.
4D4006	^SECOSQFFext	(:: x<< a b c x>> → { fact1 mult1 ... factn multn }) Factorization of irrquads and Gauss integers.
4D6006	^SUMSQRext	(Z → Z C) Returns a Gauss integer C so that |C|^2=Z. Z must be 2 or so that Z=1 mod 4. If Z $\neq$ 1 mod 4, "Z is not 1 mod 4" error. Z should be prime to ensure the existence of a solution.
518006	^CNORMext	(C → |C|^2) Square modulus of a Gauss integer.

42.1.6 Integer Tests

Addr.	Name	Description
265C1	Z=	(Z Z' → flag)
265C6	Z<>	(Z Z' → flag)
265BC	Z<	(Z Z' → flag)
265D0	Z<=	(Z Z' → flag)
265B7	Z>	(Z Z' → flag)
265CB	Z>=	(Z Z' → flag)
0F8006	^QIsZero?	(Q → flag) Tests if Q is zero. Assumes list contains only lists or hexes!.
0F7006	^DupQIsZero?	(Q → Q flag) Duplicates Q and tests if Q is zero. Assumes list contains only lists or hexes!.
0FA006	^ZIsOne?	(Z → flag) Tests if Z is Z1.
0F9006	^DupZIsOne?	(Z → Z flag) Duplicates Z, and returns TRUE if Z is 1.
109006	^DupZIsTwo?	(Z → Z flag) Returns TRUE if Z is 2.
0FC006	^ZIsNeg?	(Z → flag) Tests if Z is negative.
0FB006	^DupZIsNeg?	(Z → Z flag) Tests if Z is negative.
10A006	^DupZIsEven?	(Z → Z flag) Tests if Z is even.
107006	^ZNLT?	(Z1 Z2 → flag) TRUE if $\lvert Z1\rvert < \lvert Z2\rvert$.
19A006	^OBJINT?	(z/% → z flag) Tests if Obj is an integer.
19B006	^OBJPOSINT?	(z/% → z flag) Tests if Obj is a positive integer smaller than Zsmall.
19C006	^CKINT>0	(Obj → Obj flag) Tests if Obj is a strictly positive integer.
198006	^METAINT?	(Meta → Meta flag) Tests if Meta is an integer.

Addr.	**Name**	**Description**
199006	ˆMETAPOSINT?	`( Meta → Meta flag )` Tests if Meta is a positive integer smaller than Zsmall.
0CC006	ˆDupTypeS?	`( Z → Z flag )` Tests if Z is short ($\leq$ 64 bits).

Chapter 43
Matrices

The CAS' Symbolic Matrices are a new object on the HP49 used to represent matrices. Unlike the old array object present since the HP48, these matrices can have symbolic expressions inside them. It is also possible to have objects of different types inside the array.

This kind of matrix is actually a composite object, and you can use the functions of Chapter 11 on them.

The following disassembly of the matrix $\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$ should make it clear how to create one using MASD, and why they are actually composites:

```
MATRIX
  MATRIX
    ZINT 1
    ZINT 2
  ;
  MATRIX
    ZINT 3
    ZINT 4
  ;
;
```

It should also be noted that most (if not all) the functions described below for dealing with symbolic matrices also work with lists of lists. The reason should be obvious: the structure of these matrices and a list of list is the same, only the prolog address changes.

Some entries dealing with the old HP48 arrays, described in Chapter 10.1, also work with symbolic matrices.

43.1 Reference

43.1.1 Creating and Redimensioning Matrices

Addr.	Name	Description
371006	^MATIDN	(M/z/% → M') Creates identity matrix.
372006	^MATCON	(M ob → [ob]) Creates constant matrix from matrix.
373006	^MAKEARRY	({#el} symb → []) ({#rows #cols} symb → [[]]) Creates constant matrix/array from ob type.
345006	^DIMRANM	({} → M') Creates symbolic random matrix from dimensions.
344006	^MATRANM	(M → M') Changes all elements of matrix to elements generated randomly.
374006	^OBJDIMS2MAT	(ob {} → M) Creates constant matrix from dimension and ob.
375006	^LCPROG2M	(#n #m prg → M) Fills a matrix of specified size using a program. prg must take two arguments and return one argument. On entry MAKE2DMATRIX provide the indexes as Z integers.
376006	^MAKE2DMATRIX	(#n #m prg → M) Creates matrix from size and program (with stack checking). prg must take 2 args and return 1 arg. On entry MAKE2DMATRIX provide the indexes as Z integers.
377006	^make2dmatrix	(#n #m prg → meta-M) Create meta-matrix from size and program (with stack checking). prg must take 2 args and return 1 arg On entry make2dmatrix provide the indexes as Z integers.
341006	^MATREDIM	(M {} → M') Changes size of a matrix, removing elements and/or adding zeros, as necessary.

Addr.	Name	Description
342006	^VRRDM	([]/[[]] {} → []) Vector Right ReDiMension: adds 0 to the right.
343006	^VRRDMmeta	(meta #l → meta-#l) Meta Right ReDiMension: adds 0 to the right.

43.1.2 Conversion

Addr.	Name	Description
16A006	^{}TO[]	({} → []) Converts from list-of-lists representation to matrix. No checks on the element type.
17A006	^LIST2MATRIX	({} → []) ({{}} → [[]]) (ob → ob) Converts a symbolic list to a matrix. Does not check that matrix is a valid one. Use DTYPFMAT? to do that.
16B006	^[]TO{}	([] → {}) Converts from matrix to list-of-lists.
179006	^MATRIX2LIST	([] → { }) ([[]] → {{}}) (ob → ob) Converts a symbolic matrix to a list.
17E006	^ARRAY2MATRIX	([] → []) ([[]] → [[]]) Converts array to symbolic array if necessary.
175006	^SAMEMATRIX	(M1 M2 → M1 M2 flag) If one object is a symbolic array, converts both arrays to symbolic form. Returns TRUE for symbolic matrices and FALSE for numeric.
176006	^SAMEMATSCTYPE	(M ob → M ob flag) If M is a numeric matrix and ob is not float, converts matrix to symbolic form. Returns TRUE for symbolic and FALSE for numeric.
003007	^ArryToList	([]/[[]] → {}/{{}}) Converts normal array to list of lists; errors for symbolic arrays.

Addr.	Name	Description
17D006	^MATEXPLODE	([[ob1..obn]] → ob1..obn [[ob1..obn]])

43.1.3 Tests

Addr.	Name	Description
16C006	^DUPNULL[]?	(ob → ob flag) Tests for a null array.
359006	^NULLVECTOR?	(V → flag) Returns true if vector is null.
16F006	^CKSAMESIZE	(arry1 arry2 → arry1 arry2 flag) Tests if arry1 and 2 have the same size.
170006	^DTYPENDO?	(ob → ob flag) Tests if object is a square symbolic matrix. Convert numeric array to symbolic matrix.
173006	^2DMATRIX?	(ob → ob flag) Tests if object is a 2D matrix.

43.1.4 Calculations with Matrices

Addr.	Name	Description
320006	^MAT+	(M2 M1 → M2+M1)
321006	^MADD	(M2 M1 → M2+M1)
322006	^MAT-	(M2 M1 → M2-M1)
323006	^MSUB	(M2 M1 → M2-M1)
324006	^VADD	(V2 V1 → V2+V1)
325006	^VSUB	(V2 V1 → V2-V1)
326006	^MAT*	(M2 M1 → M2*M1) Matrix product with size and type checking.
327006	^MMMULT	(M2 M1 → M2*M1)
328006	^MVMULT	(M V → V') Product of matrix by vector.
329006	^SCL*MAT	(ob M → M*ob) Scalar times matrix.
32A006	^MAT*SCL	(M ob → M*ob) Matrix times scalar.

Addr.	Name	Description
32B006	^VPMULT	(V ob → V') Multiplies vector by a scalar.
335006	^MATSQUARE	(M → M*M)
32C006	^MAT^	(M z/% → M') Integral matrix power.
32D006	^MATCROSS	([] []' → []'') Vector product.
32E006	^MATDOT	(V2 V1 → ob) Scalar product with checking.
32F006	^RNDARRY	(M % → M) Rounds array.
330006	^TRCARRY	(M % → M) Truncates array.
332006	^MAT/SCL	(M ob → M/ob) Divides matrix by scalar.
333006	^MAT/	(V M → M^-1*V) "Divides" Vector by matrix.
334006	^MATCHS	(M → -M)
34E006	^MATINV	(M → M^-1)
336006	^MATCONJ	(M → M')
337006	^MATRE	(M → re[M])
338006	^MATIM	(M → im[M])
339006	^MATTRACE	(M → trace) Matrix trace.
33A006	^MATTRN	(M → M') Matrix transposition and conjugation.
33C006	^mattran	(M → Meta-M') Transposes matrix, returns meta-matrix.
33D006	^mattrn	(Meta-M → Meta-M') Transposes meta-matrix.
346006	^MATDET	(M → det) Determinant, expanding all (not row reduction).
347006	^MATRDET	(M → det) Determinant using row reduction.
348006	^MATFNORM	(M → ob) Frobenius norm.
349006	^MATRNORM	(M → ob) Row norm.

Addr.	Name	Description
34A006	^MATCNORM	`( M → ob )` Column norm.
174006	^MATRIXDIM	`( ob → # )` Returns symbolic matrix dimensionality of an object.

43.1.5 Linear Algebra and Gaussian Reduction

Addr.	Name	Description
34C006	^MATREF	`( M → M' )` Returns matrix in Row-Echelon form.
34B006	^MATRREF	`( M → M' )` Returns matrix in Reduced Row-Echelon form.
34F006	^MATREFRREF	`( M #full_ref → M list M' )` If #full_ref is 1, returns Reduced Row-Echelon form, otherwise returns just Row-Echolong form.
367006	^MATRIXRCI	`( ncol i M const → M' )` Multiplies row #i of symbolic matrix M by constant. ncol is not used, it's here because of the stack state at call-time from inside laRCI.
368006	^MATRIXRCIJ	`( ncol #i #j M const → M' )` Does Lj <- c*Li+Lj. ncol is not used, it's here because of the stack state at call-time from inside laRCI.
350006	^INXREDext	`( Lvar #full_ref M → Lvar pivot M )`
351006	^METAMATRED	`( Meta-M Lvar #full_red → meta-M Lvar pivot )`
352006	^METAPIVOT	`( meta-M #l #c → meta-M #l #l' #c' flag )` Searchs a pivot in column #c starting from row #l. Flag is FALSE if pivot is not found. If pivot is found #l' is the row, #c is updated to #c'.
354006	^PIVOTFLOAT	`( float → float_modulus )`
34D006	^MATRANK	`( M → Z/% )` Rank of a matrix.

43.1.6 Linear System Solver

Addr.	Name	Description
080007	^LINSOLV	(b a → y) Solves y'=ay+b.
0F4007	^SOLVEMETASYST	(meta-M → d meta-sol T) (meta-M → F) Solves linear system in meta representation. Meta-sol has been reduced to the same denominator d.
0F5007	^REDUCEMETASYST	(meta-M → meta->M') Reduces linear system in meta representation.
0F6007	^REDUCEMETAPSYST	(meta-M → meta-M') Reduces linear system in meta representation. Does not reduce last column of meta-matr. This is useful to solve linear system with parameters in the last column.
0F7007	^SOLVECRAMER	(meta-M → d meta-sol T) (meta-M → F) Solves cramer system. Meta-matr must be fully reduced. Meta-sol is reduced to the same denominator. d flag is FALSE if dimension do not match.
355006	^SYSText	(M linc → linc linc' res cas_p)
356006	^STOSYSText	(M2 M1 → M2 list)
357006	^MAKESYSText	(M_eq M_inc → M_eq M lidnt flag) Converts linear equations to a matrix and checks that equation are linear with respect to lidnt.

43.1.7 Other Matrix Operations

Addr.	Name	Description
35A006	^FINDELN	({} A → # flag) Returns index # of element {} in array.

Addr.	Name	Description
35B006	^PULLEL[S]	`( A # → A el )` Extracts element of index # from array. Array type test is made in assembly for array speed.
35C006	^BANGARRY	`( el # M → M' )` Puts el at index # of matrix M.
35D006	^PUT[]	`( el #i V → V )` Replaces #i-th vector component by element.
17B006	^LENMATRIX	`( [] → #el )` `( [[]] → #row )`
33E006	^MATSUB	`( M rmin nrows cmin ncols { #m #n } → M' )` Extracts submatrix from a matrix.
340006	^MATREPL	`( M1 M2 → M2' )` Replaces part of matrix destination (M2) by matrix source (M1). LAM1 to 9 must be bound like in Llib/LIMain.s (9:r 8:c 7:dmat? 6:f 5:md 4:nd 3:smat? 2:ms 1:ns). Copy begins in matrix d at row r and column c.
35F006	^MATRIX>DIAG	`( A ncols+1 ndiags → V )` Extracts diagonal terms. ncols+1 is there because `MATRIX>DIAG` is called inside la>DIAG.
360006	^MATRIXDIAG>	`( ncol+1 diagV dlen dims{} → M )` Constructs a matrix from a vector of diagonal terms.
361006	^la+ELEMsym	`( V ob %i → V' )` Inserts element in symbolic vector at row %i.
362006	^INSERTROW[]	`( V ob #i → V )` `( M V #i → M' )` Inserts element/vector in symbolic vector/matrix at row #i. Checks for 0 < #i < #n + 1, but does not check for matrix/vector size.
363006	^insertrow[]	`( ob #i meta → meta )` Inserts element/vector in meta-object at position #i. Checks for 0 < #i < #n + 1, but does not check for vector size.

Addr.	Name	Description
364006	^INSERTCOL[]	(M V #i → M') Inserts vector in symbolic matrix at col #i. Checks for 0 < #i < #n + 1, but does not check for matrix/vector size.
365006	^INSERT[]ROW[]	(M3 M2 #i → M) Inserts matrix2 in matrix3 starting from row #i. Checks for 0 < #i < #n+1, but does not check for matrix size.
366006	^INSERT[]COL[]	(M3 M2 #i → M) Inserts matrix2 in matrix3 starting from row #i. Checks for 0 < #i < #n + 1, but does not check for matrix size.
369006	^MATRIXCSWAP	(M #c #c' → M) Exchanges columns c and c' of a symbolic matrix.
36A006	^MATRIXRSWAP	(M #r #r' → M) Exchanges lines r and r' of a symbolic matrix.
0AC003	^SWAPROWS	(M % %' → M') SWAP two rows in matrix. Internal version of xRSWP.
36B006	^MATRIX-ROW	(M #r → M' lr) Extracts row #r from M. Checks boundaries.
36C006	^METAMAT-ROW	(meta-M #r → meta-M lr) Extracts row #r from meta-matrix. Checks boundaries.
36D006	^MATRIX-COL	(M #c → M cc) Extracts column #r from matrix. Checks boundaries.
36E006	^METAMATCSWAP	(meta-M #c #c' → meta-M) Exchanges columns c and c' of a meta-matrix.
36F006	^METAMATRSWAP	(meta-M #l #l' → meta-M) Exchanges lines l and l' of a meta-matrix (or vector).
370006	^STOMAText	(M →) Stores matrix in 'MATRIX' in current directory.

Addr.	Name	Description
378006	^ADDMATOBJext	(arry ob → arry arry) (ob arry → arry arry) Used for addition of numeric matrix and symbolic object.
379006	^VUNARYOP	(v op → V) Applies unary op(v[i]) to get V[i].
37A006	^VBINARYOP	(V2 V1 binop → V) Works even if V2 and V1 do not have not the same dimension.
37B006	^PEVAL	(V r → P[r]) Horner evaluation, where elements of V represent coefficients of a polynomial.

43.1.8 Eigenvalues, Eigenfunctions, Reduction

Addr.	Name	Description
37C006	^MATEGVL	(M → V) Computes eigenvalues of a matrix like EGVL.
37F006	^MATEGV	(M → V) Computes eigenvalues/eigenvectors of a matrix like EGV.
37E006	^MADJ	(M → M^-1 P[M] P[lambda]) Computes inverse, matrix polynomial and characteristic polynomial.
380006	^JORDAN	(M → pmin pcar {evect} {eval}) (pmadj pcar → pmin pcar {evect} {eval}) Eigenvalue/eigenfunctions computation.
22D006	^FLAGJORDAN	(M →) Internal JORDAN.
381006	^QXA	(symb lidnt → M lidnt) Converts symbolic quad form to matrix quad form.
224006	^FLAGQXA	(symb lidnt → M lidnt) Internal QXA.
382006	^AXQ	(M lidnt → symb lidnt) Converts matrix quad form to qymbolic quad form.

Addr.	Name	Description
225006	^FLAGAXQ	(M lidnt → symb lidnt) Internal AXQ.
383006	^GAUSS	(symb → D P symb') Gauss reduction of quadratic form (symbolic).
226006	^FLAGGAUSS	(symb lidnt → symb') Internal GAUSS.
384006	^SYLVESTER	(M → D P) Gauss reduction of a quadratic form (matrix).
227006	^FLAGSYLVESTER	(M → P D) Internal SYLVESTER.
228006	^PCAR	([[]] → symb) Internal PCAR.

Chapter 44
Expression Manipulation

The entries in this chapter are used for manipulation of expressions, when they are represented in their symbolic objects form. (See Chapter 45 for entries that deal with symbolics in Metaobject form). There are entries related to collection and expasion, trigonometric and exponential transformations and substitution of values in expressions.

44.1 Reference

44.1.1 Basic Operations and Function Application

Addr.	Name	Description
125006	`^x+ext`	`( ob2 ob1 → ob2+ob1 )` Symbolic addition, tests for infinities.
126006	`^x-ext`	`( ob2 ob1 → ob2-ob1 )` Symbolic subtraction, tests for infinities.
127006	`^x*ext`	`( ob2 ob1 → ob2*ob1 )` Symbolic multiplication, tests for infinities.
129006	`^x/ext`	`( ob2 ob1 → ob2/ob1 )` Symbolic division, tests for infinities.
12B006	`^x^ext`	`( ob power → ob^power )` Power.
12C006	`^EXPAND^`	`( x y → x^y=exp[y*ln[x]] )` Power with simplifications. If y is a fraction of integers, use XROOT^ instead.
4FB006	`^QNeg`	`( ob → -ob )` Symbolic negation.
4FC006	`^RNEGext`	`( ob → -ob )` Symbolic negation.

Addr.	Name	Description
4FA006	^SWAPRNEG	(ob2 ob1 → ob1 -ob2) Does SWAP then symbolic negation.
4FE006	^RREext	(ob → Re(ob)) Symboloc real part.
4FD006	^SWAPRRE	(ob2 ob1 → ob1 Re(ob2)) SWAP, then RREext.
500006	^RIMext	(ob → Im(ob)) Symbolic imaginary part.
4FF006	^SWAPRIM	(ob1 ob2 → ob2 Im(ob1)) SWAP, then RIMext.
501006	^xREext	(symb → symb') Complex real part. Expands only + - * / ^.
503006	^xIMext	(symb → symb') Complex imaginary part. Expands only + - * / ^.
505006	^RCONJext	(ob → Conj(ob)) Symbolic complex conjugate.
50D006	^xABSext	(ob → abs(ob)) Symbolic ABS function.
50A006	^RABSext	(ob → abs(ob)) Internal ABS. Internal representation.
52A006	^xINVext	(ob → 1/ob) Symbolic inversion.
557006	^xSYMINV	(symb → 1/symb) Symbolic inversion.
553006	^xSQext	(symb → sq(symb)) Symbolic square.
555006	^xSYMSQ	(symb → symb^2)
51B006	^SXSQRext	(ob → sqrt(ob)) Does not take care of the sign.
51C006	^XSQRext	(ob → sqrt(ob)) Tries to return a positive square root if nocareflag is cleared.
52B006	^xvext	(ob → sqrt(ob)) Symbolic square root, tests for 0 and 1.
552006	^xSYMSQRT	(symb → sqrt(symb))
521006	^CKLN	(ob → ln(ob)) Symbolic LN with special handling for fractions. Does not use the internal representation.

Addr.	Name	Description
522006	ˆxLNext	(ob → ln(ob)) Symbolic LN, without fraction handling.
525006	ˆEXPANDLN	(ob → ln(ob)) Symbolic LN using internal representation. Before switching to internal representation, test for ABS, 0 and 1 and, in real mode, test if ob=exp(x).
528006	ˆREALLN	(ob → ln(ob)) Internal natural logarithm for a real argument.
526006	ˆCMPLXLN	(ob → ln(ob)) Internal complex natural logarithm.
527006	ˆLNATANext	(ob → ln(ob)) Internal natural logarithm for complex.
529006	ˆxEXPext	(y d n → exp(y*n/d*i*π)) Symbolic EXP, tests for 0, infinity and i*k*π/12 where k is an integer. Tests for d=1,2,3,4,6.
52C006	ˆxCOSext	(ob → cos(ob)) Symbolic COS, tests for 0 and multiples of π/12. Also tests if ob=acos(x) or ob=asin(x).
536006	ˆxSYMCOS	(ob → cos(ob))
533006	ˆxACOSext	(ob → acos(ob)) Symbolic ACOS. Tests for 0, infinity and tables.
53F006	ˆxSYMACOS	(ob → acos(ob))
52D006	ˆxSINext	(ob → sin(ob)) Symbolic SIN, tests for 0 and multiplies of π/12. Also tests if ob=acos(x) or ob=asin(x).
538006	ˆxSYMSIN	(ob → sin(ob))
532006	ˆxASINext	(ob → asin(ob)) Symbolic ASIN. Tests for 0, infinity and tables.
53D006	ˆxSYMASIN	(ob → asin(ob))
52E006	ˆxTANext	(ob → tan(ob)) Symbolic TAN. Tests for 0 and multiplies of π/12. Also tests if ob=atan(x).
53A006	ˆxSYMTAN	(ob → tan(ob))
534006	ˆxATANext	(ob → atan(ob)) Symbolic ATAN. Tests for 0, infinity and tables.
541006	ˆxSYMATAN	(ob → atan(ob))
52F006	ˆxCOSHext	(ob → cosh(ob)) Symbolic COSH. Tests for 0, infinity and acosh(x).
545006	ˆxSYMCOSH	(ob → cosh(ob))

Addr.	Name	Description
54E006	^xACOSHext	(symb → acosh(symb)) Symbolic ACOSH.
550006	^xSYMACOSH	(symb → acosh(symb))
530006	^xSINHext	(ob → sinh(ob)) Symbolic SINH. Tests for 0, infinity and asinh(x).
543006	^xSYMSINH	(ob → sinh(ob))
54B006	^xASINHext	(symb → symb') Symbolic ASINH.
54D006	^xSYMASINH	(symb → asinh(symb))
531006	^xTANHext	(ob → tanh(ob)) Symbolic TANH. Tests for 0 and atanh(x).
547006	^xSYMTANH	(ob → tanh(ob)) Symbolic TANH.
548006	^xATANHext	(symb → symb') Symbolic ATANH.
54A006	^xSYMATANH	(ob → atanh(ob))
55F006	^xSYMFLOOR	(symb → symb')
561006	^xSYMCEIL	(symb → symb')
563006	^xSYMIP	(symb → symb')
565006	^xSYMFP	(symb → symb')
567006	^xSYMXPON	(symb → symb')
569006	^xSYMMANT	(symb → symb')
56B006	^xSYMLNP1	(symb → symb')
56D006	^xSYMLOG	(symb → symb')
56F006	^xSYMALOG	(symb → symb')
571006	^xSYMEXPM1	(symb → symb')
572006	^factorial	(symb → symb!) Symbolic factorial.
573006	^facts	(symb → symb!) Symbolic factorial.
575006	^xSYMFACT	(symb → symb!)
578006	^xSYMNOT	(symb → symb')
128006	^x=ext	(ob2 ob1 → ob2=ob1)

44.1.2 Trigonometric and Exponential Operators

Addr.	Name	Description
408006	^COS2TAN/2	(symb → symb') x → (1-(tan(x/2))^2)/(1+(tan(x/2))^2)
40B006	^SIN2TAN/2	(symb → symb') x → 2 tan(x/2)/(1+(tan(x/2))^2)
40E006	^TAN2TAN/2	(symb → symb') x → 2 tan(x/2)/(1-(tan(x/2))^2)
412006	^COS2TAN	(symb → symb2) x → 1/sqrt(1+(tan(x))^2)
414006	^SIN2TAN	(symb → symb') x → tan(x)/sqrt(1+(tan(x))^2)
41A006	^LNP12LN	(symb → symb') x → ln(x+1)
41B006	^LOG2LN	(symb → symb') x → log(x)
41C006	^ALOG2EXP	(symb → symb') x → alog(x)
41D006	^EXPM2EXP	(symb → symb') x → exp(x)-1
41E006	^SQRT2LNEXP	(symb → symb') x → exp(ln(x)/2)
41F006	^sqrt2lnexp	(meta → meta') x → exp(ln(x)/2)
420006	^TAN2EXP	(symb → symb') x → (exp(i2x)-1)/(i*(exp(i2x)+1))
422006	^ASIN2LN	(symb → symb') x → = i*ln(x+sqrt(x^2-1))+pi/2.
424006	^ACOS2LN	(symb → symb') x → ln(x+sqrt(x^2-1))/i
427006	^TAN2SC	(symb → symb') x → sin(x)/cos(x)
42A006	^SIN2TC	(symb → symb') x → cos(x)*tan(x)
42C006	^COS2ext	(symb → symb') x → sqrt(1-(sin(x))^2).
42E006	^SIN2ext	(symb → symb') x → sqrt(1-(cos(x))^2).

Addr.	Name	Description
431006	^ATAN2ASIN	(symb → symb') x → asin(x/sqrt(x^2+1))
434006	^ASIN2ATAN	(symb → symb') x → atan(x/sqrt(1-x^2))
437006	^ASIN2ACOS	(symb → symb') x → π/2-acos(x)
43C006	^ACOS2ASIN	(symb → symb') x → π/2-asin(x)
43D006	^ATAN2LNext	(symb → symb') x → i/2*ln((i+x)/(i-x))
440006	^TAN2SC2	(symb → symb') x → (1-cos(2x))/sin(2x)
442006	^TAN2CS2	(symb → symb') x → sin(2x)/(1+cos(2x))
444006	^SIN2EXPext	(symb → symb') x → (e^(i*x)-1/e^(i*x))/(2i)
446006	^COS2EXPext	(symb → symb') x → (e^(i*x)+1/e^(i*x))/2
448006	^SINH2EXPext	(symb → symb') x → (e^x-1/e^x)/2
44A006	^COSH2EXPext	(symb → symb') x → (e^x+1/e^x)/2
44C006	^TANH2EXPext	(symb → symb') x → (e^2x-1)/(e^2x+1)
44E006	^ASINH2LNext	(symb → symb') x → ln(x+sqrt(x^2+1))
450006	^ACOSH2LNext	(symb → symb') x → ln(x+sqrt(x^2-1))
452006	^ATANH2LNext	(symb → symb') x → ln((1+x)/(1-x))/2
454006	^XROOT2ext	(symb1 symb2 → symb') x y → exp(ln(y)/x)
45A006	^LN2ATAN	(symb → symb') x → ln(x)

44.1.3 Simplification, Evaluation and Substitution

Addr.	Name	Description
45B006	^VAR=LIST	(idnt {} → {}') Replaces all elements of the initial list by idnt=element.
464006	^SYMBEXEC	(ob symb → ob') If symb is an equation, executes the corresponding change of variables in ob, otherwise tries to find symb so that ob is zero. Note that change of variable works for change of user functions.
465006	^MEVALext	(ob {} {}' → ob') Replaces all occurrances of an element of list2 by the corresponding element of list1 in ob. Looks in ob from outer to inner expressions. list2 and list1 may contain secondaries. If vxxlflag is set SIGN var are leaved unchanged.
466006	^CASNUMEVAL	(symb list1 list2 → symb') Evaluation of a symbolic. The lists' formats are list1={idnt/lam1... idnt_n/lam_n} list2={value1...value_n}. The idnt's/lam's in list1 are *not* evaluated before replacing value1...value_n.
467006	^CASCOMPEVAL	(symb → symb') Evaluation of a symbolic.
468006	^REPLACE2BY1	(symb idnt a → symb') Evaluation of a symbolic replacing an idnt by a value; for example evaluation of F(X) for X=1/2)
469006	^NR_REPLACE	(symb idnt a → symb') Like REPLACE2BY1 but prevents evaluation of INT.
46B006	^CASCRUNCH	(ob → %) Like CRUNCH but in approximate mode.
46C006	^APPROXCOMPEVAL	(symb → symb') Like CASCOMPEVAL but in approximate mode.
11A007	^ALGCASCOMPEVAL	(expr → expr)

Addr.	Name	Description
297006	^SLVARext	(Lvar → Lvar') Simplifies all elements of the list that are supposed to be variables.
298006	^SIMPLIFY	(symb → symb') Simplifies one object like EVAL.
299006	^SIMP1ext	(symb → symb') Simplifies one object like EXPAND. Object must be a symbolic, a real or a complex number.
29A006	^SYMEXPAN	(symb → symb') Simplifies one object like EXPAN. Object must be symb/real/cmplx.
29B006	^SIMPVAR	(ob → ob') Simplifies variable.
2A0006	^SIMPSYMBS	(inf sup fcn var → int(inf,sup,fcn,var))
2A2006	^SIMPUSERFCN	(ob1..obn #n ob → id[]) Simplification of user functions. Tests for derivative of user functions. Ob must be an id, a symbolic, a secondary or a romptr.
2A3006	^EVALUSERFCN	(V1..Vn #n fcn → f[]) Evaluates a user function with stack checking.
2A4006	^SIMP|	(ob list → ob') Executes the WHERE operator.
2A9006	^SIMPext	(ob1 ob2 → ob1' ob2') Simplifies two objects in internal representation. Checks that o2 is not a complex or an irrquad because decomposition of the corresponding fraction with larg would generate a "Try to recover Memory".
2AD006	^SIMPGCDext	(o1 o2 gcd → o1/gcd o2/gcd) Divides o1 and o2 by gcd.
2AE006	^SIMP3ext	(a b → g a'' b'') Calculates g = gcd(a,b) and a''=a/g and b''=b/g.
2B9006	^TSIMP2ext	(symb → symb) Transcendental simplifications. Converts only sqrt ^ and XROOT to EXP/LN. LN are returned as -1/INV[-LN[]] for use by SERIES.

Addr.	Name	Description
2BA006	`^TSIMPext`	`( symb → symb )` Transcendental simplifications. Convert transcendental functions to EXP and LN.
2BB006	`^TSIMP3ext`	`( symb → symb )`

44.1.4 Collection and Expansion

Addr.	Name	Description
26E006	`^COLCext`	`( symb → symb' )` Factorization with respect to the current variable of symb and factorization of the integer content of symb.
2FE006	`^TCOLLECT`	`( symb → symb' )` Performs trigonometric linearization and then collects sines and cosines of the same angle.
2FF006	`^SIGMAEXPext`	`( symb → symb' )` Conversion to exp and ln with exponential linearization.
300006	`^LINEXPext`	`( symb → Meta )` Meta = arg_exp1 coef1 ... arg_expn coefn #2n.
301006	`^SIGMAEXP2ext`	`( Meta → symb )` Back conversion from arg_exp/coef_meta to symbolic.
303006	`^SINEXPA`	`( symb → symb' )` Expands SIN.
316006	`^LNEXPA`	`( symb → symb' )` Expands LN.
31C006	`^MTRIG2SYMB`	`( Meta → symb )` Back conversion of trig-meta to symbolic.
309006	`^COSEXPA`	`( symb → symb' )` Expands COS.
30F006	`^EXPEXPA`	`( symb → symb' )` Expands EXP.
31B006	`^LINEXPA`	`( symb → Meta )` Alternates trig operator and coefficient.
31D006	`^LNCOLCext`	`( symb → symb' )` Collects logarithms.

Addr.	Name	Description
31F006	^TEXPAext	(symb → symb) Main transcendental expansion program.
240006	^EXLR	('a=b' → a b) (ob → X ob) Internal equation splitter.

44.1.5 Trigonometric Transformations

Addr.	Name	Description
407006	^HALFTAN	(symb → symb') Converts trigonometric functions to TAN of the half angle.
411006	^TRIGTAN	(symb → symb') Convert sin and cos to tan of the same angle.
416006	^TRIGext	(symb → symb') Applies sin^2+cos^2=1 to simplify trigonometric expressions. If flag -116 is set, tries to keep only sin, else only cos.
417006	^HYP2EXPext	(symb → symb') Converts hyperbolic functions to exp and ln. Converts XROOT and ^ to exp and ln.
418006	^EXPLNext	(symb → symb') Converts all transcendental functions to exp and ln.
419006	^SERIESEXPLN	(symb → symb') Converts sqrt, ^ and XROOT to EXP/LN.
426006	^TAN2SCext	(symb → symb') Converts tan to sin/cos.
429006	^SIN2TCext	(symb → symb') Converts sin to cos*tan.
430006	^ATAN2Sext	(symb → symb') Converts ATAN to ASIN using asin(x)=atan(x/sqrt(1-x^2)).
433006	^ASIN2Text	(symb → symb') Converts ASIN to ATAN using asin(x)=atan(x/sqrt(1-x^2)).

Addr.	Name	Description
436006	^ASIN2Cext	(symb → symb') Converts ASIN to ACOS using asin(x)=pi/2-acos(x).
43A006	^ACOS2Sext	(symb → symb') Converts ACOS to ASIN using acos(x)=pi/2-asin(x).
43F006	^TAN2SC2ext	(symb → symb') Converts TAN to SIN/COS of the double angle. If flag -116 is set calls TAN2SC2, else TAN2CS2.
456006	^LN2ext	(symb → symb') If symb contains x, returns -1/inv(-ln(x)), else ln(x). Used by SERIES.

44.1.6 Division, GCD and LCM

Addr.	Name	Description
3E8006	^PSEUDODIV	(Q2 Q1 → a Q2*a/Q1 Q2*a/Q1)
3EA006	^BESTDIV2	(o2 o1 → quo mod)
3EC006	^QUOText	(o2 o1 → o2 div o1) Euclidean quotient of 2 objets (works even if o2 mod o1=0).
3ED006	^NEWDIVext	(ob2 ob1 → quo mod) Euclidean division, ob2 and ob1 may be fractions of returns a fraction of Q.
3F3006	^QUOTOBJext	(a_a-1...a0 bb_1...b0 #b #a flag → r q) SRPL Euclidean division: step 2 computes the remainder r only if flag is TRUE.
3F4006	^DIVISIBLE?	(a b → a/b T) (a b → ob F) Returns TRUE and quotient if b divides a, otherwise returns FALSE.
3F5006	^QDiv?	(a b → a/b T) (a b → F) Returns TRUE and quotient if b divides a, otherwise returns FALSE.

Addr.	**Name**	**Description**
3F6006	ˆFastDiv?	(P Q → P/Q PmodQ T) Euclidean division. Assumes P and Q have integer or Gaussian integer coefficient. Returns FALSE in complex mode or if sparse short division fails.
3F7006	ˆPOTENCEext	(z1 z2 → q r) Step by step Euclidean division for small integers.
2A5006	ˆDENOLCMext	(list → ob) Calculates the LCM of the denominator of the elements of the list. If input is not a list, returns the denominator of the object.
2A6006	ˆMETADENOLCM	(Meta → ob) Calculates LCM of the denominators of the elements of Meta.
2B1006	ˆLPGCDext	({} → {} ob) Calculates the GCD of all the elements in the list. The algorithm is far from optimal.
2B2006	ˆSLOWGCDext	(c 1 A B → c* gcd(A,B)) Euclidean algorithm for polynomial GCD. Used if A or B contains irrquads. c is the GCD of the contents of the original polynomials returned after failure of GCDHEUext.
2B3006	ˆQGcd	(ob2 ob1 → gcd) Generic internal GCD. (LAM2: GCDext ob1, ob2 → pgcd).

Chapter 45
Symbolic Meta Handling

This chapter contains words that modify metas which are exploded symbolic objects. They are used to modify the expression or to operate on them.

45.1 Reference

45.1.1 Basic Expression Manipulation

Addr.	Name	Description

157006	^SYMBINCOMP	`( symb → ob1 .. obN #n )`
		`( ob → ob #1 )`
		`( {} → {} #1 )`
		Explodes symbolic object into meta. Other objects are converted into one-object metas by pushing #1 into the stack.

386006	^m-1&m+1	`( meta → meta&1&+ meta&1&- )` Creates two copies of the meta. To the first one, adds 1 and +, to the second one, adds 1 and -.
387006	^meta1/meta	`( meta → meta 1&meta&/ )` Duplicates the meta, and inverts the expression represented by it.
388006	^1&meta	`( Meta → 1&Meta )` Prepends the number 1 to the meta.
389006	^meta/2	`( Meta → Meta&2&/ )` Divides the expression by two.
38A006	^addt2	`( Meta → Meta&2 )` Appends the number 2 to the meta.
38B006	^addt/	`( Meta → Meta&/ )` Appends division to meta.

Addr.	Name	Description
38C006	^meta2*	(Meta → 2&Meta&*) Multiplies the expression by 2.
459006	^metai*	(meta → meta*i) Multiplies meta by i.
38D006	^meta1-sq	(Meta → 1&Meta&SQ&-) Changes x into 1-x^2, where x is the original expression.
38E006	^metasq+1	(Meta → Meta&SQ&1&+) Changes x into x^2+1, where x is the original expression.
38F006	^metasq-1	(Meta → Meta&SQ&1&-) Changes x into x^2-1, where x is the original equation.
390006	^meta-1	(Meta → Meta&1&-) Subtracts one from the expression.
398006	^addt^	(Meat → Meta&^) Append power operator to meta object.
39C006	^top&addt*	(meta2 meta1 → meta2*meta1) top& addt*. No checks.
39D006	^top&addt/	(meta2 meta1 → meta2/meta1) top& addt/. No checks.
39E006	^addti	(meta → meta&i) Appends i (the Imaginary unit) to expression.

45.1.2 Basic Operations and Function Application

Addr.	Name	Description
393006	^metaadd	(Meta1 Meta2 → Meta1+Meta2) Adds 2 meta objects with trivial simplifications. metaadd checks for Meta1/2=Z0 ONE.
3AB006	^MetaAdd	(Meta2 Meta1 → Meta2+Meta1) Adds 2 meta objects with trivial simplifications. Checks for infinities then call metaadd.
1CE006	^ckaddt+	(Meta1 Meta2 → Meta1+Meta2) Adds 2 meta objects with trivial simplifications.

Addr.	Name	Description
394006	^metasub	(Meta1 Meta2 → Meta1+Meta2) Subtracts 2 meta objects with trivial simplifications. metasub checks for Meta1/2=Z0 ONE.
3AD006	^MetaSub	(Meta2 Meta1 → Meta2-Meta1) Subtracts 2 meta objects with trivial simplifications. Checks for infinities then call metasub.
1CF006	^ckaddt-	(Meta1 Meta2 → Meta1+Meta2) Subtracts 2 meta objects with trivial simplifications.
395006	^metamult	(Meta1 Meta2 → Meta1*Meta2) Multiplies 2 meta objects with trivial simplifications. Checks for meta1, meta2= Z0 or Z1, checks for xNEG.
3AF006	^MetaMul	(Meta2 Meta1 → Meta2*Meta1) Multiplies 2 meta objects with trivial simplifications. Checks for infinities/0 then call metamult.
1CD006	^ckaddt*	(Meta1 Meta2 → Meta1*Meta2) Multiplies 2 meta objects with trivial simplifications.
396006	^metadiv	(Meta2 Meta1 → Meta2/Meta1) Divides 2 meta objects with trivial simplifications. Checks for infinities and 0, meta2 =1 or Z-1, checks for xNEG.
3B1006	^MetaDiv	(Meta2 Meta1 → Meta2/Meta1) Divide 2 meta objects with trivial simplifications. Checks for infinities and 0 then call metadiv.
3F1006	^DIVMETAOBJ	(o1...on #n ob → {o1/ob...on/ob}) Division of all elements of a meta by ob. Tests if o=1.
397006	^meta^	(Meta ob → Meta&ob&^) Elevates expression to a power. If ob=1, just returns the expression. Tests for present of xNEG in the end of meta for integral powers.

Addr.	Name	Description
399006	^metapow	(Meta2 Meta1 → Meta2^Meta1) Elevates expression to a power (any other expression). If length of Meta1 is ONE, calls meta^.
3B5006	^MetaPow	(Meta2 Meta1 → Meta2^Meta1) Power. Checks for infinities then calls metapow.
39B006	^metaxroot	(Meta2 Meta1 → Meta2&XROOT&Meta1) Root of expression.
3B9006	^metaneg	(meta → meta) Checks only for meta finishing by xNEG.
3BA006	^metackneg	(meta → meta) Like metaneg but checks for meta=ob ONE.
3B7006	^MetaNeg	(Meta → Meta) Negates meta. Only checks for metas finishing by xNEG.
502006	^xSYMRE	(meta → meta') Meta complex real part. Expands only + - * / ^.
504006	^xSYMIM	(meta → meta') Meta complex imaginary part. Expands only + - * / ^.
50E006	^addtABS	(Meta → Meta') Meta ABS. Does a CRUNCH first to find sign.
510006	^addtABSEXACT	(Meta → Meta') Meta ABS. No crunch, sign is only found using exact methods.
511006	^addtSIGN	(Meta → Meta') Meta SIGN.
513006	^addtARG	(Meta → Meta') Meta ARG.
12D006	^addtXROOT	(Meta2 Meta1 → Meta') Meta XROOT. XROOT(o2,o1) is o1^[1/o2], compared to o2^o1.
12F006	^addtMIN	(Meta2 Meta1 → Meta') Meta MIN.
131006	^addtMAX	(Meta2 Meta1 → Meta') Meta MAX.

Addr.	Name	Description
133006	^addt<	(Meta2 Meta1 → Meta') Meta <.
135006	^addt<=	(Meta2 Meta1 → Meta') Meta <=.
137006	^addt>	(Meta2 Meta1 → Meta') Meta >.
139006	^addt>=	(Meta2 Meta1 → Meta') Meta >=.
13B006	^addt==	(Meta2 Meta1 → Meta') Meta ==.
13D006	^addt!=	(Meta2 Meta1 → Meta') Meta !=.
13F006	^addt%	(Meta2 Meta1 → Meta') Meta %.
141006	^addt%CH	(Meta2 Meta1 → Meta') Meta %CH. Meta2*(1+Meta'/100)=Meta1.
143006	^addt%T	(Meta2 Meta1 → Meta') Meta %T.
145006	^addtMOD	(Meta2 Meta1 → Meta') Meta MOD.
147006	^addtTRNC	(Meta2 Meta1 → Meta') Meta TRNC.
149006	^addtRND	(Meta2 Meta1 → Meta') Meta RND.
14B006	^addtCOMB	(Meta2 Meta1 → Meta') Meta COMB.
14D006	^addtPERM	(Meta2 Meta1 → Meta') Meta PERM.
14F006	^addtOR	(Meta2 Meta1 → Meta') Meta OR.
151006	^addtAND	(Meta2 Meta1 → Meta') Meta AND.
153006	^addtXOR	(Meta2 Meta1 → Meta') Meta XOR.
506006	^addtCONJ	(meta → meta') Meta complex conjugate.
523006	^addtLN	(Meta → Meta') Meta LN.

Addr.	Name	Description
535006	^addtCOS	(Meta → Meta') Meta COS.
537006	^addtSIN	(Meta → Meta') Meta SIN.
539006	^addtTAN	(Meta → Meta') Meta TAN.
53B006	^addtSINACOS	(meta → meta') If meta stands for x, meta' stands for sqrt[1-x^2].
53C006	^addtASIN	(Meta → Meta') Meta ASIN.
53E006	^addtACOS	(Meta → Meta') Meta ACOS.
540006	^addtATAN	(Meta → Meta') Meta ATAN.
542006	^addtSINH	(Meta → Meta') Meta SINH.
544006	^addtCOSH	(Meta → Meta') Meta COSH.
546006	^addtTANH	(Meta → Meta') Meta TANH.
549006	^addtATANH	(Meta → Meta') Meta ATANH.
54C006	^addtASINH	(Meta → Meta') Meta ASINH.
54F006	^addtACOSH	(Meta → Meta') Meta ACOSH.
551006	^addtSQRT	(Meta → Meta') Meta SQRT.
554006	^addtSQ	(Meta → Meta') Meta SQ.
556006	^addtINV	(Meta → Meta') Meta INV.
558006	^addtEXP	(Meta → Meta') Meta EXP. Does not apply EXP[-..]=1/EXP[..].
559006	^xSYMEXP	(Meta → Meta') Meta EXP. Applies EXP[-..]=1/EXP[..].

Addr.	Name	Description
55A006	`^addtD->R`	`( Meta → Meta' )` Meta D→R.
55C006	`^addtR->D`	`( Meta → Meta' )` Meta R→D.
55E006	`^addtFLOOR`	`( Meta → Meta' )` Meta FLOOR.
560006	`^addtCEIL`	`( Meta → Meta' )` Meta CEIL.
562006	`^addtIP`	`( Meta → Meta' )` Meta IP.
564006	`^addtFP`	`( Meta → Meta' )` Meta FP.
566006	`^addtXPON`	`( Meta → Meta' )` Meta XPON.
568006	`^addtMANT`	`( Meta → Meta' )` Meta MANT.
56A006	`^addtLNP1`	`( meta → meta )` Meta LNP1.
56C006	`^addtLOG`	`( meta → meta )` Meta LOG.
56E006	`^addtALOG`	`( meta → meta )` Meta ALOG.
570006	`^addtEXPM`	`( meta → meta )` Meta EXPM.
574006	`^addtFACT`	`( Meta → Meta' )` Meta FACT.
577006	`^addtNOT`	`( Meta → Meta' )` Meta NOT.

45.1.3 Trigonometric and Exponential Operators

Addr.	Name	Description
409006	`^cos2tan/2`	`( meta → meta' )` x → (1-(tan(x/2))^2)/(1+(tan(x/2))^2)
40A006	`^1-x^2/1+x^2`	`( meta → meta' )` x → (1-x^2)/(1+x^2)
40C006	`^sin2tan/2`	`( meta → meta' )` x → 2 tan(x/2)/(1+(tan(x/2))^2)

Addr.	Name	Description
40D006	^2x/1+x^2	(meta → meta') x → 2x/(1+x^2)
40F006	^tan2tan/2	(meta → meta') x → 2 tan(x/2)/(1-(tan(x/2))^2)
410006	^addtTAN/2	(meta → meta') x → tan(x/2)
413006	^cos2tan	(meta → meta') x → 1/sqrt(1+(tan(x))^2)
415006	^sin2tan	(meta → meta') x → tan(x)/sqrt(1+(tan(x))^2)
421006	^tan2exp	(meta → meta') x → (exp(i2x)-1)/(i*(exp(i2x)+1))
423006	^asin2ln	(meta → meta') x → = i*ln(x+sqrt(x^2-1))+π/2.
425006	^acos2ln	(meta → meta') x → ln(x+sqrt(x^2-1))/i
428006	^sin/cos	(meta → meta') x → sin(x)/cos(x)
42B006	^cos*tan	(meta → meta') x → cos(x)*tan(x)
42D006	^sqrt1-sin^2	(meta → meta') x → sqrt(1-(sin(x))^2).
42F006	^sqrt1-cos^2	(meta → meta') x → sqrt(1-(cos(x))^2).
432006	^atan2asin	(meta → meta') x → asin(x/sqrt(x^2+1))
435006	^asin2atan	(meta → meta') x → atan(x/sqrt(1-x^2))
438006	^pi/2-acos	(meta → meta') x → π/2-acos(x)
439006	^pi/2-meta	(meta → meta') x → π/2-x
43B006	^pi/2-asin	(meta → meta') x → π/2-asin(x)
43E006	^atan2ln	(meta → meta') x → i/2*ln((i+x)/(i-x))
441006	^2*1-cos/sin	(meta → meta') x → (1-cos(2x))/sin(2x)

Addr.	Name	Description
443006	^2*sin/1+cos	(meta → meta') x → sin(2x)/(1+cos(2x))
445006	^sin2exp	(meta → meta') x → (e^(i*x)-1/e^(i*x))/(2i)
447006	^cos2exp	(meta → meta') x → (e^(i*x)+1/e^(i*x))/2
449006	^sinh2exp	(meta → meta') x → (e^x-1/e^x)/2
44B006	^cosh2exp	(meta → meta') x → (e^x+1/e^x)/2
44D006	^tanh2exp	(meta → meta') x → (e^2x-1)/(e^2x+1)
44F006	^asinh2ln	(meta → meta') x → ln(x+sqrt(x^2+1))
451006	^acosh2ln	(meta → meta') x → ln(x+sqrt(x^2-1))
453006	^atanh2ln	(meta → meta') x → ln((1+x)/(1-x))/2
455006	^xroot2expln	(meta1 meta2 → meta') x y → exp(ln(y)/x)
458006	^exp2sincos	(meta → meta') Returns EXP of meta as EXP[RE]*[COS+i*SIN].

45.1.4 Infinity and Undefs

Addr.	Name	Description
3A1006	^1metaundef#	(meta → meta #) Tests presence of undef in meta. # is the position of undef.
3A0006	^2metaundef#	(meta2 meta1 → meta2 meta1 #) Tests presence of undef in meta2 and meta1. # is the position of undef.
3A2006	^metaundef	(→ meta) Returns undef meta.
3A4006	^1metainf#	(meta → meta #) Finds position of infinity in meta. Metas of length>2 are considered as finite meta.

Addr.	Name	Description
3A3006	^2metainf#	(meta2 meta1 → meta2 meta1 #) Finds position of infinity in meta 2 and meta1. Metas of length>2 are considered as finite meta.
3A5006	^metainftype	(meta → #) Returns infinity type: 1 for +infinity, 2 for -infinity or 0 for unsigned.
3A6006	^unsignedinf	(→ meta) Returns unsigned infinty.
3A7006	^plusinf	(→ meta) Returns plus infinty.
3A8006	^NDROPplusinf	(ob1..obn → meta) Replaces meta by plus infinty.
3A9006	^minusinf	(→ meta) Returns minus infinty.
3AA006	^NDROPminusinf	(ob1..obn → meta) Replace meta by minus infinty.

45.1.5 Expansion and Simplification

Addr.	Name	Description
3BB006	^metasimp	(Meta → Meta) Simplifies a meta object. Non recursive rational simplification.
118007	^DISTRIB*	(meta → meta' T) (meta → meta F) Distribute *. Returns FALSE if no distribution done.
3C2006	^DISTRIB/	(meta → meta' T) (meta → meta F) Distribute /. Returns FALSE if no distribution done.
304006	^METASINEXPA	(Meta → Meta') Expands SIN.
305006	^SINEXPA+	(Meta → Meta') Expands SIN(x+y).
306006	^SINEXPA-	(Meta → Meta') Expands SIN(x-y).

Addr.	Name	Description
307006	ˆSINEXPA*	(Meta → Meta') Expands SIN(x*y). Expands if x or y is an integer.
308006	ˆSINEXPA*1	(Meta2 Meta1 → Meta') Expands SIN(x*y). Meta1 is assumed to be an integer.
30A006	ˆMETACOSEXPA	(Meta → Meta') Expands COS.
30B006	ˆCOSEXPA+	(Meta → Meta') Expands COS(x+y).
30C006	ˆCOSEXPA-	(Meta → Meta') Expands COS(x-y).
30D006	ˆCOSEXPA*	(Meta → Meta') Expands COS(x*y).
30E006	ˆCOSEXPA*1	(meta2 meta1 → Meta') Expands COS(x*y). meta1 represents an integer.
310006	ˆMETAEXPEXPA	(Meta → Meta') Expands EXP.
311006	ˆEXPEXPA+	(Meta → Meta') Expands EXP(x+y).
312006	ˆEXPEXPA-	(Meta → Meta') Expands EXP(x-y).
313006	ˆEXPEXPA*	(Meta → Meta') Expands EXP(x*y).
314006	ˆEXPEXPANEG	(Meta → Meta') Expands EXP(-x).
315006	ˆEXPEXPA*1	(Meta2 meta1 → Meta') Expands EXP(x*y). meta1 represents an integer.
317006	ˆMETALNEXPA	(Meta → Meta') Expands LN.
318006	ˆLNEXPA*	(Meta → Meta') Expands LN(x*y).
319006	ˆLNEXPA/	(Meta → Meta') Expands LN(x/y).
31A006	ˆLNEXPAˆ	(Meta → Meta') Expands LN(xˆy).

Addr.	Name	Description
31E006	^METATANEXPA	(meta → tan[meta]) Expands tan[meta].

45.1.6 Tests

Addr.	Name	Description
39A006	^metafraction?	(Meta → Meta flag) Tests if meta is a fraction of integers.
3BC006	^metapi?	(Meta → Meta#) Tests presence of π in a meta. # is the last occurence of π or 0.
3BD006	^metaCOMPARE	(Meta2 Meta1 → Meta2 Meta1 #) Comparison of 2 meta. # =0 if undef # =1 if $>$ # =2 if $<$ # =3 if = Assumes generic situation, e.g. X^2 $>$ 0 in real mode. Look below STRICTmetaCOMPARE for a more careful comparison.
3BE006	^STRICTmetaCOMPARE	(Meta2 Meta1 → Meta2 Meta1 #) Comparison of 2 meta. # =0 if undef # =1 if $>$ # =2 if $<$ # =3 if = Unlike metaCOMPARE it does not assume generic situation.
3C3006	^metareal?	(meta → meta flag) Tests if IM[meta]==0.

Chapter 46
Polynomials

The entries in this chapter deal with computation with Polynomials.

46.1 Reference

46.1.1 Computation with Polynomials

Addr.	Name	Description
118006	^QAdd	(o1 → o2+o1) Adds two polynomials.
119006	^RADDext	(o2 o1 → o2+o1) Internal +. This is the same entry as ^QAdd.
117006	^SWAPRADD	(o2 o1 → o1+o2) SWAP, then QAdd.
115006	^QSub	(o2 o1 → o2-o1) Subtracts two polynomials.
116006	^RSUBext	(o2 o1 → o2-o1) Internal -. This is the same entry as ^QSub.
114006	^SWAPRSUB	(o2 o1 → o1-o2) SWAP, then QSub.
111006	^QMul	(Q1 Q2 → Q) Multiplication of polynomials with extensions.
112006	^RMULText	(Q1 Q2 → Q) Multiplication of polynomials with extensions. This is the same entry as ^QMul.
110006	^SWAPRMULT	(Q1 Q2 → Q) SWAP, then ^QMul.
11C006	^QDiv	(o2 o1 → o2/o1) Internal /.

Addr.	Name	Description
11B006	^RDIVext	(o2 o1 → o2/o1) Internal /. This is the same entry as ^QDiv.
11A006	^SWAPRDIV	(o2 o1 → o1/o2) SWAP, then QDiv.
0D9006	^QMod	(Q, Z → Q mod Z)
113006	^RASOP	(n1/d1 n2/d2 → d1*d2 n1*d2 n2*d1) Used by RADDext and RSUBext for rational input.
11F006	^RP#	(o2 # → o2^#) Internal power (not for matrices).
120006	^MPext	(ob # prg* → ob^#) General power with a specified multiplication program.
123006	^RPext	(o2 o1 → o2^o1) Tries to convert o1 to an integer to call RP#, otherwise x^ext.
108006	^DISTDIVext	(P Q → quo mod T) (P Q → P Q F) Euclidean division. Assumes P and Q have integer coefficientes. Returns FALSE if sparse short division fails.
3E5006	^PTAYLext	(P, r → symb) Taylor for polynomials.
15B006	^CARCOMPext	(Q1/Q2 → Q1'/Q2') Extracts leading coefficients for the first variable from a rational polynomial.
3EE006	^QDivRem	(ob2 ob1 → quo mod) Polynomial Euclidean division of 2 objects. Dispatchs to DIV2LISText for list polynomials.
3EF006	^DIV2LISText	(Z0 l1 l2 → div mod) Euclidean division, l1 and l2 are list polynomials. Test first if l1=l2, then tries fast division, if it fails switch to SRPL division.
3F8006	^PDIV2ext	(A B → Q R) Step by step Euclidean division for univar poly.
3F9006	^PSetSign	(P1 P2 → sign[P2]*P1) Sets sign of P1 according to leading coeff of P2.
3C4006	^ModExpa	(Zn Fraction → Fraction modulo Zn)

Addr.	Name	Description
3C5006	ˆModAdd	`( Q1 Q2 Zn → Z )` Modular addition. Z = Q1+Q2 (mod Zn).
3C6006	ˆModSub	`( Q1 Q2 Zn → Z )` Modular subtraction. Z = Q1-Q2 (mod Zn).
3C7006	ˆModMul	`( Q1 Q2 Zn → Z )` Modular multiplication. Z = Q1*Q2 (mod Zn).
3C8006	ˆModDiv	`( Z1 Z2 Zn → Z )` Modular division. Z = Z1/Z2 (mod Zn).
3C9006	ˆModDiv2	`( Q1 Q2 Zn → quo mod mod' )` Modular division. mod' = Q1 mod Q2 mod Zn. If Q1 and Q2 are integers, Q1 mod Q2 mod Zn is always 0.
3CA006	ˆModInv	`( Z Zn → Z' )` Modular inversion. Z' = INV(Z) (mod Zn). `NONINTERR` if GCD[Z,Zn] $\neq$ 1 or if Z = 0 (otherwise the results would be unpredictable).
3CB006	ˆModGcd	`( Q1 Q2 Zn → Q' )` Modular GCD.

46.1.2 Factorization

Addr.	Name	Description
08E006	ˆBerlekampP	`( P #prime → P F / P Lf #prime T )` Berlekamp's algorithm for finding modular factors of a univariate polynomial.
08F006	ˆBerlekamp	`( P → P F / P Lf #prime T )` Berlekamp's algorithm for finding modular factors of a univariate polynomial with a leading frontend for finding linear factors faster. The input polynomial must be square free, otherwise the polynomial is not fully factored. Due to memory restrictions byte sized coefficients are used and the following restrictions were imposed: prime$<$128 and degree$<$256. If the conditions are not met `FALSE` is returned. BCD: prime$\leq$97.

Addr.	Name	Description
0A8006	^ALG48FCTR?	(P → [meta cst_coeff TRUE | P FALSE]) Factorizes square-free polynomial in Erable format.
0A9006	^MFactTriv	(P → meta-factor P') Extracts all trivial power factors of P.
0AA006	^CheckPNoExt	(P → P flag) Checks that P does not contain any DOCOL (i.e. extensions).
0AB006	^PPP	(P → PP PC) Computes primitive polynomial and content of non-const P with respect to X1. The results are trimmed (provided P was).
0AC006	^PFactor	(P → Lfk Z) Does a complete factorization of P. The result is trimmed.
0AD006	^PSqff	(P → Lfk) Square-free and trivial factorization, including integer content, of P taken positive. Factors of same power are not necessarily merged or adjacent, but all Fi's are square-free.
0AE006	^PHFctr	(P → Lf) Heuristic factorization of polynomial taken positive. LAM FullFact? must be bound. If LAM FullFact? is TRUE, a full factorization is done. If it is FALSE, only square-free and trivial factorization is done.
0AF006	^PHFctr1	(P → Lf) Heuristic factorization of primitive polynomial. LAM FullFact? must be bound. If TRUE, a full factorization is done. When FALSE, only a square-free and trivial factorization are done.
0B0006	^PHFctr0	(P → Lf) Heuristic factorization of primitive square-free non constant polynomial.
0D8007	^P2P#	(P → P' #) Extracts trivial power of poly. P must be a valid poly (if list, begin with a non zero coeff).

Addr.	Name	Description

0B1006	^DeCntMulti	`( R → L )`
		Transforms list with count into simple list.
		`R = { {f1 #k1} ... {fn #kn} }`
		`L = { f1 f1 .. fn fn }.`

0B2006	^DoLS	`( L S F → L' )` Applies program F(Li,S) to every elem of L.

0B3006	^PNFctr	`( Z → Lf )`
		Factorization of positive integer as polynomial.
		`Lf = {} if Z is 1`
		`Lf = { {Z1 #k1} ... {Zn #kn} }`
		`o/w.`

0B4006	^PSQFF	`( P → Lsqff )` Computes the square-free factorization of primitive P. The result is trimmed (provided P was).
0B5006	^LiftZAdic	`( p z F → L )` Lift n-1 z-adic factorization into n factorization.
0B6006	^LFCProd	`( C L → C P )` Calculates combination product.
0B7006	^UFactor	`( P → Lf )` Factorization of a square free primitive univariate polynomial.
0B8006	^UFactor1	`( P → Lf )` Factorization of a square free primitive univariate polynomial of degree > 2.
0B9006	^MonicLf	`( Lfp p → Lfp' )` Converts true modular factorization to monic factorization by dividing by the leading coefficient of factor 1.
0BA006	^DemonicLf	`( Lfp lc p → Lfp' )` Converts monic modular factorization to true modular factorization by multiplying factor1 by lcoeff.

Addr.	Name	Description

0BB006	^LiftLinear	`( #root1 .. #rootn #n → )`
		Lifts modular roots of a polynomial to find linear factors of a univariate polynomial.
		`Lflin = list of found true factors`
		`Lfplin' = remaining linear factors`
		`P' = remaining polynomial`
		Assumes `UFactor` lambda variables available and uses them for input and output.

0BC006	^LiftGeneral	`( → )` Lifts factorization mod p to factorization mod p^k where p^k exceeds the factor bound for succesful true factor extraction. Assumes `UFactor` lambda variables.
0BD006	^UFactorDeg2	`( P → Lf )` Factorization of a degree 2 polynomial. Polynomial is univariate, square free and primitive.
0BE006	^CombineFac	`( P Lfp p → Tf Tfp )` Combines modular factors to true factors. P is the polynomial to factor, Lfp is the list of modular factors, and p the modulo. The entry returns the a list of found true factors (Tf) and the list of modular factors for each true factor (Tfp)
0BF006	^CombProd	`( lc Lfp p Cb → F )` Calculates modular combination.

0C0006	^CombInit	`( #r → Cb )`
		Inits modular combination list to value
		`{ 1 0 0 0 .. }.`

0C1006	^CombNext	`( Cb → Cb' flag )` Gets next possible modular combination. Assumes Cb is valid and is in tempob area.
0C2006	^RmCombNext	`( Lf Cb → Lfrm Lf' Cb' flag )` Removes next possible combination after a successful combination has been found, and remove the used factors from the factor list.
0C3006	^PFactTriv	`( P → P' Lf )` Extracts all trivial power factors of P.

Addr.	Name	Description
0C4006	^VarFactor	(P #var → P #n) Calculates what power of the given variable is a factor in P.
0C5006	^PFactPowCnt	(P → P Lk flag) Calculates trivial power factors in P. flag is TRUE if any of the powers is nonzero.
0C6006	^PDivLk	(P Lk → P') Divides polynomial by its trivial powers.
282006	^FEVIDENText	(P → meta-fact cst coeff) Real mode: full factorization over the integer Complex mode: find all 1st order factors of P.

46.1.3 General Polynomial Operations

Addr.	Name	Description
09B006	^ONE{}POLY	(ob → {ob} ob1 → Q) Replaces ONE{}N for polynomial building.
09C006	^TWO{}POLY	(ob1 ob2 → Q) Replaces TWO{}N for polynomial building.
09D006	^THREE{}POLY	(ob1 ob2 ob3 → Q) Replaces THREE{}N for polynomial building.
09E006	^TWO::POLY	(ob1 ob2 → ::) Replaces 2Ob>Seco for polynomial building.
09F006	^::POLY	(Meta → ::) Replaces ::N for polynomial building. As opposed to the regular ::N code, we do pop the binary number. This is enforced by the entry to the common polyxml code.
0A0006	^{}POLY	(Meta → Q) Replaces {}N for polynomial building. As opposed to the regular {}N code, we do pop the binary number. This allows us to enter the code here with fixed sizes, as in ONE{}POLY and TWO{}POLY.
0A7006	^>POLY	(Meta → Q) Builds polynomial.
0A1006	^>TPOLY	(P ob → P') Replaces >TCOMP for polynomial building.

Addr.	Name	Description
0A2006	^>HPOLY	(P ob → P') Replaces >HCOMP for polynomial building.
0A3006	^>TPOLYN	(P ob1 .. obn #n → P') Improved >TCOMP for polynomial building.
0A4006	^>HPOLYN	(P ob1 .. obn #n → P') Improved >HCOMP for polynomial building.
0A5006	^MKPOLY	(#n #k → P) Makes polynomial of nth variable to the power k.
2AB006	^MAKEPROFOND	(ob # → {{{...{o}...}}}) Embedds ob in the given number of lists.
4F4006	^TRIMext	(Q → Q') Removes unnecessary zeros from polynomial.
4F5006	^PTrim	(ob → ob') Trims polynomial.
0A6006	^ONE>POLY	(Q → Q') Increases variable depth. Constants (Z,Irr,C) are not modified.
302006	^TCHEBext	(zint → P) Tchebycheff polynomial. If zint>0 then 1st kind, if <0 then second kind.
3DE006	^LRDMext	(P # → []) Left ReDiMension. Adds 0 to the left of polynomial to get a symbolic vector of lenght #+1.
3DF006	^RRDMext	({} # → {}) Right ReDiMension: like LRDM but 0 at the right and {}.
3E0006	^DEGREext	({} → degre) Degree of a list-polynomial.
3E1006	^FHORNER	(P/d r → P[X]_div_[X-r]/d r P[r]/d) Horner scheme.
3E2006	^HORNext	(P r → P[X]_div_[X-r] r P[r]) Horner scheme.
3E4006	^MHORNext	(P r → P[X]_div_[X-r] r P[r]) Horner scheme for matrices.

Addr.	Name	Description
3E6006	ˆLAGRANGEext	(M → symb) Lagrange interpolation. Format of the matrix is [[x1 .. xn] [f(x1) .. f(xn)]] Returns a polynomial P such that P(xi)=f(xi)
10F007	ˆRESULTANT	(P1 P2 → P) Resultant of two polynomials. Depth of P is one less than depth of P1 and P2.
110007	ˆRESULTANTLP	(res g h P1 P2 → +/-res g' h' P1' P2') Subresultant algorithm innerloop.
111007	ˆRESPSHIFTQ	(P Q → P') Resultant of P and Q shifted. gcd[Q(x-r),P(x)]!=1 equivalent to r root of P' P' has same depth than P and Q.
112007	ˆADDONEVAR	(P → P') Adds one variable just below the main var. works for polynomial, not for fractions.
0CF007	ˆSHRINKEVEN	(P → P') Changes var Y=Xˆ2 in an even polynomial.
0D1007	ˆSHRINK2SYM	(N D → N' D') Shrinks 2 polynomials using symmetry properties.
0D2007	ˆSHRINKSYM	(N → N') Shrinks 1 polynomial using symmetry properties. Degree of N must be even. If it is odd then N should be divided by X+1.
0D3007	ˆSHRINK2ASYM	(N D → N' D') Shrinks 2 polynomials using antisymmetry properties.
0D4007	ˆSHRINKASYM	(N → N') Shrinks 1 polynomial using antisymmetry properties. Degree of N must be even. If it is odd then N should be divided by X+1.
103006	ˆPNMax	(P → Z) Gets the coefficient of P with max norm.
161006	ˆSWAPNDXF	(Qden Qnom → symb) Builds a symbolic from rational polynomial.

Addr.	Name	Description
162006	ˆNDXFext	(Qnom Qden → symb) Builds a symbolic from rational polynomial.
163006	ˆSWAPFXND	(symb ob → ob Qnom Qden) Converts symbolic to rational polynomial.
164006	ˆFXNDext	(symb → Qnom Qden) Converts symbolic to rational polynomial.
3D7006	ˆREGCDext	(a b → d u v au+bv=d)
3D8006	ˆEGCDext	(a b → d u v au+bv=d) Bezout identity for polynomials.
0EA006	ˆPEvalFast?	(Z Pn → Z Pn F / Pn[Z] T) Attempts to evaluate Pn at X1=Z using fast register arithmetic. Fails if any of the following is true: Pn is not sunivariate; Z is polynomial after all; Z size is too big for register; Any overflow occurs during Horner evaluation.
10E007	ˆFLAGRESULTANT	(symb1 symb2 → symb) Resultant of two polynomials in symbolic form.

46.1.4 Tests

Addr.	Name	Description
10B006	ˆUnivar?	(P → P flag) Tests if polynomial is univariate.
10C006	ˆSUnivar?	(P → P flag) Tests if polynomial is univariate and the coefficients are bounded by register size.
0CC007	ˆPOLYPARITY	(poly → Z) Tests if a polynomial (internal rep) is even/odd/none. Z=1 if even, -1 if odd, 0 if neither even nor odd.
0D6007	ˆPOLYSYM	(P → Z) Tests symmetry of coefficients of polynomial. Z=1 for symmetric, -1 for anti, 0 otherwise.
0D7007	ˆPOLYASYM	(P → Z) Tests "antisymmetry" of coef of polynomial. Z=1 for symmetric, -1 for anti, 0 otherwise.

Chapter 47
Root Finding

In this chapter you will find entries related to finding roots of equations.

47.1 Reference

47.1.1 Root Finding and Numerical Solvers

Addr.	Name	Description
272006	ˆMULMULText	({} % → {}') Multiplies multiplicities in a factor list by coeff.
274006	ˆMETAMM2	(meta % → meta') Multiplies by % all multiplicities of meta.
275006	ˆCOMPLISText	({} → {}')
276006	ˆMETACOMPRIM	(Meta → Meta') Suppresses multiple occurrances of the same factor by adding corresponding multiplicities.
278006	ˆMETACOMP1	(f1...fk-1 mk-1 meta-res mk fk # → f1...fk-1 mk-1 meta-res)
279006	ˆADDLISText	({} %n ob → {}') Adds ob with multiplicity %n to the list. Checks if ob is in {}.
27A006	ˆDIVISext	(ob → {divisors}) Returns list of divisors of ob.
27B006	ˆFACT1ext	(symb-poly → Lvar Q {}) {} is the list of root/multiplicity of sym with respect to the current variable.
27C006	ˆFACTOext	(symb → Lvar Q {}) {} is the list of factors/multiplicity of symb.
27D006	ˆZFACTO	(C → {} C Lfact)

Addr.	Name	Description
27E006	^SOLVext	(symb → {}) Numeric solver for univariate polynomials. The list contains the roots without multiplicity.
27F006	^FRND	(ob → ob')) Float rounding for %%, C%% or list of either type. Used by SOLVext to reconstruct factors.
280006	^BICARREE?	(P #5 → meta cst_coeff T) (P #5 → P #5 F) (P # → P # F) Searches if P is a bisquared 4-th order equation. Returns a meta of factors and the multiplying coeff in that case.
281006	^REALBICAR	(f1 #1 coef → meta rest T)
113007	^IROOTS	(P → list) Finds integer roots of a polynomial.
283006	^EVIDENText	(P → meta cst_coeff) Returns the roots of a polynomial P. Calls the numeric solver.
284006	^EVIDSOLV	(P → meta cst_coeff) Returns the roots of a 1st, 2nd order and some other poly. Calls the numeric solver if exact solving fails.
285006	^DEG2ext	(P → {}) Returns the roots of a 2nd order polynomial.
286006	^METADEG2	(P → P meta) Returns the roots of a 2nd order polynomial. P must be of order 1 or 2.
287006	^METADEG1	(P → P meta) Returns the roots of a 1st order polynomial. P must be of order 1.
288006	^DEG1	(f → r) Root of a first order factor. f is one level depth deeper than r.
289006	^FDEG2ext	(P → meta-fact cst_coef) Returns factors of a 2nd order polynomial and the corresponding multiplying coefficient. tests for 1st order polynomial.

Addr.	Name	Description
28B006	^RACTOFACext	`( r → n d )` Converts root to factor. Factor is n/d, one level depth deeper than r.
28C006	^FACTORACext	`( f → r cst_coef )` Converts a factor to a root, solving 1st order factor. f and cst_coef are one level depth deeper than r.
28D006	^RFACText	`( ob # → {} intob meta )` {} is the list of variables. Meta is made of roots or factors of numerator (N) or denomenator (D) or both (N/D), depending on #. `ZERO` for roots N/D; `ONE` for roots N; `TWO` for roots D with numeric solver call; `THREE` for roots D without num. solver call; `FOUR` for factors N/D; `FIVE` for factors N; `SIX` for factors D with numeric solver call; `SEVEN` for factors D without num.solver call.
28E006	^RFACT2ext	`( ob {} # → {} intob meta )` Like `RFACText`, but the list of variables is given.
28F006	^RFACTSTEP3	`( ob → meta-fact )` Partial square-free factorization w.r.t. the main variable. Extract trivial factors Etape 3 ob → meta-fact.
290006	^RFACTSTEP5	`( %m on → add-to-meta-res )` Factorization of a square-free polynomial.
291006	^METASOLV	`( pn cst_coeff → meta cst_coeff )` Non-integer factorization (sqrt extensions and numeric). multiplicty is in LAM 5,.
293006	^METASOLV2	`( cst_coeff p → fr1 %m [fr2 %m] # cst_coeff )` Returns roots/factors of 1st and 2nd order polynomials.
294006	^METASOLV4	`( cst1 f1 ... fk #k cst2 → fr1 %m ... frn %m #2k cst_coeff )` Returns factors or convert to roots if needed. #k=1,2 or 4, fk are of order 1 or 2.

Addr.	Name	Description
295006	ˆADDMULTIPL	(meta cst_coeff → meta' cst_coeff) Adds multiplicities to a meta. Multiplicity is in LAM 5.
296006	ˆFACTOOBJext	({ fact mult } flag prg* prgˆ → ob) Rebuilds an object from its list of factors (flag=TRUE) or roots (flag=FALSE) using prg* to multiply and prgˆ to take multiplicity power.
093006	ˆALG48MSOLV	(Lp → Lidnt Lsol) Calculates Groebner basis multivar solution. LAM3 must be bound to Lvar and LAM4 to Lidnt.
094006	ˆGMSOLV	(Lp → meta-sol) Calculates Groebner basis multivar solutions. LAM1 must be bound to the number of vars A solution is a list { o1 ... on } where #n=LAM1 ok embedded in k-1 lists is the value of the k-th var ok may be undef.
095006	ˆGBASIS	(Lp → G) Calculate Groebner basis. `G = { 1 } if no solutions` `G = { 0 } if identically true.`
096006	ˆGSOLVE	(Lp → Lg) Calculate factorized Groebner basis. `Lg = { Lg1 Lg2 .. Lgn }` `Lgi = independent solution (probably)` `Lg = {} if no solutions` `Lg = { { 0 } } if identically true.`
097006	ˆGFACTOR	(Lp fctr? → Lg) Calculate Groebner basis or factorized Groebner basis. Redundant bases are not removed.
099006	ˆREDUCE	(p G → q) Reduces polynomial with respect to given basis.

Addr.	Name	Description
09A006	^FASTREDUCE	(r P → q T / r P F) Assembly version of REDUCE for polynomials with short coefficients. Returns FALSE if an overflow occurs during the reduction. Assumes r is a genuine polynomial (not constant). Assumes G is not empty. Assumes G does not contain zeros (is trimmed).
37D006	^ROOTM2ROOT	({}/V → V') Transforms list of root/multiplicites to vector of roots.
0F2007	^PASCAL_NEXTLINE	({} → {}') Finds next line in the Pascal triangle.
0F3007	^DELTAPSOLVE	(Q → P) Solves P(x+1)-P(x)=Q(x). Internal polynomial function.

Chapter 48
Calculus Operations

The entries in this chapter are related to several aspects of Calculus, such as limits, derivates, partial fraction expansions and Laplace transformations.

48.1 Reference

48.1.1 Limits and Series Expansion

Addr.	Name	Description
46F006	^SYMTAYLOR	(symb id %/z → symb) Taylor series expansion around point 0 (McLaurin's series) with regard to given variable, and of the given order.
471006	^TRUNCDL	(DL-l reste-l → truncated_DL) Series expansion truncation.
472006	^LIMSERIES!	(expression X=a|X %|zint →) a lim DL-l rest-l num-l/deno-l equiv-l lvar # Series expansion. #=1 for X=a-h or X=-1/h.
477006	^LIMIT!	(symb → DL-l reste-l num-l/deno-l equiv.-l lim. lvar flag) lim. = { symf direction }
478006	^LIMSTEP1!	(symb → { DL-l reste-l num-l/deno-l equiv.-l } flag)
47C006	^LIMLIM!	(# lvar equiv-l → lvar lim)
47F006	^LIMCMPL!	(reste-1-l reste-2-l → reste-l)
480006	^LIMEQUFR!	(n/d # → n/d-l equiv %)
481006	^LIMEQU!	({} # → {} / {}-equiv-l {}-equiv-l { # # # })

Addr.	Name	Description

483006	^LIM+-!	(DL1...DLn #n op → DL flag)
		DL = { DL-l reste-l num-l/deno-l equiv-l }.

48C006	^LIMDIVPC!	(#ordre num-l deno-l → num-l deno-l)
48E006	^LIMPROFEND!	(num deno #prof → num deno)
490006	^LIM%#!	(num-l deno-l {%...%} → num-l' deno-l' #prof {%...%})
49E006	^LIM#VARX!	(lvar lvar → #varx)
4A1006	^HORNEXP!	(lim lvar X-l reste-l → lvar DL reste-l)
4B6006	^VARCOMP!	(var1 var2 → flag)
4BA006	^VARCOMP32!	(var → 0:)
4BD006	^LIMVALOBJ!	(ob lvar → symb)
4BE006	^LIMVAL!	(ob → coeff val)
4BF006	^EQUIV!	({} lequiv → equiv ordre)
4C0006	^LVARXNX2!	(ob → ob lvarx lvarnx)
4C2006	^FindCurVar	(symb → symb) Sets a new current var if needed.
4C3006	^LIMVAR!	(symb → symb lvar)
15C006	^RISCH13	({}/{}' → {}'') Assuming {}' has length 1, divides all elements of {} by this element. Used by `RISCHext` and by SERIES to have a nicer output of series.

48.1.2 Derivatives

Addr.	Name	Description
3DC006	^PDer	({} → der)
1A1006	^DERIVext	(ob id → ob') (ob sym → ob') (ob V → V') Calculates the derivative of the object. For a list argument calculates the gradient with respect to the variables in the list. If the variable is a symbolic, the first variable in it is used. Note that the gradient is a vector quantity, thus the result is returned as a list.

Addr.	Name	Description
1A3006	^DERIVIDNT	(ob id → ob') Main entry point for derivative with respect to a identifier.
1A4006	^DERIVIDNT1	(ob → ob') Main entry point for derivative with respect to the identifier stored in LAM1.
1A5006	^DERIV	(symb → symb') Derivative of symb with respect to the variable stored in LAM1.
1A6006	^METADERIV	(Meta → Meta') Derivative of Meta object.
1BD006	^METADER&NEG	(Meta → Meta') Meta derivative and negate.
1A9006	^METADER+	(Meta&+ → Meta') Meta derivative of addition.
1AA006	^METADER-	(Meta&- → Meta') Meta derivative of subtraction.
1AB006	^METADER*	(Meta&* → Meta') Meta derivative of multiplication.
1AC006	^METADER/	(Meta&/ → Meta') Meta derivative of division.
1AD006	^METADER^	(Meta&^ → Meta') Meta derivative of power.
1AE006	^METADERFCN	(Meta → Meta') Meta derivative of a function.
1AF006	^METADERDER	(symb_id_; sym_fcn_; xDER #3 → Meta') Meta derivative of a derivative of a function.
1B0006	^METADERI4	(Meta → Meta') Meta derivative of a defined integral.
1B1006	^METADERI3	(Meta → Meta') Meta derivative of an undefined integral.
1B2006	^METADERIFTE	(Meta → Meta') Meta derivative of IFTE.
1B4006	^METADEREXP	(Meta → Meta') Meta derivative of EXP.
1B5006	^METADERLN	(Meta → Meta') Meta derivative of LN.

Addr.	Name	Description
1B6006	^METADERLNP1	(Meta → Meta') Meta derivative of LNP1.
1B7006	^METADERLOG	(Meta → Meta') Meta derivative of LOG.
1B8006	^METADERALOG	(Meta → Meta') Meta derivative of ALOG.
1B9006	^METADERABS	(Meta → Meta') Meta derivative of ABS.
1BA006	^METADERINV	(Meta → Meta') Meta derivative of INV.
1BB006	^METADERNEG	(Meta → Meta') Meta derivative of NEG.
1BC006	^METADERSQRT	(Meta → Meta') Meta derivative of SQRT.
1BE006	^METADERSQ	(Meta → Meta') Meta derivative of SQ.
1BF006	^METADERSIN	(Meta → Meta') Meta derivative of SIN.
1C0006	^METADERCOS	(Meta → Meta') Meta derivative of COS.
1C1006	^METADERTAN	(Meta → Meta') Meta derivative of TAN.
1C2006	^METADERSINH	(Meta → Meta') Meta derivative of SINH.
1C3006	^METADERCOSH	(Meta → Meta') Meta derivative of COSH.
1C4006	^METADERTANH	(Meta → Meta') Meta derivative of TANH.
1C5006	^METADERASIN	(Meta → Meta') Meta derivative of ASIN.
1C6006	^METADERACOS	(Meta → Meta') Meta derivative of ACOS.
1C7006	^METADERATAN	(Meta → Meta') Meta derivative of ATAN.
1C8006	^METADERASH	(Meta → Meta') Meta derivative of ASINH.
1C9006	^METADERACH	(Meta → Meta') Meta derivative of ACOSH.

Addr.	Name	Description
1CA006	^METADERATH	(Meta → Meta') Meta derivative of ATANH.
1B3006	^DERARG	(meta-symb → arg1 ... argk der1 ... derk #k op) Finds derivative of arguments.
1CB006	^pshder*	(Meta1 Meta2 → Meta2&Meta1'&*) Meta derivative utility.
1CC006	^SQRTINVpshd*	(Meta1 Meta2 → Meta2&SQRT&INV&Meta1'&*) Meta derivative utility.

48.1.3 Integration

Addr.	Name	Description
07F007	^ODE_INT	(symb idnt → symb) Integration with addition of a constant.
2C5006	^IBP	(u'*v u → u*v -u*v') Internal integration by parts. If u is a constant return INTVX(u'*v)+u. If stack 2 is a list it must be of the form { olduv u'*v } then olduv will be added to u*v at stack level 2. This permits multiple IBP in algebraic mode, e.g. IBP(ASIN(X)^2,X) IBP(ANS(1),sqrt(1-X^2)) IBP(ANS(1),C) the last step with an integral containing a cst C.
2D0006	^PREVALext	(symb inf sup x → symb|x=sup - symb|x=inf) Evaluates an antiderivative between 2 bounds Does not check for discontinuities of symb in this interval.
2D1006	^WARNSING	(symb inf sup vx → symb inf sup vx) Warns user for singularity.
2D2006	^INText	(symb x → int[$,x, symb, xt]) Return unevaluated integral.

Addr.	Name	Description
2D3006	^INT3	(f(x) x y → F(y) where F'=f) Undefined integration. No limit for underdetermined form.
3DD006	^INTEGRext	({} → prim)

48.1.4 Partial Fractions

Addr.	Name	Description
3D2006	^PARTFRAC	(o → symb) Partial fraction expansion of o with respect to the current variable.
3D3006	^INPARTFRAC	(o list → symb) Partial fraction expansion of o. lvar must be bound to LAM2, list is =lvar if o is in external format. list is `NULL{}` if o is still in internal format.

48.1.5 Differential Equations

Addr.	Name	Description
07E007	^DESOLVE	(list symb1 → list_sols) (symb symb1 → list_sols) Solves ordinary differential equation. For some ode's returned level2 is not symb1.
081007	^LDECSOLV	(2nd_member char_eq → solution) Linear differential equation with constant coefficients.
082007	^LDEGENE	(eq. carac → sol generale)
083007	^LDEPART	(2nd membre, eq carac → eq. carac, sol part)
084007	^LDSSOLVext	(V M → V') M is the matrix of the system. V is the vector of the 2nd members.
085007	^ODETYPESTO	(type →) Store ode type in variable ODETYPE.

Addr.	Name	Description

086007	ˆODE_SEPAR	`( symb → symb symb-y symb-x T )`
		`( symb → symb F )`
		Tries to separate symb as a product of a function of y and a function of x.

48.1.6 Laplace Transformation

Addr.	Name	Description
087007	ˆLAPext	`( symb → symb' )` Laplace transform for polynomial*exp/sin/cos. Returns LAP() for unknown transforms.
088007	ˆILAPext	`( symb → symb' )` Inverse Laplace transform for rational fractions. Delta functions for the integral part.
08B007	ˆILAPEXP	`( ck rk → ck*exp[rk*x] )`

Chapter 49
Summation

In this chapter, you will find the main entries related to summation, and also some sub-routines used by those entries.

49.1 Reference

Addr.	Name	Description
0F9007	^SUM	(sym idnt → sym) Internal SUM. The variable can be specified.
0FB007	^SUMVX	(sym → sym) Internal SUMVX. Works always with respect to the current variable.
0FD007	^RATSUM	(sym → sym) Discrete rational sum.
0FE007	^FTAYL	(f shift → f') Taylor shift for rational fractions.
0FF007	^CSTFRACTION?	(ob → ob flag) Taylor shift for rational fractions. Returns TRUE if ob is a cst fraction.
104007	^HYPERGEO	(symb → symb) Tests and does hypergeometric summation.
100007	^NONRATSUM	(z/symb → symb) Discrete summation (hypergeometric case).
103007	^meta_cst?	(meta → meta flag) Tests for meta to be cst with respect to current var.
108007	^ZEILBERGER	(f(n,k) n k d → C T) (f(n,k) n k d → F) Zeilberger algorithm * NOT IMPLEMENTED YET*.

Addr.	Name	Description
109007	^SYMPSI	(sym → Psi(x)) Digamma function.
10B007	^SYMPSIN	(sym int → Psi(x,n)) Digamma function.
11C007	^%%PSI	(%%x → %%) Digamma function.
10D007	^IBERNOULLI	(#/zint → Q) Bernoulli numbers.
0D9007	^NDEvalN/D	(num deno n d → num' deno') Evals list poly over a list fraction.
0DA007	^PEvalN/D	(P n d → num d #) Evals list poly over a list fraction.
3C1006	^vgerxssSYMSUM	(Meta2 Meta1 → meta) Symbolic sum with tests for two zints. lam'sumvar bound to 'id/lam' and lam'sumexpr to 'expr'.

Chapter 50
Modular Operations

The entries in this chapter are related to modular arithmetic and other modular operations.

50.1 Reference

50.1.1 Modulo Operations

Addr.	Name	Description
252006	ˆFLAGFACTORMOD	(symb → symb) FACTOR modulo.
253006	ˆMFACTORMOD	(M → M') FACTOR modulo for amtrices.
256006	ˆLIFCext	({contfrac} → fraction) Converts continued fraction to rational.
0E1006	ˆPEvalMod	(Q Z Zn → Q') Computes value of polynomial mod Zn.
0E2006	ˆQAddMod	(Q1 Q2 Zn → Q') Polynomial addition modulo Zn.
0E3006	ˆQSubMod	(Q1 Q2 Zn → Q') Polynomial subtraction modulo Zn.
0E4006	ˆQMulMod	(Q1 Q2 Zn → Q') Polynomial multiplication modulo Zn.
0E5006	ˆQDivMod	(Q1 Q2 Zn → Qquo Qrem) Polynomial division modulo Zn. In regular division the coefficients in the remainder can increase very quickly to tens of digits, thus it is important to normalize the coefficients whenever possible.

Addr.	Name	Description
0E6006	^QInvMod	(Q Zn → Q') Polynomial inversion modulo Zn.
0E7006	^QGcdMod	(Q1 Q2 Zn → Q') Polynomial GCD modulo Zn for univariate polynomials. The result is made monic.
4C5006	^ISOL1	(symb id → id symb')
4C6006	^ISOLALL	(symb id → id {}) Internal SOLVE.
4C7006	^ISOL2ext	(symb id → symb') (symb id → {}) Like ISOL1 if isolflag is set. Otherwise returns the list of all found solutions.
4C8006	^BEZOUTMSOLV	(Lpoly Lidnt → Lidnt sols) If no extension in Lpoly, calls ALG48 GSOLVE Otherwise, solves by Bezout "Gaussian" elimination. In the latter case, if system seems underdetermined, Lidnt is truncated. Then the system must be exactly determined and polynomials must be prime together.
4C9006	^ROOT{}N	(meta of roots → list of roots) Drops tagged roots.
4CA006	^MHORNER	(poly-l {r1...rk} # → P[r1...rk]) Top-level call. Poly-l might be a matrix.
4CB006	^MHORNER1	(P { r } → P[..r..])
4CC006	^SQFFext	(Q → { F1 mult1 .. Fn multn })
4CD006	^MSQFF	(Q → F1 mult1 .. Fn multn #2n) Full square-free factorization of object. The result is given as a Meta object.
4CE006	^%1TWO	(ob → ob %1 #2) Square free factorization of unknown (?) object. See MSQFF.
4CF006	^MZSQFF	(Z → Z1 mult1 .. Zn multn #2n) Full factorization of an integer.

Addr.	Name	Description
4D0006	^MZSQFF1	(Meta curfac %n newfac T → Meta curfac %n+1) (Meta curfac %n newfac F → Meta' newfac %1) Adds integer factor to factor list. If the factor is the same as the last time, only the multiplicity is increased.
4D2006	^MLISTSQFF	(P → Meta) Full square-free factorization of a polynomial with a recursive call on the GCD of all coefficients.
4D3006	^METASQFFext	(P-list → S1 %1 ..Se-1 %e-1 %e ee Te Re) Square-free factorization.
4DE006	^LIDNText	(ob → {}) Gets list of all ids present in ob.
4DF006	^LVARXNXext	(symb → symb x lvarnx lvarx) Finds variable of symb depending on current variable and other variable. Using LVAR is impossible here because of sqrt.
4E0006	^ISPOLYNOMIAL?	(ob → flag) Returns TRUE if symb is polynomial with respect to current variable.
4E1006	^2POLYNOMIAL?	(symb1 symb2 → symb1 symb2 flag) Returns TRUE if symb1 and symb2 are polynomial with respect to current variable.
4E2006	^VXINDEP?	(symb → symb flag) Returns TRUE if symb is independent of current variable.
4E4006	^RLVARext	(ob → {}) Recursive search of all variables.
4E5006	^LLVARDext	(o → #depth o lvar)
4E6006	^VXLVARext	(symb → symb lvar)
4E7006	^LVARext	(ob → ob {}) List of variables. Square roots *are* included in the list of rational operators.

Addr.	Name	Description
4E8006	^VX>LVARext	(ob → ob {}) Like LVARext but the current variable is added using >HCOMP. Square roots *are* included in the list of rational operators.
4E9006	^VX>	({} → {}') If VX is in the list then moves it to the beginning of the list. Otherwise does nothing.
4EA006	^VX!	({} → {}) If VX is in the list then moves it at the beginning. Otherwise VX is added to the beginning of the list.
4EC006	^LIDNTLVAR	(symb lidnt → symb lidnt lvar) lvar is the list of variables in symb, but elements of lidnt are moved to the beginning of lvar.
4ED006	^LISTOPRAC	(→ {}) Returns the list of rational operator with sqrt appended to the list.
4EE006	^LISTOPext	(→ {}) List of basic "rational" operators without square root.
4EF006	^LISTOPSQRT	(→ {}) List of basic "rational" operators with square root.
4F0006	^LVARDext	(ob listop → lidnt) (Meta listop → lidnt) Determines list of variables in ob (or Meta) using the given list of basic "rational" operators.
4F2006	^DEPTHext	(ob → #) Returns the max number of embedded lists in ob.
4F3006	^DEPTHOBJext	(objet # → depth)
4F6006	^TRIMOBJext	(ob → ob ') Trims object.

Addr.	Name	Description
4F7006	^NEWTRIMext	(Q → Q) Recursively tests if Q is a list of one constant element. This is much faster than TRIMOBJext and sufficient for the output of programs which are trimmed on the fly.
4F8006	^>POLYTRIM	(meta → {}) Equivalent to {}POLY TRIMOBJext.
4F9006	^ELMGext	(ob → ob') Trims small numbers (less than epsilon).
0E9006	^IsV>V?	(v1 v2 → flag) Returns TRUE if v1 is lexicographically after v2.
0EB006	^PZadic	(Q Z → Q')
104006	^LISTMAXext	(P → P Z T depth) (P → P ? F #0) Step 1 for gcdheu: Returns FALSE if gcdheu can not be applied (e.g. if P contains irrquads). Returns TRUE otherwise, Z is the max of all integers of P or 2*max if there are complex in P.
0EC006	^GCDHEUext	(A B → a b c pr[pgcd] A'/G' B'/G' flag) Heuristic GCD.

Chapter 51
Sign Tables

A sign table is a list which describes thes sign of a expression in different intervals of a parameter. The list has an odd number of elements and looks like this:

`{ value1 sign1.2 value2 sign2.3 ...signN-1.N valueN }`

The values are key values of the parameter, usually $-\infty$, $+\infty$, and the locations of singularities or zeros in the expression. The values must be ordered and can be numbers or symbolic expressions. The signs show the sign of the expression in the interval between the adjacent values. Signs are `'-'`, `'+'`, and `'?'` (if the sign is unknown). To compute the sign table of an expression with respect to the current variable, use the entry `SIGNE1ext`. For example, the sign table of the expression '$X^2 - 1$' is

{ $-\infty$ '+' -1 '-' 1 '+' $+\infty$ }

Below is a list of the entries related to sign tables.

51.1 Reference

Addr.	Name	Description
237006	ˆSIGNE	(symb $\rightarrow$ sign) Compute the sign table of the expression with respect to the current variable. Internal version of the UserRPL command SIGNTAB.
0DC007	ˆSIGNE1ext	(expr $\rightarrow$ sign) Sign table of a polynomial or rational expression.
0DE007	ˆSIGNUNDEF	($\rightarrow$ sign) Returns undefined sign table.
0DF007	ˆSIGNPLUS	($\rightarrow$ sign) Returns always positive sign table.

Addr.	Name	Description
0E0007	^SIGNMOINS	(→ sign) Returns always negative sign table.
0E1007	^SIGNELN	(sign → sign) Returns ln of a sign table.
0E2007	^SIGNEEXP	(sign → sign') Returns exp of a sign table.
0E3007	^SIGNESIN	(sign → sign') Returns sin of a sign table.
0E4007	^SIGNECOS	(sign → sign') Returns cos of a sign table.
0E5007	^SIGNETAN	(sign → sign') Returns tan of a sign table.
0E6007	^SIGNEATAN	(sign → sign') Returns atan of a sign table.
0E7007	^SIGNESQRT	(sign → sign') Returns sqrt of a sign table.
0E8007	^SUBSIGNE	(sign min max → sign') Truncates a sign table.
0E9007	^SIGNERIGHT	(sign ob → sign') Places ob at the end of a sign table.
0EA007	^SIGNELEFT	(sign ob → sign') Places ob at the beginning of a sign table.
0EB007	^>SIGNE	(sign → sign') Prepends { -infinity ? } to a sign table.
0EC007	^SIGNE>	(sign → sign') Appends { ? +infinity } to a sign table.
0ED007	^SIGNMULText	(sign1 sign2 → sign') Multiplies two sign tables.
0DB007	^POSITIFext	(ob → ob flag) Tries to determine if ob is positive. In internal representation, this depends on increaseflag so that x-1 is positive if increaseflag is cleared, negative otherwise, because x is assumed to tend to +infinity or zero.
0EE007	^ZSIGNECK	(ob → ob flag) Returns sign of an expression. Error if unable to find sign.

Addr.	Name	Description
0F0007	ˆZSIGNE	(ob → zint) Returns sign of an expression. zint=1 for +, -1 for -, 0 for undef. Expression does not need to be polynomial/rational.
0F1007	ˆzsigne	(meta → zint) Returns sign of a meta symbolic. zint=1 for +, -1 for -, 0 for undef. Expression does not need to be polynomial/rational.
07D007	ˆCHECKSING	(symb inf sup vx → symb inf sup vx flag) Checks for singularities in expr.

Chapter 52
Errors

The CAS error messages all have numbers starting with DE. You can get a full list in Appendix E.

Entries ^ERABLEERROR and ^GETERABLEMSG add DE00 to the message number, so you only specify the last two digits of the message number. You can naturally use the error commands described in Chapter 22 with the CAS errors, using the full error numbers.

52.1 Reference

Addr.	Name	Description
57E006	^ERABLEERROR	(# →) Calls CAS Error.
57D006	^GETERABLEMSG	(# → $) Get string in erable messages table.
090006	^ErrInfRes	Error 305h Generates "Infinite Result" error.
091006	^ErrUndefRes	Error 304h Generates "Undefined Result" error.
092006	^ErrBadDim	Error 501h Generates "Invalid Dimension" error.
57F006	^CANTFACTOR	Error DE1Ch Generates "Unable to find factor" error.
580006	^TRANSCERROR	Error DE20h Generates "Not reducible to a rational expression" error.
581006	^NONUNARYERR	Error DE21h Generates "Non unary operator" error.
582006	^INTERNALERR	Error DE26h Generates "CAS internal error" error.

Addr.	Name	Description
583006	^INVALIDOP	Error DE28h Generates "Operator not implemented (SERIES)" error.
584006	^ISOLERR	Error DE2Ah Generates "No solution found" error.
585006	^NONINTERR	Error DE2Ch Generates "No solution in ring" error.
586006	^INTVARERR	Error DE32h Generates "No name in expression" error.
587006	^Z>#ERR	Error DE35h Generates "Integer too large" error.
0EF007	^SIGNEERROR	Error DE36h Generates "Unable to find sign" error.
588006	^Z<0ERR	Error DE46h Generates "Negative integer" error.
589006	^VXINDEPERR	Error DE47h Generates "Parameter is cur. var. dependent" error.
58A006	^NONPOLYSYST	Error DE49h Generates "Non polynomial systrem" error.
58B006	^COMPLEXERR	Error DE4Dh Generates "Complex number not allowed" error.
58C006	^VALMUSTBE0	Error DE4Eh Generates "Polyn. valuation must be 0" error.
58D006	^SWITCHNOTALLOWED	Error DE4Fh Generates "Mode switch not allowed here" error.
119007	^NONALGERR	Error DE50h Generates "Non algebraic in expression" error.
58E006	^ERR$EVALext	(seco → action)
58F006	^Sys1IT	(ob →) Execute object if display flag is set.

Chapter 53
CAS Configuration

The entries in this chapter provide ways to configure the CAS operations. The configurations that can be done here are the same that can be done by the user via flags or the MODES input form.

53.1 Reference

Addr.	Name	Description
08F007	^CFGDISPLAY	($\rightarrow$) Display current configuration of the CAS.
090007	^NEWVX	($\rightarrow$) Input new current variable from the user.
091007	^NEWMODULO	($\rightarrow$) Input new modulo from the user.
092007	^SWITCHON	(#flag $\rightarrow$) Asks the user if a certain mode may be switched on by toggling system flag #flag. Errors if the user does not want to switch.
093007	^SWITCHOFF	(#flag $\rightarrow$) Asks the user is a certain mode may be switched off by toggling system flag #flag. Error if the user does not want to switch.
094007	^FLAGNAME	(# $\rightarrow$ # $) Find the name of a flag.
1DC007	(^PUSHFLAGS)	($\rightarrow$) Internal version of User PUSH command: stores the current flag settings and path in the CASDIR/ENVSTK variable.

Addr.	Name	Description
1DD007	(^POPFLAGS)	($\rightarrow$) Internal version of User POP command: pops the last pushed flag settings and path from the CASDIR/ENVSTK variable.
095007	^COMPLEXON	($\rightarrow$) Turns complex mode on. Depending on system flag 120, the user is asked first.
096007	^COMPLEXOFF	($\rightarrow$) Turns complex mode off. Depending on system flag 120, the user is asked first.
097007	^EXACTON	($\rightarrow$) Turns exact mode on. Depending on system flag 120, the user is asked first.
098007	^EXACTOFF	($\rightarrow$) Turns exact mode off. Depending on system flag 120, the user is asked first.
099007	^COMPLEXMODE	($\rightarrow$) Set complex mode, refresh configuration display.
09A007	^SETCOMPLEX	($\rightarrow$) Set complex mode.
09B007	^COMPLEX?	($\rightarrow$ flag) Test complex mode.
09C007	^REALMODE	($\rightarrow$) Set real mode, refresh configuration display.
09D007	^CLRCOMPLEX	($\rightarrow$) Set real mode.
09E007	^EXACTMODE	($\rightarrow$) Set exact mode, refresh configuration display.
09F007	^SETEXACT	($\rightarrow$) Set exact mode and gcd mode.
0A0007	^NUMMODE	($\rightarrow$) Set numeric mode, refresh configuration display.
0A1007	^CLREXACT	($\rightarrow$) Clear exact mode.
0A2007	^EXACT?	($\rightarrow$ flag) Test exact mode.

Addr.	Name	Description
0A3007	^STEPBYSTEP	($\rightarrow$) Set step by step flag, refresh display.
0A4007	^NOSTEPBYSTEP	($\rightarrow$) Clear step by step flag, refresh display.
0A5007	^VERBOSEMODE	($\rightarrow$) Set verbose mode, refresh configuration display.
0A6007	^SILENTMODE	($\rightarrow$) Set silent mode, refresh configuration display.
0A7007	^RECURMODE	($\rightarrow$) Set recursive mode, refresh configuration display.
0A8007	^NONRECMODE	($\rightarrow$) Set nonrecursive mode, refresh configuration display.
0A9007	^PLUSAT0	($\rightarrow$) Set positive mode, refresh configuration display.
0AA007	^SETPLUSAT0	($\rightarrow$) Set positive mode.
0AB007	^PLUSATINFTY	($\rightarrow$) Set positive infinity mode, refresh configuration display.
0AC007	^CLRPLUSAT0	($\rightarrow$) Set positive infinity mode.
0AD007	^SPARSEDATA	($\rightarrow$) Set full data mode, refresh configuration display.
0AE007	^FULLDATA	($\rightarrow$) Set sparse mode, refresh configuration display.
0AF007	^RIGORMODE	($\rightarrow$) Set rigorous mode, refresh configuration display.
0B0007	^SLOPPYMODE	($\rightarrow$) Set sloppy mode, refresh configuration display.

Addr.	Name	Description
0B1007	^SLOPPY?	(→ flag) Test sloppy mode.
1D2006	^SAVECASFLAGS	(→) Saves CAS flags and current var.
1D4006	^RESTORECASFLAGS	(→) Restore CAS flags and current var.
1D5006	^CASFLAGEVAL	(→) Execute next runstream object with flag protection.
0C2007	^RCLMODULO	(→ Z) Fetch MODULO from the home directory.
0C3007	^RCLPERIOD	(→ sym) Fetch PERIOD from the home directory.
0C4007	^RCLVX	(→ id) Fetch VX from home directory.
0C5007	^STOVX	(ob →) Store object in VX.
0C6007	^STOMODULO	(ob →) Store object in MODULO.
0C7007	^RCLEPS	(→ %) Fetch EPS from home directory.
0C8007	^ISIDREAL?	(id → id id T) (id → id F) Test if id is in the REALASSUME list.
0C9007	^ADDTOREAL	(id →) Add `idnt` to the list of real var.
0CA007	^RESETCASCFG	(→) Reset CAS config.
1D0006	^VERNUMext	(→ %version) CAS version number.

Chapter 54
CAS Menus

The entries in this chapter return the built-in menus of CAS commands, or do some other actions related to menus. For general information on menus, turn to Chapter 37.

54.1 Reference

Addr.	Name	Description
1D1006	^MENUXYext	(#2 #1 → {}) Make list of Erable commands between the given numbers.
08D007	^MENUext	($6...$1 →) If the CAS quiet flag is not set, displays the six strings as menu keys. Otherwise does nothing.
0B2007	^MENUCHOOSE?	(→ prg flag) Return best CHOOSE command.
0B3007	^MENUCHOOSE	({} →) Offers a selection to the user. If Flag -117 is set, only installs a menu. If not, offer a CHOOSE box.
0B4007	^MENUGENE1	(→ {}) Menu for CAS.
0B5007	^MENUBASE1	(→ {}) Base algebra menu.
0B6007	^MENUCMPLX1	(→ {}) Complex operations menu.
0B7007	^MENUTRIG1	(→ {}) Trigonometric operations menu.
0B8007	^MENUMAT1	(→ {}) Matrix operations menu.

Addr.	Name	Description
0B9007	^MENUARIT1	($\rightarrow$ {}) Arithmetic operations menu.
0BA007	^MENUSOLVE1	($\rightarrow$ {}) Solver menu.
0BB007	^MENUEXPLN1	($\rightarrow$ {}) Exponential and logarithmic operations menu.
0BC007	^MENUDIFF1	($\rightarrow$) Differential calculus menu.

Chapter 55
Internal Versions of User RPL Commands

The entries in this chapter are the closest correspondents to User RPL commands.

55.1 Reference

Addr.	Name	Description
218006	ˆISPRIME	(z/% → %0/%1) Internal ISPRIME.
1D6006	ˆFLAGEXPAND	(symb → symb') Internal xEXPAND. Expands symbolic expression.
1D8006	ˆFLAGFACTOR	(symb → symb') (z → symb) Internal xFACTOR. Factors symbolic or number.
1D9006	ˆFLAGLISTEXEC	(symb {} → symb') Internal xSUBST for the case that level 1 is an array or a matrix.
1DA006	ˆFLAGSYMBEXEC	(symb symb' → symb'') Internal xSUBST for the case that level 1 is a symbolic.
1DB006	ˆFLAGIDNTEXEC	(symb id → symb') Internal xSUBST for the case that level 1 is an id or a lam.
1DC006	ˆFLAGINTVX	(symb → symb') Internal xINTVX.

Addr.	Name	Description
1DD006	^DERVX	(symb → symb') Internal xDERVX.
1DE006	^SOLVEXFLOAT	(% → {}) Internal xSOLVEVX for a float.
1DF006	^SYMLIMIT	(symb symb' → symb'') Internal xLIMIT for scalars.
1E0006	^FLAGMATRIXLIMIT	([] symb → []') Internal xLIMIT for matrices.
1E1006	^TAYLOR0	(symb → symb') Internal xTAYLOR0.
1E2006	^FLAGSERIES	(symb id z → {} symb') Internal xSERIES.
1E4006	^PLOTADD	(symb →) Internal xPLOTADD.
1E5006	^FLAGIBP	(symb1 symb2 → symb3 symb4) Internal xIBP.
1E6006	^FLAGPREVAL	(symb1 symb2 symb3 → symb4) Internal xPREVAL. Evaluates symb1 at the points symb2 and symb3 and takes the difference.
1E7006	^MATRIXRISCH	([] id → symb') Internal xRISCH for matrix arguments.
1E8006	^FLAGRISCH	(symb id → symb') Internal xRISCH for non-matrix argumetns.
1E9006	^FLAGDERIV	(symb id → symb') Internal xDERIV.
1EA006	^FLAGLAP	(symb → symb') Internal xLAP.
1EB006	^FLAGILAP	(symb → symb') Internal xILAP.
1EC006	^FLAGDESOLVE	(symb symb' → symb'') Internal xDESOLVE.
1ED006	^FLAGLDSSOLV	(symb1 symb2 → symb3) Internal xLDEC.
1EF006	^FLAGTEXPAND	(symb → symb') Internal xTEXPAND.
1F0006	^FLAGLIN	(symb → symb') Internal xLIN.

Addr.	Name	Description
1F1006	^FLAGTSIMP	(symb → symb') Internal xTSIMP.
1F2006	^FLAGLNCOLLECT	(symb → symb') Internal xLNCOLLECT.
1F3006	^FLAGEXPLN	(symb → symb') Internal xEXPLN.
1F4006	^FLAGSINCOS	(symb → symb') Internal xSINCOS.
1F5006	^FLAGTLIN	(symb → symb') Internal xTLIN.
1F6006	^FLAGTCOLLECT	(symb → symb') Internal TCOLLECT.
1F7006	^FLAGTRIG	(symb → symb') Internal xTRIG.
1F8006	^FLAGTRIGCOS	(symb → symb') Internal xTRIGCOS.
1F9006	^FLAGTRIGSIN	(symb → symb') Internal xTRIGSIN.
1FA006	^FLAGTRIGTAN	(symb → symb') Internal xTRIGTAN.
1FB006	^FLAGTAN2SC	(symb → symb') Internal xTAN2SC.
1FC006	^FLAGHALFTAN	(symb → symb') Internal xHALFTAN.
1FD006	^FLAGTAN2SC2	(symb → symb') Internal xTAN2SC2.
1FE006	^FLAGATAN2S	(symb → symb') Internal xATAN2S.
1FF006	^FLAGASIN2T	(symb → symb') Internal xASIN2T.
200006	^FLAGASIN2C	(symb → symb') Internal xASIN2C.
201006	^FLAGACOS2S	(symb → symb') Internal xACOS2S.
206006	^STEPIDIV2	(z1 z2 → z3 z4) Internal xIDIV2.
207006	^FLAGDIV2	(symb1 symb2 → symb3 symb4) Internal xDIV2.

Addr.	Name	Description
208006	^FLAGGCD	(symb1 symb2 → symb3) Internal xGCD for the case with two symbolica arguments.
209006	^PEGCD	(symb1 symb2 → symb3 symb4 symb5) Internal xEGCD for polynomials.
20B006	^ABCUV	(symb1 symb2 symb3 → symb4 symb5) Internal polynomial xABCUV.
20C006	^IABCUV	(z1 z2 z3 → z4 z5) Internal integer xIABCUV.
20D006	^FLAGLGCD	({} → {} symb) Internal xLGCD.
20E006	^FLAGLCM	(symb1 symb2 → symb3) Internal xLCM.
20F006	^FLAGSIMP2	(symb1 symb2 → symb3 symb4) Internal xSIMP2.
210006	^FLAGPARTFRAC	(symb → symb') Internal xPARTFRAC.
211006	^FLAGPROPFRAC	(symb → symb') Internal xPROPFRAC.
212006	^FLAGPTAYL	(P(X) r → P(X+r)) Internal xPTAYL.
213006	^FLAGHORNER	(symb1 symb2 → symb3 symb4 symb5) Internal xHORNER.
214006	^EULER	(z → z') Internal xEULER.
216006	^FLAGCHINREM	(A1 A2 → A3) Internal xCHINREM.
217006	^ICHINREM	(A1 A2 → A3) Internal xICHINREM.
219006	^SOLVE1EQ	(symb id → {}) Internal xSOLVE for single equations.
21A006	^SOLVEMANYEQ	([] []' → {}'') Internal xSOLVE for arrays of equations.
21B006	^ZEROS1EQ	(symb id → {}) Internal xZEROS for single equations.

Addr.	Name	Description
21C006	^ZEROSMANYEQ	([] []' → {}) Internal xZEROS for arrays of equations.
21D006	^FCOEF	([] → symb) Internal xFCOEF.
21E006	^FROOTS	(symb → []) Internal xFROOTS.
21F006	^FACTORS	(symb → {}) Internal xFACTORS.
220006	^DIVIS	(symb → {}) Internal xDIVIS.
223006	^rref	(M → A M') Internal xrref.
229006	^MADNOCK	(M → symb1 []' []'' symb3) Internal xMAD.
22A006	^SYSTEM	([] []' → []'' {} []''') Internal xLINSOLVE.
22B006	^VANDERMONDE	({} → M) Internal xVANDERMONDE.
22C006	^HILBERTNOCK	(z → M) Internal xHILBERT.
22E006	^CURL	([exprs] [vars] → []) Internal xCURL.
22F006	^DIVERGENCE	([exprs] [vars] → symb) Internal xDIV.
230006	^LAPLACIAN	([expr] [vars] → symb) Internal xLAPL.
231006	^HESSIAN	(symb A → M A' A'') Internal xHESS.
232006	^HERMITE	(z → symb) Internal xHERMITE.
233006	^TCHEBNOCK	(%degree → symb) Internal xTCHEBYCHEFF.
234006	^LEGENDRE	(z → symb) Internal xLEGENDRE.
235006	^LAGRANGE	(A → symb) Internal xLAGRANGE.
236006	^FOURIER	(symb z → C%) Internal xFOURIER.

Addr.	Name	Description
238006	^TABVAR	(symb → symb {{}} grob) Internal xTABVAR.
239006	^FLAGDIVPC	(symb1 symb2 z → symb3) Internal xDIVPC.
23A006	^FLAGTRUNC	(symb1 symb2 → symb3) Internal xTRUNC.
23B006	^FLAGSEVAL	(symb → symb') Internal xSEVAL.
23C006	^XNUM	(symb → symb') Internal xXNUM.
23D006	^REORDER	(symb id → symb') Internal xREORDER.
23E006	^USERLVAR	(symb → symb []) Internal xLVAR.
23F006	^USERLIDNT	(symb → []) Internal xLNAME.
241006	^ADDTMOD	(symb1 symb2 → symb3) Internal xADDTMOD for scalars.
242006	^MADDTMOD	(M M' → M'') Internal xADDTMOD for matrices.
243006	^SUBTMOD	(symb1 symb2 → symb3) Internal xSUBTMOD for scalars.
244006	^MSUBTMOD	(M M' → M'') Internal xSUBTMOD for matrices.
245006	^MULTMOD	(symb1 symb2 → symb3) Internal xMULTMOD.

Chapter 56
Miscellaneous

In this chapter are listed the entries that did not fit in any of the previous chapters.

56.1 Reference

56.1.1 Verbose Mode Display Routines

Addr.	Name	Description
579006	^Verbose1	($ →) Display message on line 1 if verbose mode on.
57A006	^Verbose2	($ →) Display message on line 2 if verbose mode on.
57B006	^Verbose3	($ →) Display message on line 3 if verbose mode on.
57C006	^VerboseN	($ # →) Display message on given line if verbose mode on.

56.1.2 Evaluation

Addr.	Name	Description
257006	^EvalNoCKx*	(ob ob' → ob'')
258006	^EvalNoCKx+	(ob ob' → ob'')
259006	^EvalNoCKx-	(ob ob' → ob'')
25A006	^EvalNoCKx/	(ob ob' → ob'')
25B006	^EvalNoCKx^	(ob ob' → ob'')
25C006	^EvalNoCKxCHS	(ob → ob')
25D006	^EvalNoCKxINV	(ob → ob')
25E006	^EvalNoCKxMOD	(ob ob' → ob'')

Addr.	Name	Description
25F006	^EvalNoCKxPERM	(ob ob' → ob'')
260006	^EvalNoCKxCOMB	(ob ob' → ob'')
261006	^EvalNoCKxOR	(ob ob' → ob'')
262006	^EvalNoCKxAND	(ob ob' → ob'')
263006	^EvalNoCKxXOR	(ob ob' → ob'')
264006	^EvalNoCKxXROOT	(ob ob' → ob'')
265006	^TABVALext	(fnct x {} → {}') Table of values.

56.1.3 Conversion

Addr.	Name	Description
266006	^TOLISText	(o1..on #n → Lvar Q1..Qn) Convert meta of symbolic objects to internal form.

267006	^FROMLISText	(Lvar Meta L → L')
		Conversion of elements of Meta objec to user format. Meta does not contain the #n number of element. L is the list of depth of the elements of Meta. For example to convert a polynomial, a vector and a matrix:
		Lvar = { X }
		Meta = { Z1 Z3 }
		{ Z0 Z1 }
		{ { Z1 { Z1 Z0 } } }
		L = { #0 #1 #2 }
		L' = { 'X+2' { 0 1 } { { 1 X } } }.

56.1.4 Qpi

Addr.	Name	Description
074007	^QPI	(ob → ob') Internal xXQ.
073007	^QpiZ	(ob → symb) Calls ^Qpi% and converts the resulting (real) integers into zints.

Addr.	Name	Description
075007	^QpiSym	(symb → symb') Internal xXQ for symbolics.
076007	^QpiArry	([] → []') Internal xXQ for arrays. Converts each element of the array.
077007	^QpiList	({} → {}') Internal xXQ for lists. Converts each element of the list.
078007	^Qpi	(%/C% → symb) Internal xXQ for real and complex numbers.
079007	^Qpi%	(% → symb) xXQ for reals, but does not convert numbers to zints.
07A007	^GetRoot	(%' → %' %'') Tries to find a square number which is a factor of the argument. The algorithm only tries numbers smaller than 1024^2-1 and assumes that % is an integer. The returned results are such that %=(%')^2*%''. For numbers which do not contain a square factor, %'=1 and %''=%.
07B007	^Approx	(% → %' %'') Approximates a real number with a fraction. Returns numerator %' and denominator %''. The accuracy of the approximation is determined by the current display format.

56.1.5 Infinity

Addr.	Name	Description
2E2006	^INFINIext	(→ '∞')
2E3006	^MINUSINFext	(→ '-∞')
2E4006	^PLUSINFext	(→ '+∞')
2E5006	^?ext	'?' Pushed the undefined symbolic.
2E6006	^POSINFext	(symb → symb #) Returns #1 if the symbolic contains '∞'.
2E1006	^TESTINFINI	(ob → ob flag) Test if object contains infinity.

Addr.	Name	Description
2E7006	^POSUNDEFext	(symb → symb #) Returns #1 if the symbolic contains the undefined symbolic '?'.

56.1.6 Built-In Constants

Addr.	Name	Description
2EA006	^pi	(→ 'π')
2EB006	^metapi	(→ π #1)
2F1006	^meta-pi	(→ π xNEG #2)
2E8006	^pisur2	(→ 'π/2')
2F2006	^metapi/2	(→ π 2 x/ #3)
2E9006	^pisur-2	(→ '-π/2')
2F4006	^meta-pi/2	(→ π 2 x/ xNEG #4)
2F3006	^metapi/4	(→ π 4 x/ #3)
2F5006	^meta-pi/4	(→ π 4 x/ xNEG #4)
2F6006	^pifois2	(→ '2*π')
2EC006	^'xPI	(→ xPI)
2F9006	^base_ln	(→ 'e')
2FA006	^meta_e	(→ e #1)
2EE006	^'xi	(→ xi)
2ED006	^metai	(→ i #1)
2EF006	^ipi	(→ 'i*π')
2F0006	^metaipi	(→ i π x* #3)
2F8006	^metapi*2	(→ π 2 x* #3)
2F7006	^deuxipi	(→ '2*i*π')

56.1.7 List Application

Addr.	Name	Description
3F0006	^DIVOBJext	({o1...on} ob → {o1/ob...on/ob}) Division of all elements of a list by ob. Tests if ob=1.
3F2006	^LOPDext	({o1...on} ob → {o1/ob...on/ob}) `LOPDext` calls `QUOText` for the division, unlike DIVOBJ which calls `RDIVext`.

Addr.	Name	Description
269006	^LOP1ext	({} ob binop → {}') Applies non-recursively << ob binop >> to the elements of the list.
26A006	^LOPAext	({} ob binop → {}') Applies recursively << op binop >> to the elements of the list (not the list elements themselves).
10F006	^LOPMext	(ob {} → {}') Multiplies each element of the list by the given object.
45F006	^LISTEXEC	(ob {} → ob') (ob {} → {}') The list should be of the form { 'X=1' 'Y=2' ... } in the first case or { 'X=1' 'X=2' } in the second case. In the first case, all occurences of X in ob are replace by 1, or Y by 2, etc. In the second case ob is evaluated with X=1, X=2 successively.
460006	^LISTEXEC1	({} objet → {}')
461006	^SECOEXEC	({} prog → {}) Executes prog on each element of ob.
268006	^PFEXECext	(symb prg → symb)
26B006	^LISTSECOext	(composite → composite) Applies 1LAM non-recursively to all elements of the list.
26D006	^CK1TONOext	(ob → ob') Applies prg to ob, recursively for lists. prg is fetched from runstream.

56.1.8 Irrquads

Addr.	Name	Description
167006	^TYPEIRRQ?	(ob → flag) Is ob an irrquad?
168006	^DTYPEIRRQ?	(ob → ob flag) DUP, then ^TYPEIRRQ?.
165006	^QXNDext	(irrq → a b c) b=0 and c=1 if stack level 1 is not an irrq.

Addr.	Name	Description
166006	^NDXQext	(a b c → irrq)
2D8006	^IRRQ#ULTIMATE	(ob → # c) Finds « depth and returns ultimate c of an irrq.
508006	^QCONJext	(irrq → irrq') irrq-conjugate of an irrq. This is *not* the complex conjugate.
509006	^QABSext	(irrq → irrq sign) Finds the sign of an irrq. Work always if irrq is made of Z.
51A006	^QNORMext	(Zirr → a^2-b*c^2) Irrq-norm of an irrquad. This is *not* the complex modulus.
4D4006	^SECOSQFFext	(:: x<< a b c x>> → { fact1 mult1 ... factn multn }) Factorization of irrquads and Gauss integers.
124006	^PREPARext	(o1 o2 → a1 b1 c1 a2 b2 c2) Returns irrquad decomposition of o1 and o2. with either c1=c2 or c1 and c2 have no factors in comon. c1<c2, ordering handled by LESSCOMPLEX? is made by type, then by CRC.
2DA006	^LISTIRRQ	(ob {} → {}') Add the C-part of all irrquads of object to the list.

56.1.9 Miscellaneous

Addr.	Name	Description
3E7006	^PSEUDOPREP	(o2 o1 → o2*a1.n^ o1 a1.n^)
3FB006	^HSECO2RCext	(ob → ob') Conversion of constants from internal to user form.
3FC006	^SECO2CMPext	(seco → symb) Back conversion of complex. polarflag should be disabled if not at the top level of rational expressions.

Addr.	Name	Description
3FF006	^VALOBJext	(# {..{Q}..} {var1..varn} → {..{ob}..}) Back conversion of objects embedded at depth # in lists. Simplifies var1..varn.
401006	^VAL2ext	(# {..{Q}..} {var1..varn} → {..{ob}..}) Back conversion of objects embedded at depth # in lists. Does not simplify var1..varn. Conversion is done in asc. power if positivfflag is set, which is useful for SERIES and LIMIT commands.
402006	^INVAL2	(P # → symbpoly) LAM2 must contain Lvar, # is the depth.
403006	^METAVAL2	(# Meta_list → Meta_symb) LMA2 must contain Lvar, LAM1 is modified.
404006	^VAL1	(ob → ob) LAM2 must contain Lvar, LAM1 is modified.
405006	^VAL1M	(ob → Meta_symb) LAM2 must contain Lvar, LAM1 is modified.
45C006	^IDNTEXEC	(symb idnt → symb') Tries to find idnt such that symb=0. Return a solution as an equality 'idnt=..' in symb'.
121006	^MP0	(ob → ob 1) Returns number 1 of the selected type. The symbolic/ROMPTR one looks very strange it is used to avoid infinity^0/undef^0 to return 1.
26C006	^rpnQOBJext	(ob → ob') prg is fetched from the stack. Looks for all d1, d2, ... at the beginning of the name of idnt to determine if idnt represents a derivative of a user function. Stops if at a time the stripped idnt is in the current directory. Example 'd2d1Y' returns { #2 } << >> if 'd2d1Y' is not defined and 'd1Y' is defined as << >> or { #2 #1 } 'Y' if d2d1Y d1Y and Y are not defined.

Addr.	Name	Description
29D006	^SIMPIDNT	(idnt → ob) Evaluates idnt (looks recursively for its content if defined). Does not error for circular definition, but displays a warning.
29F006	^RCL1IDNT	(idnt/lam → ob) Recursive content of an idnt. LAM1 to LAM3 must be bound.
2A7006	^SWPSIMPNDXF	(ob2 ob1 → ob1/ob2) Simplified fraction (internal).
2A8006	^SIMPNDXFext	(ob2 ob1 → ob2/ob1) Simplified fraction (internal).
2B6006	^CMODext	(C2 C1 → C1 C2_mod_C1)
2BD006	^SQFF2ext	(l1...ln #n-1 → l1'...ln' #n-1)
2BE006	^PPZ	(p → p/pgcd pgcd) ob is the gcd of all constant coefficients of P (integer, Gauss integers, irrquads with the implementation of the "gcd" for irrquads).
117007	^PPZZ	(ob → ob zint) PPZ with further check to ensure returning a zint.
2BF006	^PZHSTR	(a z → a mod z)
2C0006	^HORNER1ext	(P r → P[r])
2C1006	^PEval	(P r → P[r]) P must be a list polynomial.
2C6006	^SQRT_IN?	({} → {} flag) Returns TRUE if one element of {} is a symb containing a sqrt.
2C7006	^IS_SQRT?	(symb → flag)
2C9006	^IS_XROOT?	(symb → flag)
2CA006	^STOPRIMIT	(symb →) Stores antiderivative in PRIMIT variable.
2CB006	^CONTAINS_LN?	(symb → symb flag)
2D4006	^FOURIERext	(symb n → cn) Computes n-th Fourier coefficient of a 2 π periodic function.

Addr.	Name	Description
2D9006	^LESSCOMPLEX?	`( ob1 ob2 → ob1 ob2 flag )` Compares objects by type and then by CRC. flag is true if ob1 is less complex than ob2 (ob1>ob2). If ob1 or ob2 is an irrq, find first ultimate type of ob1 and ob2. If these ultimate types are equal sort is done by comparing the << depth.
2DD006	^TABLECOSext	`( → {} )` Table of special COS values (k*pi/12).
2DE006	^TABLETANext	`( → {} )` Table of special TAN values (k*pi/12).
101007	^LINEARAPPLY	`( symb nonrat_prg rat_prg → symb )` Applies linearity. nonrat_prg is applied for a non rational part symb → symb. rat_prg is applied for a rational part symb → symb. Linearity is applied on symb.
106007	^A/B2PQR	`( A B → P Q R )` Writes a fraction A/B as E[P]/P*Q/E[R]. Q and positive shifts of R are prime together.

107007	^GOSPER?	`( P Q R → P R Y T )`
		`( P Q R → F )`
		Solves P = Q E[Y] - R Y for Y.

0CB007	^FRACPARITY	`( fr → Z )` Tests if a fraction (internal rep) is even/odd/none. Z=1 if even, -1 if odd, 0 if neither even nor odd.
0D5007	^FR2ND%	`( fraction-l → N D % )` Extract trivial power of fraction.
4D1006	^MSECOSQFF	`( ob → Meta )` Factorization of an extension.

Part V

Appendices

Appendix A
Development Tools

You have basically two choices for developing software for the HP49G. The programs can either be written and tested on a PC, using special tools and an emulator, or you can write software directly on the HP49G.

This chapter will describe tools for the HP49G calculator that make it a suitable programming environment for System RPL development. The HP49G calculator includes a built-in compiler, disassembler and some sort of debugger (which, to say the truth, could be improved), plus some other little tools that can be of use to the System RPL programmer. However, for big programming tasks this is not enough: some other tools are necessary to make programming easier. Because of this, some third-party tools will also be described. With a good knowledge of the built-in and third-party tools, the HP49G can be used as a complete and compact programming environment. All the programs described here can be freely downloaded from The HP Software Archive, `http://www.hpcalc.org`.

The built-in programming tools you will need are, by default, not accesible to the user. They are in two libraries, which are not attached by default. Library 256 contains several useful commands for "hacking" with the calculator, and also the disassembler. Library 257 contains MASD, the compiler. You should have these libraries always attached. If you have extable installed (and you should — see section A.1), then library 256 will be automatically attached. Library 257 (MASD) does not really need to be attached, because it is possible to call MASD from library 256. Nevertheless, it is still good to have it attached.

The `STARTUP` variable is useful to configure the calculator. This variable (which must be in the `HOME` directory) contains an object to be executed after each warmstart. It can be used to set all parameters lost by a warmstart that you want to keep, or to do anything else you want. The following program will set user mode (which is lost in a warmstart); for efficient programming (and even for efficient use) it is essencial to make some key assignments. The program also attaches library 257.

```
« -62 SF 257 ATTACH »
```

A.1 The Entry Points Library

For System RPL development, the extable library is virtually indispensable. This library contains the tables of supported entry points and addresses. It is with the help of this library that you can write `DUP` and get the correct address for this command; without it, you would need to enter `PTR 3188` every time or write an equate for this command manually. In disassembly (including the System RPL stack (see section A.3)), it allows you to get the name of the commands, instead of only their addresses. Basically, this library is pretty much essential.

Transfer extable to your calculator and install it as any other library. That is all you need to do to use command names instead of addresses. Extable appears in the library menu, and it contains five user-accessible commands.

The first command, `nop`, does nothing :-). Probably, there was a command in that position before, but it was removed, and another command that does nothing was put there not to change the other rompointers.

The other four commands, fortunately, are sometimes useful :-) (if not directly then through the Emacs library). The `GETADR` command returns the address of an entry. Just put the name of the entry (a string) in level one and run it. The inverse operation is done by `GETNAME`: give it an address, and it will return the name of the entry.

If you do not know the exact name of an entry, the last two commands will help you. Put a string with the first few letters of the command in level one, run `GETNAMES` and, voilà, a list with the names of all commands that *start* with those letters is returned. The last command, `GETNEAR`, is even more powerful: give it a string, and all commands whose names *contain* that string (even if in the middle of the command) will be returned.

A.2 About Key Assignments

Even though assigning keys is not directly related to System RPL programming, we will describe here the KEYMAN library, written by Wolfgang Rautenberg (e-mail: raut@math.fu-berlin.de). The latest version is 9.2001. This library simplifies the assignment, deletion and recalling of keys, but, most importantly, allows a key to behave differently if it is pressed longer than usual or double pressed.

You will find several commands inside this library. The `A?D` command is used to assign and delete keys. To assign something to a key, put the object in level one and press `A?D` shortly. Then, press the key you want to assign to (shifts and shift-holds work, of course). The key is assigned. To delete an assignment, press `A?D` for a slightly longer time, and then the key from which you want to remove the assignment. The command `RclK` allows one to recall the assignment of any key. It works like the previous commands: press it (briefly) and then the key. A longer press will return a list of all the keys assigned.

The commands above are just other ways to do what was already possible with the built-in commands. But the real power is in the `IfE?P`, `IfD` and `IfL` commands. The first serves two functions: it allows a key to have different meanings when in edit mode and when not, or to have different meanings when in program mode and when not. To use it, put the object to be run in edit or program mode in level two, the object to be run in normal mode in level one, and press `IfE?P`. A short press will create a program that evaluates the object in level two if the calculator is in edit mode, or the object in level one if not. A longer press does the same, but the test is based on whether program entry mode is active or not.

The `IfD` and `IfL` commands are similar. To use `IfD`, put in level two the object to be run if the key is pressed twice (like with a computer mouse) — double pressed — and put in level one the object to be run if the key is pressed once. Run `IfD`, and you will have a single program that executes one of the objects according to how the key was pressed. Note that assignments produced with `IfD` will slightly delay execution on a single keypress, since the calculator must wait to see if the double press will happen or not. The command `IfL` is similar, but it allows different actions based on how long the key is pressed: you have seen this behaviour in the `A?D` command. The object to be run in a longer press is in level two.

All the `If` commands have an extra feature. Any of the two objects in the stack can be a real number in the form `rc.p`, where `r` is the row, `c` is the column and `p` is the plane (normal, left-shifted, right-shifted, left-shift-hold, etc.). In the program created with the `If` commands, these numbers will be replaced by the standard key assignment of the corresponding key. This is really useful for making assignments which do not disturb the normal function of a key but just add functionality in a special mode or keypress technique. If you give a real number that is not a valid keycode, it will be replaced by a command to make a beep.

Two other commands can sometimes be useful: `→TO?` inserts the System RPL command `TakeOver` in the beginning of the program when the key is pressed shortly. This is necessary if you want the command to be executed while the command line is active. A longer press inserts `UnlockAlpha` in the beginning of the program, useful when it is assigned to an alpha-shifted key. Finally, `K&SA` recalls the keycode and standard assignment for any key. This is used when you want to add new functionality to a key. When this standard assignment is a command in a library (that is, a ROM Pointer, also called a XLIB name), the pointer is recalled to level two, and its contents is put in level one.

In the following sections, we will show some examples of key assignments built with KEYMAN commands.

A.3 Hacking Tools

The tools described here make the life of the programmer easier. They give access to some functions which are normally not available for pure User RPL users of the calculator. First, the built-in tools in the HP49G will be described. Later, a third-party library will be described.

Before describing the built-in tools found in library 256, we will mention a flag that is very useful to System RPL programmers: flag –85. When this flag is set, the "System RPL Stack" is active: in the stack the objects are decompiled using the System RPL decompiler before being displayed. That means that, where one would see just `External` with the normal stack, the name for the entry (or `PTR` and the address, if no name is found) will be displayed, if you have the extable library (see section A.1) installed. Play with it a bit and you will see how useful it can be. Some objects (most notabily real numbers and integers) keep their usual notation, but in the interactive stack all objects are decompiled. This "System RPL Stack" is like the one produced by the command `SSTK` command of the JAZZ library for the HP48 calculators.

Probably you will be switching between the two kinds of stack display all the time. It is a good idea to assign a simple program to a key to toggle this display. A possibility is to assign it to Right-shift `MODE`. This normally is the key that marks the end of selection in edit mode. Since this key is unused when not in edit mode, it is a good example of the use of the KEYMAN library. To create this assignment, first put the program to be run when in edit mode in level two. This is easy: just use keycode 22.3. Then, write a simple User RPL (or System RPL, if you want) program to toggle flag –85 (this task if left to the reader — but read the description of OT49 in section A.3.1 first). Finally, press `IfE?P` briefly and use `→TO?` on the resulting program (because it must be able to run while in edit mode), and assign it to the key, with `A?D` or the `ASN` command.

Library 256 contains some useful tools for the programmer. This library does not show up in the library menu (because it does not have a title), but you can get a menu with its commands by typing `256 MENU`. If the library is attached (as it should be), you can type the commands, look up them in the catalog and an option will appear in the Apps menu, which says “Development lib”, giving access to all the commands in the library.

Here is a description of the commands present in the library:

Command	**Description**
`→H`	“To hex”: This converts an object into a string of hexadecimal characters. A common tool since the HP48 days to ease transfer of binary objects.
`H→`	“From hex”: This is the opposite transformation: creates an object from a string of hexadecimal characters.
`→A`	“To address”: Given an object, this command returns the address of the object, which is always a five-nibble hxs. Objects whose address is less than `# 80000h` are in ROM, and objects whose address is greater than that are in RAM.
`A→`	“From address”: This recalls the object at the specified address.
`S→H`	“String to hex”: Converts a string into its characters’ hexadecimal representation. For example, since 5A, 59 and 58 are the hexadecimal codes for X, Y and Z respectively, `"XYZ"` becomes `"8595A5"`.
`H→S`	“Hex to string”: The opposite transformation.
`→LST`	“Make list”: Creates a list from a user meta object or another composite. A user meta object is any number of objects in the stack followed by a count represented as a real number. Be careful, because this command is not sufficiently argument-protected.
`→ALG`	“Make algebraic”: Creates an algebraic object from a user meta object or another composite. This may easily result in `'Invalid Expression'`.
`→PRG`	“Make program”: Creates a program from a user meta object or another composite.
`COMP→`	“From composite”: Explodes any composite object into a user meta object.

Command	Description
→RAM	"To RAM": Dumps any ROM object into RAM. Can extract some commands for disassembly, but see section A.5 for more information.
SREV	"Reverse string": Reverses a string, very fast.
POKE	Writes data to any address in RAM. Put in level two a hxs with the address, and in level one a string of hex digits to be written at that address. This is a very easy way of destroying any "masterpiece" you have created on the calculator :-).
PEEK	Extracts raw hex digits from any address. Put the address in level two (an hxs) and the number of nibbles to get (another hxs) in level one.
APEEK	"Address peek": Like PEEK, but always gets five nibbles, returning them as a hxs.
R~SB	"Real ↔ system binary": Converts reals to bints and vice-versa.
SB~B	"System binary ↔ binary": Converts bints to hxs's, and vice-versa.
LR~R	"Long real ↔ real": Converts long reals to reals and vice-versa.
S~N	"String ↔ name": Converts strings to identifiers (global names) and vice versa.
LC~C	"Long complex ↔ complex": Converts long complexes to complexes and vice-versa.
ASM→	"From ASM": Disassembles Code objects (machine-language) into source code.
CRLIB	"Create library": A library creator. This is described in Appendix B.
CRC	Calculates the CRC. The argument is a string of hex digits.
MAKESTR	"Make string": Creates a string with the number of characters given in level one (a real number).
SERIAL	Returns a string with the internal Serial Number of the HP49.
ASM	Provides access to the MASD compiler. See section A.4 for more information.
ER	Used in conjunction with ASM. See section A.4 for more information.
→S2	Disassembles an object. See section A.5 for more information.
XLIB~	Creates a rompointer (XLIB name) from the library number (level two) and command number (level one). It also explodes rompointers into its two components.

A.3.1 Operating Tools for the HP49

Wolfgang Rautenberg is the author of a library called Operating Tools (or OT49 for short) with several commands, some of which are useful to the System RPL programmer. The latest version of this library is 3.2002.

OT49 contains a library creator and a library splitter (written by Peter Geelhoed). To split any library, just put its number in the stack and run `D↔L`.

The `DType` command displays the type of the object in level one. If that object is a rompointer (XLIB) or flashpointer, its contents is recalled (unless it is pure machine-language code) and the contents' type is displayed, with an asterisk appended.

One of the most useful commands is `3tog`. It toggles between three representations of composite objects: as a list, as a program and as a user-metaobject. This can be used to manipulate System RPL programs without actually decompiling them. `3tog` explodes a program onto the stack, you can use stack commands to rearrange things and then `3tog` again to rebuild the program.

Another very useful command is `Fl~`. It is a flag toggler. Just give the number of the system or user flag, run it, and the flag is toggled. It will also display in the header what has just been done.

The `MDA~` command compiles or decompiles an object (depending wheter the input is a string or another object).

A.4 The Compiler

The compiler included in the HP49G calculator is MASD. It is a newer version of the compiler found in the MetaKernel program for HP48G calculators. If you have already used the MetaKernel, then you can probably skip most of this section. But, even if you have never used MASD, there should be no difficulties learning how to use it. There are no big differences between MASD syntax and that of other System RPL compilers such as JAZZ (for the HP48 calculators), the HP Tools or the GNU Tools.

MASD is called with the command `ASM`. It expects a string in level one, and returns the compiled object. If there are errors, the string and a list will be put in the stack. This list is used by the `ER` command, described shortly.

The first difference to be observed from those that are coming from JAZZ or one of the PC Tools is that MASD, for some unknown reason, needs the source to end with a "@" character. All source code files must be identified with this token, or MASD will refuse to even look at them. The character must be on a line by itself, at the start of the line, and with no character after it (not even a newline). This way, it is pretty much just cumbersome. (To be useful, it would be the character marking the end of the source, but there should not be all those restrictions on its placement, and text after it should be allowed — and ignored.) However, for the Emacs RPLCPL (see section A.6), the @ acquires at least one purpose: it allows the calculator to automatically distinguish between a System RPL source file and a Uer RPL program or command line.

The other thing to note concerns the current MASD *mode*. There are two modes, selected by flag –92: Assembly Language mode (flag –92 cleared) and System RPL mode (flag –92 set). Probably, you will set flag –92 and thus MASD will be by default in System RPL mode. Then, nothing else needs to be changed to compile System RPL programs (just add the @ in the end). It is still possible to compile Assembly Language code in System RPL mode: just surround the code between CODE and ENDCODE.

If you are in Assembly Language mode, it is possible to compile System RPL code by inserting these two lines before the source:

```
!NO CODE
!RPL
```

Both are called directives. The !NO CODE directive tells MASD to compile our source as System RPL code, and not as Machine Language code. (Once more, you can insert assembly language code between CODE and ENDCODE.) It is a good idea to always put these two lines at the start of all programs even if you use System RPL mode: this way, the source can be compiled regardless of the flag settings.

Here is a simple program source ready for MASD:

```
!NO CODE
!RPL
::
  DUPTYPEZINT?
  case
    FPTR2 ^Z>R
  DUPTYPEREAL? ?SEMI
  SETTYPEERR
;
@
```

The above is the disassembly of the CKREAL entry. As you can see, it automatically converts integers to real numbers.

It is a nice idea to assign the ASM command to a key: you will need it many times.

If there was an error during compilation, the original string is put in level two, and a list is put in level one. In this case, run the ER command. It will display a list of errors for you to choose, and will jump directly to that error in the source. Correct the error, press ENTER and the choose another error, until all errors have been corrected. Then, run ASM (and ER, if necessary) again. Better yet, use the ASM2 command from library 257, which calls ASM and then, if there was any error, ER.

A.4.1 MASD and the Different Kinds of Entries

A System RPL program can call three different kinds of entries: normal entries, which point to some address in ROM, flashpointer entries, which point to a command in one of the HP49's flash banks, and rompointer entries, which point to a command in a library (built-in or not).

For supported "normal" entries, no special precautions need to be taken. You can just include the name of the command. To call an unsupported entry, you will have to use PTR <address>, where <address> is the address as listed in the tables.

For supported flashpointer entries (whose names always start with ^), you have to prefix the entry's name with FPTR2. So, to call the flashpointer command ^Z>R, you will have to include this in your program:

```
FPTR2 ^Z>R
```

An unsupported flashpointer entry is called with FPTR <bank> <cmd>. <bank> are the last three digits of the address as listed in the table (but in practice it can be no bigger than Fh), and <cmd> are the first three digits.

Calling rompointer entries (which have names starting with ~) is very similar to calling flashpointers. If it is supported, just prefix it with ROMPTR2. For unsupported entries, you have to use this syntax: ROMPTR <lib> <cmd>. <lib> are the last three digits of the address, and <cmd> the first three.

A.4.2 MASD's Special Features

The MASD compiler supports some special features that are not a part of the System RPL programming language, but that can be useful to the programmer.

The first feature eases the use of unsupported entries. You can define a name for a unsupported entry, making it behave as if an entry in extable. (This only works for normal entries, not flashpointer or rompointers.) To do that, use the following structure:

`EQU` *`name address`*

where *`name`* is the name of the entry, and *`address`* is its address. For example, the line below defines the entry `2NELCOMPDROP`, which returns the second element of a composite:

```
EQU 2NELCOMPDROP 2825E
```

With that definition, you can use `2NELCOMPDROP` instead of `PTR 2825E` to access that command. Note that this only works for normal entries.

Another way to ease the inclusion of unsupported entries (especially rompointers and flashpointers), but that is useful not only for that, are the `DEFINE`s. The structure is like this:

`DEFINE` *`name value`*

where *`name`* is a single word, and *`value`* is the rest of the line.

After that definition, whenever *`name`* is found in the source file, it will be replaced by *`value`*. So, if you are going to use the browser (see Chapter 33), it might be convenient to define this:

```
DEFINE ^Choose3 FTPR 2 72
```

so that you can simply insert `^Choose3` when you want to call the browser.

A.4.2.1 Unnamed Local Variable Binding

There is a structure that allows you to refer to local variables with names in the source, but that produces unnamed local variables, thus combining ease of use with speed.

The local variables are bound with

```
{{ name1 name2 ... nameN }}
```

After that, entering `name1` will become `1GETLAM`, `name2` will become `2GETLAM`. Preceding the name of a variable with `=` or `!` stores something in the variable, that is, `=name1` becomes `1PUTLAM`, and so on.

Pay attention to the way the names are bound: the first variable name corresponds to `1GETLAM` (that is, the object that was in level one), the second to `2GETLAM` (the object that was in level two), and so on. *This is the opposite of what JAZZ does.* It is possible, however, to get the ordering as JAZZ does, by putting the `!JAZZ` directive in the beginning of the source file.

A.4.2.2 Including Source Files

Using the `INCLUDE` pseudo-command, you can include other source files in your main program. This is like the `#include` directive in C programs.

It is used like this: `INCLUDE` *`variable`*. The contents of the named variable are read as if they were included in the source file. The included file should also end with an "`@`".

One use of this feature is to include a file with definitions of several constants or unsupported addresses.

A.5 Disassembly

As it was briefly mentioned in the description of Library 256 (see section A.3), the command →`S2` is the System RPL disassembler. It will disassemble any object in level one into its source code suitable for reassembly with MASD. Unfortunately, there are still some bugs in MASD, which prevent some disassembled objects to be correctly re-assembled. We all hope that in a newer version this bugs will be corrected.

Often, one wants to see how one of the built-in commands in the HP's ROM is built. The JAZZ library for the HP48 calculators made that easy. Unfortunately, it is difficult to do that with only the built-in tools in the HP49G. There are, however, two two libraries for this purpose: Nosy by Jurjen N. E. Boss, and CQIF, by Pierre Tardy. Both allow to extract and disassemble ROM code, and both can be used together with Emacs (see section A.6).

A.5.1 Using Nosy

Nosy, written by Jurjen N. E. Boss (j.bos@interpay-iss.demon.nl) and presently at version 4.0 is a tool to disassemble the HP49G's ROM. It is very easy to use, and, unlike CQIF?, can be easily used wihout Emacs. It also displays more information that CQIF?, such as the names of flash pointers. Because of this, it is slower than CQIF?

To use Nosy, put the entry name, pointer, an address in the stack (some other inputs are also accepted — see Nosy's documentation) and run the command `Nosy`. This will open an interactive browser where you can view the disassembled entry and browse the ROM like a hypertext document. Use the arrow keys to scroll. You can quickly view another command inside the disassembled source by moving the highlight to it and pressing ENTER or F6. This will open another browser just like the first one. To go back one level, press Backspace, and to exit press ON.

There are some other functions you can use in the interactive browser, consult the documentation for details.

A.5.2 Using CQIF?

With the help of the CQIF? (*Comment Qu'Ils Font?*) library, written by Pierre Tardy (e-mail: tardyp@iname.com), currently at version 1.7.7F, the task of disassembling built-in commands is also simplified. It contains several tools for the HP49G hacker. We will not describe everything from the library here, read its documentation if you want to know what else it can do for you.

The most useful command is `CQIF?`. This command is the basic way to disassemble some part of the HP's ROM. It accepts several kinds of inputs. If you give a string with the name of an entry, that entry is disassembled. You can put an address (a hxs), and run `CQIF?` to disassemble whatever is at that address. It will also accept the entry pointer itself, rompointers and flashpointers. To ease the disassembly of User RPL commands, you can enter the command inside a list or program (that is, enter `{ DUP }` or `« DUP »` to disassemble the User RPL command `DUP`).

`CQIF?` disassembles step-by-step, so if the command is only a pointer to another command, you need run the `CQIF?` command several times to get to the real code of the command. Just remove any unnecessary junk from the stack, keeping the last result of `CQIF?` and run it again. Eventually you will reach the command.

Another useful command in the library is `DISPATCH`. It does a virtual dispatch based on the object types. To use it, put the objects you would use as arguments to some command in the stack. Then, recall that command (probably using `CQIF?`) to level one. Run `DISPATCH`. The object that would be run for those argument types (by means of some dispatching command like `CKn&Dispatch`) is put in level one.

The other commands are not so useful to System RPL programs. But it is a nice idea to read the documentation and see what CQIF? can do for you.

A.6 The Editor, and Emacs

When you use the HP49G to develop programs, you will spend most of the time writing or changing the source code. This is done in the editor.

The HP49G editor is much better than the one in the HP48 calculators. However, it can be made even better. There are two variables that are run before entering and after leaving the editor. We will see what can be done with them. We will also describe a library that enhances the editor with some features useful in particular for programming.

Before starting the editor, the variable `STARTED` is evaluated. You can put a program in this variable to be run before editing any object. And, after leaving the editor, the `EXITED` variable is run. There are many things these variables can do. A very simple (and very useful) thing is to remove the header during editing, giving a few more lines of text. After the editor is exited, the header is restored to the default setting. It is very simple to do this: `STARTED` just needs to clear the header:

```
« 0 →HEADER »
```

And `EXITED` restores the header:

```
« 2 →HEADER »
```

Change `2` to `1` if you normally use only one line of header. Note that Emacs (see below) removes the header automatically.

For even better customization of the editor there is the Emacs library, written by one of us (CD, e-mail: dominik@astro.uva.nl) and Peter Geelhoed (e-mail: P.F.Geelhoed@student.tnw.tudelft.nl). This library gives the editor some of the features of the famous GNU Emacs editor, such as completion, automatic indentation, incremental search, regular expression search and a macro language. The latest version, at the time of this writing, is 1.10. Again, we will not describe everything in the library — see the manual for more information.

Probably the single most useful feature of the Emacs library is command completion. It is activated by the `RPLCPL` command. This is only useful in edit mode, so you will need it assigned to a key, with `TakeOver` before. If you have the KEYMAN library (see section A.2), just put a program like this in the stack:

```
« RPLCPL »
```

and run `→TO?`. Then, assign the resulting object to a key. It is a nice idea to assign it to the same key both with and without the alpha-mode on. Of course, you do not need a program. Just the rompointer (got with `{ RPLCPL } HEAD` or some similar trick) is enough, but you must still run `→TO?`.

To try it, enter the first few letters of any User RPL command. Press the key to which you assigned `RPLCPL`. If there was only one command starting with those letters, what you typed will be completed. If there were more than one, a choose box will appear from which you can select the derised command. The command line will be completed. This is something *really* useful.

Provided you have the extable library installed (as you should — see section A.1), the completion also works for System RPL command names. If the last character in the string is a `@` (as required by MASD), then System RPL completion is automatically used. As an added bonus, if you press the key to which `RPLCPL` is assigned longer, then the lookup of System RPL commands is done with `GETNEAR` (see section A.1). You can then enter `case`, ask for completion, and get all words that have `case` in the middle — not only in the begining.

Another command that sometimes is useful is `DYNCPL`. It should also be assigned to some key, and also does completion. But it looks in the file you are editing for words that start with the typed letters. It is useful for the names of local variables and such. It works in a slightly different way: press the key, and the word will be completed with the first word. To accept it, press ENTER. To abort, press ON. To search for another match, press the same key that invoked `DYNCPL`. Any other key will accept the match and execute that key.

One more command that is useful to be assigned to a key is `RPLED`. This command imitates the `ED` command in the JAZZ library: it decompiles the object in level one, opens an editor for you to edit it, and, upon exit, recompiles the object (if you are lucky, that is. If the object cannot be compiled because of some MASD bug, exit the editor with a longer-pressed ENTER. This will allow you to select not to compile the file).

`RPLED` also displays a menu with useful operations, described below. If you call `RPLED` when in edit mode, the menu is redisplayed.

To get a description of all the commands in the menu, read the documentation that comes with Emacs. Here we will present the most useful ones:

`CO..` calls `RPLCPL`. Left-shift `CO..` calls `DYNCPL`. See above for explanations of these commands.

With `|>` you can collect a few keystrokes into a macro and then run this macro over and over. Press left-shift `|>` to start the macro recorder, then execute the commands which should be part of the macro and exit with ON. Then use `|>` to run the macro. Holding down the `|>` key automatically repeats the macro until you release the key.

`Find` starts an incremental search. Press this key, then start typing the string you want to find. Type as many characters as necessary, then press ENTER to go to that cursor position. To cancel the search and go back to where the search started, press ON. Press the right arrow to find the next match.

When you press Left-shift `Find`, you start a (non-incremental) regular expression search. Read Emacs' manual for more information.

`Meta` starts a special mode in which many useful editing commands are directly accessible with single key presses. The transmit indicatior is on while this mode is active. To exit, press ENTER or ON.

Left-shift `Meta` suspends the editor and goes back to the stack. To return to the editor, press CONT (Left-shift ON).

Right-shift `Help` is the menu of Emacs configuration. A choose box appears with several actions. Selecting `Options` will show a dialog, which allow you to configure some aspects of Emacs: whether the minifont is used by default, whether the third page of the menu contains some templates for System RPL and Assembly Language development, the library to use by the `EDOB` command (described below), and some other things. There are also options to edit the `emacs` variable (which allows one to add macro commands to Emacs. Again, we refer you to Emacs' documentation), to edit the `diagram` variable (used by the SDiag library, but this variable is not discussed in this document), and to make some key assignments.

Pressing Left-shift `Help` toggles between the minifont and the current font.

`Indnt` indents the current line according to context. However, it is better to write the code already indented than to correct it later... Still, sometimes (such as when cutting and pasting), this can save some time. When left-shifted, removes `*␣` from the beginning of the current line (or all the selected lines), and when pressed right-shifted inserts the `*␣`.

`{↔}`, when pressed in a delimiter, jumps to the matching one. Works with `::` and `;`, `{` and `}` and a few others pairs.

`( → )` shows the stack diagram for the entry point under the cursor. For this to work, the SDiag library, distributed with Emacs, must the installed. All stack diagrams listed in this book are also available on the calculator through this library, and this can be of great help.

`EDOB` can be used to look into the ROM and into the contents of variables without exiting the editor. If you have ever used JAZZ's `ED` editor, this works similarly to the Right-shift Y key. It disassembles the entry under the cursor, and its source is viewed in another editor. Exit this sub-editor with ON or ENTER to go back to the original editing section. Of course, you can call `DOB` again in the sub-editor. This command requires the CQIF? or Nosy libraries (see section A.5). When Nosy is used, you can press this key longer to run the Nosy browsing environment. (The default is to start a new sub-editor.)

You should assign EDOB to a key since you will use it frequently. Because of the similarity with the Nosy and CQIF? commands, is suitable to assign both commands to the same key. When used in edit mode, EDOB is called. When not, Nosy or CQIF? is called. It is very easy to create an assignment like this with KEYMAN (see section A.2). First, put the list { EDOB Nosy } in the stack, and use OBJ→ or COMP→ to explode it. Drop the number of objects and run IfE?P. Use →TO? to add TakeOver to the object (since it needs to work in edit mode), and assign it to a key. If you have used the HP48 and JAZZ, Right-shift-hold +/- or Right-shift-hold 1/x will remind ED, and will not interfere with the normal operation. You can replace Nosy with CQIF? here if you prefer the latter library.

Actually, while the above example is very educational, it is not really necessary. EDOB automatically calls CQIF? or Nosy (depending on Emacs' settings) when it is called outside edit mode.

The last page of the Emacs menu contains some templates for System RPL and Assembly Language programming. Try them, you will easily discover what they do.

If you need help with Emacs menu commands, just press Help. It has help on the commands and on the menu keys. If you select help on the meny keys, it will display a screen describing the two pages of the Emacs menu. Each page is represented by three rows of labels, which mean, from top to bottom, the unshifted action, the left-shifted action and the right-shifted action. Some commands are inverted, these have different actions when pressed longer.

There is much more that Emacs can do. Please read the documentation to discover about the rest of the features.

A.7 Debugging

The debugging facilities for System RPL of the HP49G are the same as for User RPL: the built-in debugger (Left-shift CAT, NXT twice and RUN). Unfortunately, it does not work very well with some commands, which will be described later.

To start debugging, put the program or the name of the variable in which the program is stored in level one and press DBUG. Then, use the other commands to examine the program. The `SST` command executes the next step in the program and displays what has just been executed. You will need the System RPL stack (see section A.3) active for this to be useful. If the command being run is a sub-routine, `SST` executes this as a single step. `SST↓` is similar, but if the command is a sub-routine, it steps into this sub-routine and executes its first command.

To see the next two actions of the program, but not execute them, press `NEXT`. To stop the program being debugged, press `KILL`. To make it resume its normal operation, press CONT (Left-shift ON).

To insert a breakpoint into your program, insert the command `HALT` (`xHALT` for System RPL programmers) in the program at the point you want the program to stop. Then use the commands above to debug the program.

The debugger does not work with commands that take arguments from the runstream, such as `'` or `IT`. Do not try stepping over one of these commands, the only thing you will get is a nice crash :-). Currently, the only way to debug these commands is by inserting `xHALT` after these commands, and using CONT to skip past the next `xHALT`.

For simple to moderately complex programs, the procedures described above will be sufficient to find and correct bugs. If you run into a more serious problem with a complicated program, a bigger hammer may be needed: `SDB` in Jazz49.

A.8 JAZZ for the HP49

The JAZZ library, written originally by Mika Hesikanen and others for the HP48, implemented many of the features so far discussed in this chapter in a single, compact and very consistent library. On the HP48, this was without any doubt the best programming environment. JAZZ has been ported to the HP49 by Daniel Lidström (e-mail: danli97@ite.mh.se). However, at the time of this writing it is not a full replacement for MASD and the other tools. In particular it has no support for flashpointers and therefore cannot assemble or disassemble programs containing flashpointers. Another drawback is that Jazz needs to be installed in port 0, occupying 70kB (50kB for the light version) of RAM space. It also needs its own table of entry points, 40kb more, but that can fortunately be installed in any port.

The area where Jazz49 brings unique functionality to the HP49 is debugging, both of machine language programs (`DB`) and of System RPL programs

(SDB). In contrast to the HP49 built-in debugger, SDB can handle runstream commands correctly, so there is no need to insert many xHALT commands into the program. The lack of flashpointer support means that you cannot single-step the contents of flashpointers. Also the display of current and next commands in the status line is affected by this: when the program is near a flashpointer, “Invalid Object” will be displayed instead of the current and next commands.

To use the debugger, put the program to be debugged or just its name in level 1, and run the SDB command. You will then be presented a menu with your possible actions. →SST executes the next step. →IN is similar, but it will step inside of sub-routines. Use SNXT to show the next steps to be executed. You can insert breakpoints into programs with the SHALT command (write xSHALT in System RPL programs). Note that SHALT only works if SDB is already running, so you need to start your program with SDB and then press CONT to jump to the break point.

Very useful is also the possibility to browse loop and LAM environments with the LOOPS and LAMS commands, respectively. For more information on these and other commands, please refer to the JAZZ documentation.

Now you may wonder if you should really sacrifice 50kB of RAM for the occasional need to do serious debugging. Here is a solution: keep a BZ compressed version of the JAZZ library (light version) stored in port 2 under the name “Jazz”. When you need to debug, you can quickly install JAZZ with a small progamm:

```
« :2: Jazz RCL ˜ 0. STO 992. ATTACH »
```

where ˜ is the decompressor program in OT49. With a similar program, you remove it from port 0 when you are done.

Appendix B
Creating Libraries

Libraries are collections of commands that the user can access as if they were built-in in the system. If you have written a complex program with several sub-routines, it is much more convenient to distribute it as a library instead of as a directory. As a library, the user will not need to navigate through the variables to access your program; he can just type the command name from anywhere. Library commands appear in the catalog, and they can have on-line help. There is a menu showing all installed libraries, and a library can add itself or selected commands to some of the menus, such as the APPS menu.

Moreover, you can make only some of the commands in the library accessible to the user. This way, you can prevent the user from running commands that they should not, and you only need to provide error-checking for the user-accessible commands.

That should have been enough to convince you to distribute your programs as libraries. But you might be wondering, "But how do I *create* a library?"

Easy: the `CRLIB` command in library 256 (see section A.3) will do that for you. You just need to create a few special variables in a directory, which specify some aspects of the library, and then run that command. You will then get a library from the contents of the directory, which can be distributed.

Instead of `CRLIB`, you can use the `D↔L` command from the OT49 library (see section A.3.1). This command eases the entry of some of the variables below, and provides an easy way to add help to library commands. However, it does not add anything really new to the library creation process.

B.1 The Special Variables

In the directory that will be converted to a library, some variables, all having names starting with $, have special meanings that configure the created library. The table below lists the variables and their meanings.

Variable	Meaning
`$ROMID`	This specifies the number of the library. Each library should have a unique number, since there cannot be two libraries with the same number. It should be a real or an integer, in the range 769 to 1791.
`$TITLE`	This is the title of the library. The first five characters will be shown in the library menu. You can have a library without a title, but you will not be able to access the library from the library menu.
`$VISIBLE`	This is a list of variable names. The variables listed here will be made into user-accessible commands in the resulting library.
`$HIDDEN`	This is a list of variable names. The variables listed here will be converted into hidden commands in the resulting library.
`$CONFIG`	This is the library configuration object. This object will be evaluated at each warmstart. Normally, these configuration programs attach the library. This can be done by storing something like "`:: romid TOSRRP ;`" here, where `romid` is the library id. If you want, you can simply store the real number `1.` in `$CONFIG`, and a default configuration object will be produce, which attachs the library at each warmstart.
`$MESSAGE`	This is a list of strings which will be available in the library for use as (error) messages or general strings. If each message is only used once, it is not really worthwhile to create a message table. But if messages are used in many places, or if you want to make it easy to change messages to a different language, a message table is very useful. The list can contain up to 256 strings. Each message on the calculator is identified by a unique bint `#lllmm` consisting of a 3-digit library number (like `6FE`) and a two-digit message number `01`...`FF`. To access a message from a program use "`#lllmm JstGetTHEMESG`". To throw an error using a message number, use "`#lllmm DO#EXIT`". See Chapter 22 for more information.
`$EXTPRG`	This is the name of a command that allows customization of some menus, addition of help to commands and more. See below for more information on this.

Note that unlike other library creators, only the variables that are listed in `$VISIBLE` or `$HIDDEN` are made into command in the library. Variables that do not appear in either list are not converted. `$MESSAGE` is optional, you do not have to specify it.

B.2 The Library Message Handler

Libraries on the HP49G can contain a message handler. This program is called by the operating system at various occasions, in order to give the library a chance to modify menus, provide online help for its commands and other actions.

When creating a library from a directory, the reserved variable `$EXTPRG` can contain the name of a variable in the directory which will later become a rompointer in the library. This rompointer must be a program which accepts a bint on level one (a code representing one of the messages) and, depending upon the specific message, other arguments on higher stack levels.

B.2.1 Menu Extensions

The majority of messages can be used to extend some built-in menus. Among these are the APPS choose menu, several other choose menus, the SEARCH, GOTO and Tools submenus in the editor menu etc. When the message handler is called to extend a menu, the current menu is on the stack either as a list or as a meta. The program can then modify this menu and return it. So, the stack diagram for menu extensions is one of:

(`{ key1 ... keyN } #msg` → `modified_list #msg`)
(`key1 ... keyN #n #msg` → `modified_meta #msg`)

The message number bint stays on the stack, so that the message handler of another library can be called immediately to do its work in the same way.

The following menus on the HP49G can be extended using library messages.

#msg	Menu	Menu Type
0	APPS	list
1	Main STAT menu	list
2	Hypothesis submenu in STAT	list
3	Confidence Interval submenu in STAT menu	list
4	Finance menu	list
5	Numeric Solver menu	list
6	Time menu	list
8	Games (inside APPS)	meta
11	Editor SEARCH menu (when flag -117 is clr)	list
12	Editor TOOLS menu (when flag -117 is clr)	list

#msg	Menu	Menu Type
13	Editor GOTO (when flag -117 is clr)	list
14	Editor SEARCH menu (when flag -117 is set)	meta
15	Editor TOOLS menu (when flag -117 is set)	meta
16	Editor GOTO (when flag -117 is set)	meta

As an example, we show a message handler of a library whose ROMID is 1234. This handler will will add the library menu to the APPS menu, and a particular rompointer to the Games menu. When adding to the APPS menu, the example also makes sure that the new item is numbered just like the other items in the APPS menu. This should be done by all libraries.

```
::
  ZERO OVER#=case             (APPS menu)
  ::
    SWAPINCOMP                (save #msg, explode list)
    #1+DUP #>$                (make index for new entry)
    ".My Library" !append$    (add name to index number)
    ' :: % 1234. InitMenu% ;  (action: set my menu)
    TWO{}N                    (label & action -> list)
    SWAP P{}N                 (add new entry)
    SWAP                      (get the ZERO back)
  ;
  EIGHT OVER#=case            (Games submenu)
  ::
    DROP                      (drop the message)
    { "PlayMe" ROMPTR 4D2 0 } (new entry for menu)
    SWAP#1+                   (add to meta)
    EIGHT                     (put msg number back)
  ;
;
```

B.2.2 Online Help for Libary Commands

On the HP49G, all the CAS commands have a short help text which can be displayed from the catalog, or with the `SDIAG` command in the Emacs library (see section A.6). When the catalog choose box highlights a CAS command, the menu under the choose box has an additional button, the HELP button. Pressing this button shows the corresponding help text. External libraries can provide help for their commands in a similar way, using the message handler and messages number nine and ten. Message nine is a query if the library provides help for a given rompointer. The stack diagram is

(`romptr FALSE NINE` → `romptr TRUE/FALSE NINE`)
where the `TRUE/FALSE` in stack level two indicates if the library is prepared to provide help for the rompointer in level three. This message is used to determine if the HELP button in the CATalog should be turned on.

Message ten is then used to actually display the help when the user presses the HELP button. The stack diagram here is
(`romptr TEN` → `FALSE`)

Before pushing `FALSE`, the message handler should display the help text.

The following example is a message handler which provides a short help string for every visible command in the library.

```
::
  NINE #=casedrop
  ::
    DROPTRUE NINE             (all cmds have help)
  ;
  TEN #=casedrop
  ::
    DUP DECOMP$ NEWLINE&$ SWAP (save cmd name as string)
    ROMPTR># SWAPDROP          (index of romptr in lib)
    {                          (list of help strings)
      "Help text for romptr 0"
      "Help text for romptr 1"
      ...
      "Help text for romptr N" (last visible rompointer)
    }
    SWAP#1+ NTHCOMPDROP        (extract correct help str)
    &$                         (Add the command name)
    FALSE SWAP ViewStrObject   (display the text)
  ;
;
```

Note that `ViewStrObject` conveniently pushes `FALSE` on the stack, which is the required return value of message nine. The message handler gets a bit more complicated if help is only provided for a few rompointers. In this case, the handler of message nine must check the rompointer against a list, and message ten must use `Lookup` or something similar to extract the help text. Instead of simply displaying a string, message ten can also do more complicated things, like launching a whole application to provide help.

The library creator in the OT49 library (see section A.3.1) provides a simple way to add help support to a library.

B.2.3 The Library Menu Message

If the menu of a library is invoked via the LIBS menu (rightshift 2), the romid of this library is sent to the message handler of the library. The library may use this for easter egg-like stuff (displaying an icon (see for example the Libman library), doing something funny with the menu (e.g. LTool) or playing a melody). It can also change the menu settings, for example to provide functionality for the shifted menu buttons (e.g. ConstTools).

The stack diagram for this message is

(`#romid` → `#romid`)

The following example is the message handler of a library `# 60F` and it temporarily displays a copyright notice when the library menu is selected.

```
::
  # 60F OVER#=case
  ::
    ZEROZERO
    "(c) 2001 Some Author"
    $>grob XYGROBDISP
    SetDA1Temp
  ;
;
```

Appendix C
User RPL Commands

The listing here is of all the user-accessible commands and functions, with their addresses. In most cases, the User RPL name of a command is equal to the System RPL name with leading ˜ and x stripped. The few exceptions are marked in the table.

C.1 Reference

Addr.	Name	Description
030314	˜xABCUV	(pa pb c → u v)
39A07	xABS	(x → x')
390E4	xACK	(→)
390C9	xACKALL	(→)
025314	˜xACOS2S	(symb → symb')
3A8D8	xACOSH	(x → x')
3A7DC	xACOS	(x → x')
06E314	˜xADDTMOD	(symb1 symb2 → symb3)
0000DE	xADDTOREAL	(var →)
3AAE5	xALOG	(x → x')
04B0AB	xAMORT	(n → princ intr bal)
3CA07	xAND	(x1 x2 → x3)
3F033	xANS	(n → ob)
3D7AC	xAPPLY	({symb1 .. symbn} f → f(symb1...symbn))
3EAC7	xARCHIVE	(:port:name →) (:IO:name →)
3C8C6	xARC	(c r θ1 θ2 →) ({#x #y} #r θ1 θ2 →)
3A390	xARG	(c → θ)
085314	˜xARIT	(→)

Addr.	Name	Description
3BEC5	xARRY>	([] → x1...xn {n})
		([[]] → x11...xnm {n m})
		UserRPL: ARRY→
3BE9B	x>ARRY	(x1..xn n → [])
		(x11...xnm {n m} → [[]])
		UserRPL: →ARRY
024314	~xASIN2C	(symb → symb')
023314	~xASIN2T	(symb → symb')
3A88E	xASINH	(x → x')
3A756	xASIN	(x → x')
3EEE7	xASN	(obj key →)
		('SKEY' →)
38DE1	xASR	(# → #')
022314	~xATAN2S	(symb → symb')
3A94F	xATANH	(x → x')
3A844	xATAN	(x → x')
3EB64	xATTACH	(n →)
3C49F	xAUTO	(→)
3C3B2	xAXES	(c →)
		({c tick $x $y } →)
04A314	~xAXL	({} → [])
		([] → ())
049314	~xAXM	([A] → [M])
04C314	~xAXQ	([nxn] [n] → [nxn]' [n])
3C9D3	xBAR	(→)
3E196	xBARPLOT	(→)
080314	~xBASE	(→)
		aka: xALGB
3EDCC	xBAUD	(n →)
39765	xBEEP	(freq dur →)
3E2C1	xBESTFIT	(→)
3B655	xBIN	(→)
3E171	xBINS	(min width n → [[]] [])
3C70A	xBLANK	(#width #height → grob)
3C6E0	xBOX	({#n1 #m1} {#n2 #m2} →)
		(c1 c2 →)
38F21	xB>R	(# → R)
		UserRPL: B→R
3EE47	xBUFLEN	(→ nchars 0/1)

Addr.	Name	Description
39480	xBYTES	(obj → chksum size)
01E0DE	xC2P	({} → ?????)
07E314	˜xCASCFG	(→)
0330DE	xCASCMD	(→ ?)
38B28	xCASE	(→)
3AD1B	xCEIL	(x → n)
3C3DC	xCENTR	((x,y) →)
		(x →)
3B4E9	xCF	(n →)
03A314	˜xCHINREM	([]1 []2 → []3)
04D0AB	xCHOOSE	(title {elems} pos → ob 1)
		(title {elems} pos → 0)
3BC19	xCHR	(n → $)
3B362	x%CH	(x1 x2 → x3)
01D0DE	xCIRC	(prg {} → ?????)
3EDAC	xCKSM	(n_type →)
3DD4E	xCLEAR	(ob1 .. obn →)
3DD8E	xCLSIGMA	(→)
		UserRPL: CLΣ
39144	xCLKADJ	(ticks →)
39839	xCLLCD	(→)
3EC95	xCLOSEIO	(→)
3E91A	xCLUSR	(→)
		UserRPL: CLVAR
081314	˜xCMPLX	(→)
3B193	xCNRM	([] → col_norm)
3E5A0	xCOLCT	(symb → symb')
0300DE	xCOLLECT	(symb → symb')
3E0FD	xSIGMACOL	(x_col y_col →)
		UserRPL: COLΣ
0380AB	x→COL	([[]] → [v1]...[vn] n)
		([] → x1...xn n)
03E0AB	xCOL-	([] n → []' xn)
		([[]] n → [[]]' [vn])
0390AB	xCOL→	([v1]...[vn] n → [[]])
		(x1...xn n → [])
03F0AB	xCOL+	([[]] [[]]' n → [[]]'')
		([] x n → []')

Addr.	Name	Description
3B423	xCOMB	(n k → Cn,k) Symbolic argument allowed.
0260AB	xCOND	([[n*n]] → x)
3C967	xCONIC	(→)
39A6C	xCONJ	(x → x')
0180AB	xCONLIB	(→)
3BF77	xCON	({ n } x → []) ({ n k } x → [[]]) ([] x → []')
0190AB	xCONST	(name → x)
3989C	xCONT	(→)
38F41	xCONVERT	(x1_u1 x2_u2 → x3_u2)
3DE24	xCORR	(→ x_correlation)
3A6C2	xCOSH	(x → x')
3A5D0	xCOS	(x → x')
3DE3F	xCOV	(→ x_covariance)
3C58E	xC>PX	((x,y) → {#n #m}) UserRPL: C→PX
393CA	xCRDIR	(name →)
3D128	xCR	(→)
3B208	xCROSS	([1] [2] → [3])
3BAF5	xC>R	((x,y) → x y) UserRPL: C→R
057314	˜xCURL	([func] [vars] → [])
0120AB	xCYLIN	(→)
0610AB	xDARCY	(xe/D yRe → xDarcy)
39104	xSETDATE	(date →)
39078	xDATE	(→ date)
39238	xDATE+	(date ndays → date')
0690AB	xdB	(→ %1)
0150DD	xDBUG	(prog →) (name →)
39218	xDDAYS	(date1 date2 → days)
3B670	xDEC	(→)
3E576	xDECR	(name → x_new)
3E85C	xDEFINE	('name=expr' →) ('name(name1...)=expr(name1...) →)
3B549	xDEG	(→)

Addr.	Name	Description
391D8	xDELALARM	(n →)
3D1C7	xDELAY	(x_delay →)
3EF3B	xDELKEYS	(rc.p →) (0 →) ('S' →)
3C51F	xDEPND	(name →) ({name y1 y2} →) ({y1 y2} →) (y1 y2 →)
3DCA7	xDEPTH	(→ n)
00E314	~xDERIV	(symb var → symb')
003314	~xDERVX	(symb → symb')
00F314	~xDESOLVE	(eq func → func')
3B1BA	xDET	([[]] → x)
3EB84	xDETACH	(n →) (:port:n →)
3D202	x∂	(symb var → symb')
03A0AB	x→DIAG	([[]] → vec)
03B0AB	xDIAG→	([] { dims } → [[]])
084314	~xDIFF	(→)
00E0AB	xDIFFEQ	(→)
39725	xDISP	(obj n_line →)
0160DD	xDISPXY	(ob {#x #y} %size →) Display ob (decompiled if nexessary) at the given display coordinates, using either the system font (%size=2) or the minifont (%size=1).
056314	~xDIV	([func] [vars] → func)
026314	~xDIV2	(symb1 symb2 → squot srem)
072314	~xDIV2MOD	(symb1 symb2 → squot srem)
044314	~xDIVIS	(symb → {})
071314	~xDIVMOD	(symb1 symb2 → sq)
062314	~xDIVPC	(symb1 symb2 n → symb3)
3816B	xDO	(→)
39527	xDOERR	(n →) ($ →) (0 →)
05B0AB	xDOLIST	({1}...{n} n prog → {}) ({1}...{n} prog → {} (n=1))

Addr.	Name	Description
0540AB	xDOSUBS	({} n prog → {}')
		({} prog → {}' (n=1))
3B1E1	xDOT	([1] [2] → x)
3C484	xDRAW	(→)
06B0AB	xDRAW3DMATRIX	([[]] v_min v_max →)
3C4BA	xDRAX	(→)
3DC56	xDROP2	(ob1 ob2 →)
3DCC7	xDROPN	(ob1...obn n →)
3DC3B	xDROP	(ob →)
3B06E	xD>R	(x → (π/180)x)
		UserRPL: D→R
3EFEF	xDTAG	(tag:obj → obj)
3DC05	xDUP2	(1 2 → 1 2 1 2)
3F29A	xDUPDUP	(1 → 1 1)
3DCE2	xDUPN	(1...n n → 1...n 1...n)
3DBEA	xDUP	(ob → ob ob)
0090DD	xEDITB	(ob → ob')
0070DD	xEDIT	(ob → ob')
39B1E	xCONSTANTe	(→ e)
		UserRPL: e
02E314	~xEGCD	(symb1 symb2 → symb3 symb4 symb5)
02C0AB	xEGV	([[]] → [[evect]]' [evals])
02D0AB	xEGVL	([[]] → [egval])
3805D	xELSE	(→)
38A54	xENDDO	(1/0 →)
		UserRPL: END
0570AB	xENDSUB	(→ x)
		Number of lists in DOSUBS.
3B5DA	xENG	(n →)
088314	~xEPSX0	(symb1 → symb2)
3BDE6	xEQ>	('l=r' → l r)
		UserRPL: EQ→
00B0DD	xEQW	(symb → symb')
3C553	xERASE	(→)
3955B	xERR0	(→)
39591	xERRM	(→ $msg)
39576	xERRN	(→ $nerr)
038314	~xEULER	(z1 → z2)
395AC	xEVAL	(ob → ?)

Addr.	Name	Description
06C314	˜xEXLR	(symb → symb1 symb2)
076314	˜xEXPANDMOD	(symb1 → symb2)
000314	˜xEXPAND	(symb1 → symb2)
		([symb1] → [symb2])
3E5E9	xEXPAN	(symb1 → symb2)
3E25E	xEXPFIT	(→)
017314	˜xEXPLN	(symb1 → symb2)
3AB6F	xEXPM	(x → x')
3A9B7	xEXP	(x → x')
0050AB	xEYEPT	(xx xy xz →)
0620AB	xF0λ	(y_lambda xT → x_power)
001314	˜xFACTOR	(symb → symb1*symb2...)
		(z → z1*z2...)
077314	˜xFACTORMOD	(symb → symb1*symb2...)
043314	˜xFACTORS	(z → {z1 m1...})
		(symb → {symb1 m1...})
0600AB	xFANNING	(x_x/D y_Re → x_fanning)
3F2DF	xFAST3D	(→)
3B635	xFC?C	(n → 0/1)
3B529	xFC?	(n → 0/1)
041314	˜xFCOEF	([] → symb)
01A0AB	xFFT	([] → []')
00C0DD	xFILER	(→)
391AE	xFINDALARM	(date → n)
		({date time} → n)
		(0 → n)
3ED76	xFINISH	(→)
3B59A	xFIX	(n →)
0170AB	xFLASHEVAL	(# → ?)
3ACD1	xFLOOR	(x → n)
00F0DD	xFONT6	(→ font)
00E0DD	xFONT7	(→ font)
00D0DD	xFONT8	(→ font)
0030DD	xFONT→	(→ font)
0020DD	x→FONT	(font →)
38252	xSTARTVAR	(start finish →)
		UserRPL: FOR
05E314	˜xFOURIER	(symb z → c_z)
3AC87	xFP	(x → x')

Addr.	Name	Description
39745	xFREEZE	(n →)
042314	˜xFROOTS	(symb → [])
3B615	xFS?C	(n → 0/1)
3B509	xFS?	(n → 0/1)
3C955	xFUNCTION	(→)
3D56B	x|	(symb {var val ...} → x')
06B314	˜xFXND	('x/y' → x y)
0070DE	xGAMMA	(x → x')
04D314	˜xGAUSS	(symb [vars] → [diag] [P] symb' [vars])
075314	˜xGCDMOD	(x1 x2 → x3)
02C314	˜xGCD	(x1 x2 → x3)
0550AB	x∆LIST	({} → {}')
3C22D	xGETI	(ob pos → ob' pos' elm) ob = [] or [[]] or {} or name pos = n or {n} or {n m}
3C1C7	xGET	(ob n → elm) ob = [] or [[]] or {} or name pos = n or {n} or {n m}
3C74A	xGOR	(g_targ {#n #m} grob → g_targ') (g_targ (x,y) grob → g_targ') (PICT ... ... →)
3B57F	xGRAD	(→)
3C5AE	xGRAPH	(→) UserRPL: PICTURE
00A0AB	xGRIDMAP	(→)
07C314	˜xGROBADD	(gr1 gr2 → gr3)
3C8A1	x>GROB	(ob n_chrsize → grob) UserRPL: →GROB
3D503	xSUM	(var n1 n2 symb → x) UserRPL: Σ
3DDEE	xSIGMA-	(→ x) (→ []) UserRPL: Σ-
3DDC4	xSIGMA+	(x →) (x1...xn →) UserRPL: Σ+
3E156	xSIGMALINE	(→ symb) UserRPL: ΣLINE

Addr.	Name	Description
0590AB	xΣLIST	({} → x)
3DE90	xSUMX2	(→ xsum) UserRPL: ΣX2
3DE5A	xSUMX	(→ xsum) UserRPL: ΣX
3DEC6	xSUMXY	(→ xsum) UserRPL: ΣXY
3DEAB	xSUMY2	(→ xsum) UserRPL: ΣY2
3DE75	xSUMY	(→ xsum) UserRPL: ΣY
3C7D8	xGXOR	(g_targ {#n #m} g_src → g_targ') (g_targ (x,y) g_src → g_targ') (PICT ... ... →)
046314	~xHADAMARD	([M1] [M2] → [M3])
020314	~xHALFTAN	(symb → symb')
3880D	xHALT	(→)
0040DD	x→HEADER	(n →)
0050DD	xHEADER→	(→ n)
05C314	~xHERMITE	(z → symb)
059314	~xHESS	(symb [vars] → [M] [grad] [vars])
3B68B	xHEX	(→)
054314	~xHILBERT	(z → [M])
3C9C1	xHISTOGRAM	(→)
3E1CA	xHISTPLOT	(→)
3B14C	xHMS-	(hms1 hms2 → hms3)
3B12C	xHMS+	(hms1 hms2 → hms3)
3B0EC	x>HMS	(x → x') UserRPL: →HMS
3B10C	xHMS>	(x → x') UserRPL: HMS→
39405	xHOME	(→)
037314	~xHORNER	(symb1 x → symb2 x symb3)
031314	~xIABCUV	(n1 n2 n3 → n4 n5)
0060DE	xIBERNOULLI	(n → x)
00B314	~xIBP	(uv' v → uv -u'v)
03B314	~xICHINREM	([]1 []2 → []3)
027314	~xIDIV2	(n1 n2 → quot rem)

Addr.	Name	Description
3C02E	xIDN	(n → [[]])
		([[]] → [[]]')
		(name → [[]])
02F314	˜xIEGCD	(n1 n2 → c b a)
37F48	xIF	(→)
387AC	xIFERR	(→)
01B0AB	xIFFT	([] → []')
396A4	xIFT	(0/1 obj → ?)
395F3	xIFTE	(0/1 objT objF → ?)
39B3B	xi	(→ i)
011314	˜xILAP	(symb → symb')
3B87E	xIM	((x,y) → y)
		([] → []')
3E54C	xINCR	(name → x')
3C33E	xINDEP	(name →)
		({name x1 x2} →)
		({x1 x2} →)
		(x1 x2 →)
08A314	˜x∞	(→ '+∞')
		Infinity
04C0AB	xINFORM	($ {flds} fmt {rst} {init} → {} 1)
		($ {flds} fmt {rst} {init} → 0)
3EEBD	xINPUT	($prompt $ → $')
		($prompt {specs} → $')
3D434	x∫	(x1 x2 symb var → symb')
3F007	xINT	(f(var) var x0 → F(x0))
004314	˜xINTVX	(f(x) → F(x))
074314	˜xINVMOD	(x → x')
3A32B	xINV	(x → 1/x)
		([[]] → [[]]')
3AC3D	xIP	(x → n)
029314	˜xIQUOT	(n1 n2 → n3)
02B314	˜xIREMAINDER	(n1 n2 → n3)
3F0B7	xI>R	(n → x)
		UserRPL: I→R
3E648	xISOL	(symb var → symb')
03C314	˜xISPRIME?	(n → 1)
		(n → 0)

Addr.	Name	Description
050314	˜xJORDAN	([nxn] → minpol chrpol {} [])
3EE2C	xKERRM	(→ msg)
07B314	˜xKEYEVAL	(rc.p → ?)
39854	xKEY	(→ rc 1)
		(→ 0)
06C0AB	x→KEYTIME	(ticks →)
06D0AB	xKEYTIME→	(→ ticks)
3ECE4	xKGET	(name →)
		("name" →)
		({names} →)
		({{old new}...} →)
394F1	xKILL	(→)
3C5C9	xLABEL	(→)
05D314	˜xLAGRANGE	([2xn] → pol)
0000DD	x→LANGUAGE	(n →)
0010DD	xLANGUAGE→	(→ n)
058314	˜xLAPL	(symb [vars] → symb')
010314	˜xLAP	(symb → symb')
397E5	xLAST	(→ ob1 .. obn)
		UserRPL: LASTARG
3C881	x>LCD	(grob →)
		UserRPL: →LCD
3C866	xLCD>	(→ grob)
		UserRPL: LCD→
02D314	˜xLCM	(symb1 symb2 → symb3)
055314	˜xLCXM	(n1 n2 prog → [])
012314	˜xLDEC	(symb1 symb2 → symb3)
05A314	˜xLEGENDRE	(n → pol)
032314	˜xLGCD	({symb...} → {} gcd)
0160AB	xLIBEVAL	(# → ?)
3EB42	xLIBS	(→ {title nlib nport ...})
005314	˜xLIMIT	(func point → lim)
3C68C	xLINE	((x1,y1) (x2,y2) →)
		({#n1 #m1} {#n2 #m2} →)
3E214	xLINFIT	(→)
0150AB	xLININ	(symb var → 0/1)
052314	˜xLINSOLVE	([eqs] [vars] → [eqs] {pp} sol)
014314	˜xLIN	(symb → symb')

Addr.	Name	Description
3BAC1	xLIST>	({} → ob1...obn n)
		UserRPL: LIST→
3B7D2	x>LIST	(ob1 .. obn n → {})
		UserRPL: →LIST
06D314	˜xLNAME	(symb → [vars])
016314	˜xLNCOLLECT	(symb → symb')
3AB2F	xLNP1	(x → x')
3AA01	xLN	(x → x')
3E239	xLOGFIT	(→)
3AA73	xLOG	(x → x')
0320AB	xLQ	([[]] → [[L]] [[Q]] [[P]])
3DF83	xLR	(→ Intercept Slope)
02B0AB	xLSQ	([B] [[A]] → []')
		([[B]] [[A]] → [[]]')
0300AB	xLU	([[]] → [[L]] [[U]] [[P]])
06A314	˜xLVAR	(symb → symb [vars])
051314	˜xMAD	([] → det inv coeff cpol)
3B02E	xMANT	(x → x')
066314	˜xMAP	({} prog → {}')
3DAD0	xMATCHUP	(symb {spat srepl} → symb' 0/1)
		(symb {spat srepl scond} → symb' 0/1)
		UserRPL: ↑MATCH
3DEE1	xMAXSIGMA	(→ xmax)
		(→ [x1...xn])
		UserRPL: MAXΣ
39AE4	xMAXR	(→ MAXR)
3ADA5	xMAX	(x y → x')
0760AB	xMCALC	(var →)
		({vars} →)
		("ALL" →)
3DEFC	xMEAN	(→ xmean)
		(→ [x1...xn])
3E8C1	xMEM	(→ x)
3E9D4	xMENU	(% →)
07A314	˜xMENUXY	(n1 n2 →)
3DF17	xMINSIGMA	(→ xmin)
		(→ [x1...xn])
		UserRPL: MINΣ

Addr.	Name	Description
0110DD	x→MINIFONT	(font →)
0120DD	xMINIFONT→	(→ font)
0730AB	xMINIT	(→)
39B01	xMINR	(→ MINR)
3AE2B	xMIN	(x y → x')
0740AB	xMITM	(title {vars} →)
079314	˜xMODSTO	(mod →)
3AFCB	xMOD	(x y → x')
0770AB	xMROOT	(var → x)
		("ALL" →)
04E0AB	xMSGBOX	($ →)
0720AB	xMSOLVR	(→)
070314	˜xMULTMOD	(symb1 symb2 → symb3)
0750AB	xMUSER	(var →)
		({vars} →)
		("ALL" →)
0060DD	x→NDISP	(n →)
01C0AB	xNDIST	(xq v x → x')
3F2B5	xNDUPN	(ob n → ob .. ob n)
39976	xNEG	(x → x')
394AA	xNEWOB	(ob → ob)
3831C	xNEXT	(→)
03D314	˜xNEXTPRIME	(n → n')
3DE09	xNSIGMA	(→ nrows)
		UserRPL: NΣ
3F264	xNIP	(ob1 ob2 → ob2)
3CB13	xNOT	(x → x')
3F0FC	xNOVAL	(→)
0560AB	xNSUB	(→ npos)
3BBF9	xNUM	($ → n)
39785	x>NUM	(x → x')
		UserRPL: →NUM
0060AB	xNUMX	(n →)
0070AB	xNUMY	(n →)
3885C	xRPN->	(ob1 .. obn →)
		UserRPL: →
3BE38	xOBJ>	(ob → ?)
		UserRPL: OBJ→
3B6A6	xOCT	(→)

Addr.	Name	Description
3950C	xOFF	(→)
3EC75	xOPENIO	(→)
3E8F0	xORDER	({names} →)
3CA8D	xOR	(x y → x')
3DC8C	xOVER	(1 2 → 1 2 1)
039314	˜xPA2B2	(n → n')
3C98B	xPARAMETRIC	(→)
3EDEC	xPARITY	(n →)
0090AB	xPARSURFACE	(→)
034314	˜xPARTFRAC	(symb → symb')
393EA	xPATH	(→ {HOME dir1 .. dirn})
04F314	˜xPCAR	([nxn] → pol)
0450AB	xPCOEF	([roots] → [coefs])
00D0AB	xPCONTOUR	(→)
01F0AB	xPCOV	(→ xpcovariance)
3C4F5	xPDIM	((xmin,ymin) (xmax,ymax) →)
		(#width #height →)
3B477	xPERM	(n k → n')
0460AB	xPEVAL	([coefs] x → x')
3EAA7	xPGDIR	(name →)
3F27F	xPICK3	(1 2 3 → 1 2 3 1)
3DCFD	xPICK	(1...n n → 1..n 1)
3C72A	xPICT	(→ PICT)
05A0AB	xΠLIST	({} → x)
06A0AB	xPINIT	(→)
39AC7	xPI	(→ π)
		UserRPL: π
3C638	xPIXOFF	((x,y) →)
		({#n #m} →)
3C60E	xPIXON	((x,y) →)
		({#n #m} →)
3C662	xPIX?	((x,y) → 1/0)
		({#n #m} → 1/0)
3EE9D	xPKT	(data type → response)
00A314	˜xPLOTADD	(f →)
3C392	xPMAX	((x,y) →)
3C372	xPMIN	((x,y) →)
3C979	xPOLAR	(→)

Addr.	Name	Description
3BB94	xPOS	(str substring → n/0)
		({} ob → n/0)
073314	~xPOWMOD	(symb exp → symb')
3D0D7	xPR1	(ob → ob)
3DFDD	xPREDV	(x → y)
3E01D	xPREDX	(y → x)
3DFFD	xPREDY	(x → y)
00C314	~xPREVAL	(f x1 x2 → symb)
		(f x1 x2 → x)
03E314	~xPREVPRIME	(n → n')
3D1E7	xPRLCD	(→)
38BBF	xPROMPT	($ →)
08B314	~xPROMPTSTO	(var →)
0440AB	xPROOT	([coefs] → [roots])
035314	~xPROPFRAC	(x → symb')
3D10D	xPRST	(→)
3D143	xPRVAR	(name →)
		({names} →)
		(:port:name →)
01D0AB	xPSDEV	(→ xpsdev)
		(→ {x1...xn})
0040DE	xPSI	(symb → symb')
0030DE	xPsi	(symb n → symb')
036314	~xPTAYL	(pol x → pol')
3E87C	xPURGE	(name →)
		({names} →)
		(:port:name →)
		(:port:nlib →)
3C139	xPUTI	(ob pos obj → [] pos')
		ob = [] or [[]] or {} or name
		pos = n or {n} or {n m}
3C0BF	xPUT	(ob pos obj → ob')
		ob = [] or [[]] or {} or name
		pos = n or {n} or {n m}
3EA49	xPVARS	(nport → {} mem)
3C5E4	xPVIEW	((x,y) →)
		({#n #m} →)
3C56E	xPX>C	({#m #n} → (x,y))
		UserRPL: PX→C

Addr.	Name	Description
3DA3E	x->Q	(x → a/b) UserRPL: →Q
3DA63	x->QPI	(x → symb) UserRPL: →Qπ
0310AB	xQR	([[]] → [[Q]] [[R]] [[P]])
3E66F	xQUAD	(symb var → symb')
3D6F6	xQUOTE	(ob → 'ob)
028314	˜xQUOT	(p1 p2 → p3)
04B314	˜xQXA	(symb [vars] → [[]] [vars])
3B564	xRAD	(→)
3B3E6	xRAND	(→ x)
02A0AB	xRANK	([[]] → n)
0350AB	xRANM	({m n} → [[]])
3DBCA	xPREDIV	(x y → x/y) UserRPL: RATIO
38F01	xR>B	(x → #) UserRPL: R→B
3D393	xRCEQ	(→ EQ)
3B7ED	xR>C	(x y → (x,y)) UserRPL: R→C
3918E	xRCLALARM	(n → {date time action rep})
3B715	xRCLF	(→ {#s1 #u1 #s2 #u2})
03F0DE	xRCLVX	(→ name) Recall the current content of the reserved CAS variable VX.
3DDA9	xRCLSIGMA	(→ [[]]) UserRPL: RCLΣ
3EF79	xRCLKEYS	(→ {ob ... key ...})
3EA2E	xRCLMENU	(→ x)
3E6F1	xRCL	(var → x) (:port:nlib → lib) (:port:name → ob) (:port:{path} → ob)
3B6FA	xRCWS	(→ n)
3B0AE	xR>D	(x → (180/π)x) UserRPL: R→D

Addr.	Name	Description
3BEEC	xRDM	(ob size → ob')
		(name size →)
		ob= [] or [[]]
		size = {n} or {n m}
3B401	xRDZ	(x →)
3ED22	xRECN	(name →)
		($name →)
0110AB	xRECT	(→)
3ED56	xRECV	(→)
048314	˜xREF	([[]] → [[]]')
3B819	xRE	((x,y) → x)
		([] → []')
02A314	˜xREMAINDER	(p1 p2 → p3)
0130DD	xRENAME	(name name' →)
069314	˜xREORDER	(pol var → pol')
38105	xREPEAT	(1/0 →)
3B9D2	xREPL	(ob pos new → ob')
		ob= [[]] or [] or {} or $ or PICT
		pos= N or {n m} or (n,m)
3C41A	xRES	(n_int →)
3EAE7	xRESTORE	(:port:name →)
0050DE	xRESULTANT	(p1 p2 → res)
05D0AB	xREVLIST	({1...n} → {n...1}')
3F070	xR>I	(x → n)
		UserRPL: R→I
00D314	˜xRISCH	(f var → F)
0220AB	xRKFERR	({} h → {} h dy err)
0210AB	xRKFSTEP	({} tol h → {} tol h')
0200AB	xRKF	({} xtol xTf → {} xtol)
		({} {xtol step} xTf → {} xtol)
38E01	xRL	(# → #')
38E21	xRLB	(# → #')
3AEB1	xRND	(x n → x')
3B16C	xRNRM	([] → x)
3DD33	xROLLD	(n ... 1 n → 1 n...2)
3DD18	xROLL	(1...n n → 2...n 1)
06F0AB	xROMUPLOAD	(→)
3D3CE	xROOT	(prog/s var guess → x)
		(prog/s var {guesses} → x)

Addr.	Name	Description
3DC71	xROT	(1 2 3 → 2 3 1)
03C0AB	xROW-	([[]] nrow → [[]]' [])
		([] n → []' elt)
03D0AB	xROW+	([[]] [[]]' n → [[]]'')
		([[]] [] n → [[]]')
		([] n n' → [])
0360AB	x→ROW	([[]] → [1]...[n] n)
		([] → x1...xn n)
0370AB	xROW→	([1]...[n] n → [])
		(x1...xn → [])
38E41	xRR	(# → x')
38E61	xRRB	(# → x')
0340AB	xRREF	([[]] → [[]]')
047314	˜xrref	([[]] → [pp] [[]]')
078314	˜xRREFMOD	([[]] → [[]]')
0240AB	xRRKSTEP	({} xtol h last → {} xtol h' cur)
0230AB	xRRK	({} xtol xTfinal → {} xtol)
0250AB	xRSBERR	({} h → {} h dy err)
3B22F	xRSD	([B] [[A]] [Z] → []')
		([[B]] [[A]] [[Z]] → [[]]')
0400AB	xRSWP	([]/[[]] i j → []/[[]])
3C9E5	xSAME	(ob1 ob2 → 1/0)
3EE82	xSBRK	(→)
3C444	x*H	(xf →)
		UserRPL: SCALEH
3C464	x*W	(yf →)
		UserRPL: SCALEW
3C4D5	xSCALE	(xs ys →)
3E1EF	xSCATRPLOT	(→)
0330AB	xSCHUR	([[]] → [[Q]] [[T]])
3B5BA	xSCI	(n →)
3E127	xSCLSIGMA	(→)
		UserRPL: SCLΣ
3E385	xSCONJ	(name →)
07D314	˜xSCROLL	(ob →)
3DF32	xSDEV	(→ xsdev)
		(→ [x1...xn])

Addr.	Name	Description
3ECB0	xSEND	(name →)
		({names} →)
		({{old new}...} →)
0530AB	xSEQ	(prog var start end incr → {})
007314	~xSERIES	(func var order → {} symb')
3ED91	xSERVER	(→)
064314	~xSEVAL	(symb → symb')
3B4C9	xSF	(n →)
3E696	xSHOW	(symb name → symb')
		(symb {names} → symb')
0630AB	xSIDENS	(x → x')
0020DE	xSIGMA	(f var → F)
0010DE	xSIGMAVX	(f(x) → F(x))
05F314	~xSIGNTAB	(symb → {})
3A3EE	xSIGN	(x → x')
033314	~xSIMP2	(x y → x/gcd y/gcd)
0220DE	xSIMPLIFY	(symb → symb')
018314	~xSINCOS	(symb → symb')
3A678	xSINH	(x → x')
3E331	xSINV	(name →)
3A57C	xSIN	(x → x')
3BB1F	xSIZE	(ob → n)
		(ob → {N m})
38E81	xSL	(# → #')
38EA1	xSLB	(# → #')
00C0AB	xSLOPEFIELD	(→)
3E35B	xSNEG	(name →)
0290AB	xSNRM	([] → x)
03F314	~xSOLVE	(symb var → {zeros})
086314	~xSOLVER	(→)
008314	~xSOLVEVX	(symb → {zeros})
05E0AB	xSORT	({} → {}')
0130AB	xSPHERE	(→)
3A4EF	xSQ	(x → x')
38EC1	xSR	(# → #')
0280AB	xSRAD	([[]] → x)
38EE1	xSRB	(# → #')
3EC55	xSRECV	(n → $ 0/1)
0100DD	xSREPL	(str find repl → str')

Addr.	Name	Description
381AB	xSTART	(start finish →)
3B5FA	xSTD	(→)
3851F	xSTEP	(n →) (symb →)
3D3AE	xSTEQ	(ob →)
3EE62	xSTIME	(x →)
39164	xSTOALARM	(time → n) ({date time act rep} → n)
3B749	xSTOF	({#s1 #u1 #s2 #u2} →)
3DD6E	xSTOSIGMA	(ob →) UserRPL: STOΣ
0400DE	xSTOVX	(name →) Store object into the reserved CAS variable VX.
3EF07	xSTOKEYS	({ob key ...} →) ({'S' ob key ...} →) ('S' →)
3E739	xSTO	(ob name →) (ob :port:name →) (lib port →) (ob 'name(i)' →)
3E823	xSTO>	(ob id →) (ob symb →) Like xSTO, but if the level 1 argument is symbolic, use the first element of it as the variable to write to.
3E406	xSTO-	(ob name →)
3E46C	xSTO/	(ob name →)
3E4D2	xSTO*	(ob name →)
3E3AF	xSTO+	(ob name →)
3BBD9	xSTR>	($ → ob) UserRPL: STR→
0580AB	xSTREAM	({} prog → x)
3BBBE	x>STR	(ob → $) UserRPL: →STR
3B6C1	xSTWS	(n →)
3B8D7	xSUB	(ob start end → ob') ob= [[]], $, {}, grob start,end = n, {n m}, (n,m)

Addr.	Name	Description
002314	~xSUBST	(symb var=s1 → symb')
06F314	~xSUBTMOD	(x1 x2 → x3)
02E0AB	xSVD	([[]] → [[U]] [[V]] [S])
02F0AB	xSVL	([[]] → [])
3DC20	xSWAP	(ob1 ob2 → ob2 ob1)
04E314	~xSYLVESTER	([[]] → [D] [P])
39705	xSYSEVAL	(# → ?)
061314	~xTABVAL	(symb(x) {vals} → symb(x) {{vals} {res}})
060314	~xTABVAR	(symb(x) → symb(x) {{}{}} grob)
3EFB1	x->TAG	(ob tag → :tag:ob) UserRPL: →TAG
0520AB	xTAIL	({} → {}') ($ → $')
01C0DE	xTAN2CS2	(symb → symb')
021314	~xTAN2SC2	(symb → symb')
01F314	~xTAN2SC	(symb → symb')
3A70C	xTANH	(x → x')
3A624	xTAN	(x → x')
006314	~xTAYLOR0	(symb → symb')
3E6CA	xTAYLR	(symb var n → symb')
05B314	~xTCHEBYCHEFF	(n → pol)
01A314	~xTCOLLECT	(symb → symb')
0640AB	xTDELTA	(x y → x')
065314	~xTEVAL	(ob → ? time)
013314	~xTEXPAND	(symb → symb')
3C8FA	xTEXT	(→)
37F7F	xTHEN	(0/1 →)
39093	xTICKS	(→ #)
39124	xSETTIME	(time →) UserRPL: →TIME
3905D	xTIME	(→ time)
0650AB	xTINC	(x y → x')
3C6B6	xTLINE	((x1,y1) (x2,y2) →) ({#n1 #m1} {#n2 #m2} →)
019314	~xTLIN	(symb → symb')
3E97B	xTMENU	(% → [InitMenu%]) (Ob → [@LIST InitMenu])

Addr.	Name	Description
3DF4D	xTOT	(→ xsum)
		(→ {x1...xn})
0270AB	xTRACE	([[]] → x)
045314	˜xTRAN	([[]] → [[]]')
		(name →)
3EE0C	xTRANSIO	(n →)
01C314	˜xTRIGCOS	(symb → symb')
082314	˜xTRIGO	(→)
01D314	˜xTRIGSIN	(symb → symb')
01B314	˜xTRIG	(symb → symb')
01E314	˜xTRIGTAN	(symb → symb')
3C084	xTRN	([[]] → [[]]')
		(name →)
3AF3E	xTRNC	(x n →)
063314	˜xTRUNC	(symb1 symb2 → symb3)
3C99D	xTRUTH	(→)
015314	˜xTSIMP	(symb → symb')
391F8	xTSTR	(date time → $)
39456	xTVARS	(ntype → {})
		({n...} → {})
0470AB	xTVM	(→)
0480AB	xTVMBEG	(→)
0490AB	xTVMEND	(→)
04A0AB	xTVMROOT	(var → x)
3B2DC	x%T	(x y → 100y/x)
3BC39	xTYPE	(ob → %type)
38FD7	xUBASE	(u → u')
3900B	xUFACT	(u1 u2 → u3)
0140DD	xUFL1→MINIF	(ob n → font)
38FB5	x>UNIT	(x u → u')
		UserRPL: →UNIT
3F249	xUNPICK	(obn...ob1 ob n → ob...ob2)
3F22E	xUNROT	(1 2 3 → 3 1 2)
38195	xUNTIL	(→)
39420	xUPDIR	(→)
3E07D	xUTPC	(n x → x')
3E0BD	xUTPF	(n1 n2 x → x')
3E09D	xUTPN	(n v x → x')
3E0DD	xUTPT	(n x → x')

Addr.	Name	Description
38F81	xUVAL	(u → x)
3C2AC	xV>	([]/() → x y) ([]/() → x y z) (in current co-system) UserRPL: V→
3C2D6	x>V2	(x y → []) (x y → ()) UserRPL: →V2
3C30A	x>V3	(x y z → []) UserRPL: →V3
053314	˜xVANDERMONDE	({} → [[]])
3943B	xVARS	(→ {})
3DF68	xVAR	(→ x) (→ [x1...xn])
08C314	˜xVER	(→ $)
00F0AB	xVERSION	(→ $ $)
00A0DD	xVISITB	(name →)
0080DD	xVISIT	(name →)
3DB04	xMATCHDN	(symb {spat srepl} → symb' 0/1) (symb {spat srepl scond} → symb' 0/1) UserRPL: ↓MATCH
3BDB2	xVTYPE	(name → n)
3A442	xSQRT	(x → x') UserRPL: √
39819	xWAIT	(sec →) (0 → rc.p)
380DB	xWHILE	(→)
0080AB	xWIREFRAME	(→)
390AE	xWSLOG	(→ $ $ $ $)
3ABAF	xFACT	(x → x') UserRPL: !
3E03D	xXCOL	(n →)
0700AB	xXGET	(name →)
3EC35	xXMIT	($ → 1) ($ → $rest 0)
067314	˜xXNUM	(x → x')

Addr.	Name	Description
3CB7A	xXOR	(# #' → #'')
		($ $' → $'')
		(1/0 1/0 → 1/0)
3AD65	xXPON	(% →)
		(symb →)
0710AB	xXPUT	(name →)
068314	˜xXQ	(x → x')
0500AB	xXRECV	(name →)
3C915	xXRNG	(x1 x2 →)
3A278	xXROOT	(y x → Y')
06E0AB	xXSERV	(→)
04F0AB	xXSEND	(name →)
0000AB	xXVOL	(x1 x2 →)
0030AB	xXXRNG	(x1 x2 →)
39CFC	x-	(x y → x-y)
39F49	x/	(x y → x/y)
39DE8	x*	(x y → x*y)
3CF80	x<=?	(x y → 1)
		(x y → 0)
		UserRPL: ≤
3CD21	x#?	(x y → 1)
		(x y → 0)
		UserRPL: ≠
3D01F	x>=?	(x y → 1)
		(x y → 0)
		UserRPL: ≥
39B58	x+	(x y → x+y)
3CE42	x<	(x y → 1)
		(x y → 0)
3CBF6	x==	(x y → 1)
		(x y → 0)
3CEE1	x>	(x y → 1)
		(x y → 0)
398B9	x=	(x y → x=y)
3B251	x%	(x y → xy/100)
3E05D	xYCOL	(n →)
3C935	xYRNG	(y1 y2 →)
00B0AB	xYSLICE	(→)
0010AB	xYVOL	(y1 y2 →)

Addr.	Name	Description
3A097	x^	(y x → y^x)
0040AB	xYYRNG	(y1 y2 →)
040314	~xZEROS	(symb var → {zeros})
05F0AB	xZFACTOR	(xTr yPr → xZf)
0020AB	xZVOL	(x1 x2 →)

Appendix D
Library 256 and EXTABLE

D.1 The Development Library 256

Library 256 is built-in in the calculator, and contains several commands for the programmers. Some of this commands are described in section A.3. The list belows contains the commands and their rompointer address, should you want to use one of them in your program.

Addr.	Name	Description
000100	x→H	(ob → $hex)
001100	xH→	($hex → ob)
002100	x→A	(ob → hxs)
003100	xA→	(hxs → ob)
004100	xA→H	(hxs → $hex)
005100	xH→A	($hex → hxs)
006100	x→CD	($hex → code)
007100	xCD→	(code → $hex)
008100	xS→H	($ → $hex)
009100	xH→S	($hex → $)
00A100	x→LST	(comp → {})
		(ob1..obn %n → {})
00B100	x→ALG	(comp → symb)
		(ob1..obn %n → symb)
00C100	x→PRG	(comp → ::)
		(ob1..obn %n → ::)
00D100	xCOMP→	(comp → ob1...obn %n)
00E100	x→RAM	(ob → ob)
00F100	xSREV	($ → $')
010100	xPOKE	(hxs $hex →)
011100	xPEEK	(hxs1 hxs2 → $hex)
012100	xAPEEK	(hxs → hxs')

Addr.	Name	Description
013100	xR~SB	(% → #)
		(# → %)
014100	xSB~B	(# → hxs)
		(hxs → #)
015100	xLR~R	(%% → %)
		(% → %%)
016100	xS~N	($ → ID)
		(ID → $)
017100	xLC~C	(%%C → %C)
		(%C → %%C)
018100	xASM→	(Code → $)
019100	xBetaTesting	(→ $)
01A100	xCRLIB	(→ lib)
01B100	xCRC	($ → #crc)
01C100	xMAKESTR	(xlen → $)
01D100	xSERIAL	(→ $)
01E100	xASM	($ → ob)
01F100	xER	($ {errors} → $')
020100	x→S2	(ob → $)
021100	xXLIB~	(xlib xn → ROMPTR)
		(ROMPTR → xlib xn)

D.2 The EXTABLE Library

The EXTABLE library contains the table of entry points and their addresses. It contains some commands which might be useful in programs, also listed below. For a description of these commands, see section A.1.

Addr.	Name	Description
001102	xGETADR	($ → hxs)
		Get the address of an entry name.
002102	xGETNAME	(hxs → $)
		Get the entry name corresponding to an address.
003102	xGETNAMES	($start → {})
		Get all entry names which start with the given string.
004102	xGETNEAR	($sub → {})
		Get all entry names which contain the given string.

Appendix E
Error Messages

This appendix lists all “error” messages in the HP49G, even if most have nothing to do with errors. It is possible to generate an error with the following messages directly, with words such as `ERRORSTO` or `ERROROUT`, or recall them to the stack with `JstGetTHEMESG`. See Chapter 22.

All numbers listed are in hexadecimal format.

The symbol ➥ represents a line break.

#n	Message	#n	Message
1	Insufficient Memory	1D	Erase Fail, Locked Block
2	Directory Recursion	1E	Write Adr outside ROM
3	Undefined Local Name	1F	Write Fail, Rom Faulty
4	Undefined XLIB Name	20	Write Fail, Low bats
5	Memory Clear	21	Write Fail, Locked Block
6	Power Lost	101	No Room to Save Stack
7	Warning:	102	Can’t Edit Null Char.
8	Invalid Card Data	103	Invalid User Function
9	Object In Use	104	No Current Equation
A	Port Not Available	106	Invalid Syntax
B	No Room in Port	107	Real Number
C	Object Not in Port	108	Complex Number
D	Recovering Memory	109	String
E	Try To Recover Memory?	10A	Real Array
F	Replace RAM, Press ON	10B	Complex Array
10	No Mem To Config All	10C	List
11	Undefined FPTR Name	10D	Global Name
12	Invalid Bank Data	10E	Local Name
13	Full Check Bad Crc	10F	Program
14	Cmprs: not a user bank	110	Algebraic
15	No or 2 system bank	111	Binary Integer
16	Invalid bank	112	Graphic
17	Invalid bank number	113	Tagged
18	Inexisting pack	114	Unit
19	Pack twice	115	XLIB Name
1A	Ins. Mem.␣	116	Directory
1B	Erase Fail, Rom faulty	117	Library
1C	Erase Fail, Low bats	118	Backup

#n	Message
119	Function
11A	Command
11B	System Binary
11C	Long Real
11D	Long Complex
11E	Linked Array
11F	Character
120	Code
121	Library Data
122	External
124	LAST STACK Disabled
125	LAST CMD Disabled
126	HALT Not Allowed
127	Array
128	Wrong Argument Count
129	Circular Reference
12A	Directory Not Allowed
12B	Non-Empty Directory
12C	Invalid Definition
12D	Missing Library
12E	Invalid PPAR
12F	Non-Real Result
130	Unable to Isolate
131	No Room to Show Stack
132	Warning:➥
133	Error:
134	Purge?
135	Out of Memory
136	Stack
137	Last Stack
138	Last Commands
139	Key Assignments
13A	Alarms
13B	Last Arguments
13C	Name Conflict
13D	Command Line
13F	Interrupted
140	Integer
141	Symbolic Matrix
142	Font
143	Aplet
144	Extended Real
145	Extended Complex
146	FlashPtr
147	Extended Ptr
148	MiniFont
149	Extended 1

#n	Message
14A	Extended 2
14B	Extended 3
14C	YES
14D	NO
14E	TRUE
14F	FALSE
150	Are you sure?
151	Low Memory Condition➥Please Wait...
152	CATALOG
153	Nonexistent Find Pattern
154	Not Found
155	Nonexistent Replace Pattern
156	Can't Find Selection
157	Y= not available
158	Warning:➥Changes will not be saved
159	Result not editable in EQW
201	Too Few Arguments
202	Bad Argument Type
203	Bad Argument Value
204	Undefined Name
205	LASTARG Disabled
206	Incomplete➥Subexpression
207	Implicit () off
208	Implicit () on
301	Positive Underflow
302	Negative Underflow
303	Overflow
304	Undefined Result
305	Infinite Result
501	Invalid Dimension
502	Invalid Array Element
503	Deleting Row
504	Deleting Column
505	Inserting Row
506	Inserting Column
601	Invalid Σ Data
602	Nonexistent ΣDAT
603	Insufficient Σ Data
604	Invalid ΣPAR
605	Invalid Σ Data LN(Neg)
606	Invalid Σ Data LN(0)
607	Invalid EQ
608	Current equation:
609	No current equation.
60A	Enter eqn, press NEW
60B	Name the equation,➥press ENTER
60C	Select plot type

#n	Message
60D	Empty catalog
60E	undefined
60F	No stat data to plot
610	Autoscaling
611	Solving for␣
612	No current data. Enter
613	data point, press Σ+
614	Select a model
615	No alarms pending.
616	Press ALRM to create
617	Next alarm:
618	Past due alarm:
619	Acknowledged
61A	Enter alarm, press SET
61B	Select repeat interval
61C	␣␣␣␣I/O setup menu
61D	Plot type:␣
61E	""
61F	␣(OFF SCREEN)
620	Invalid PTYPE
621	Name the stat data,➥press ENTER
622	Enter value (zoom out➥if >1), press ENTER
623	Copied to stack
624	x axis zoom w/AUTO.➥
625	x axis zoom.➥
626	y axis zoom.➥
627	x and y axis zoom.➥
628	IR/wire:␣␣␣␣␣␣␣
629	ASCII/binary:␣␣
62A	baud:␣␣␣␣␣␣␣␣␣␣
62B	parity:␣␣␣␣␣␣␣␣
62C	checksum type:␣
62D	translate code:
62E	Enter matrix, then NEW
62F	No Associated Numeric View
701	Algebraic
702	RPN
703	Standard
704	Std
705	Fixed
706	Fix
707	Scientific
708	Sci
709	Engineering
70A	Eng
70B	Degrees
70C	Radians
70D	Grads
70E	Rectangular
70F	Polar
710	Spherical
711	Operating Mode. . .
712	Number Format. . . . . .
713	Angle Measure. . . . . .
714	Coord System. . . . .
715	FM,
716	Beep
717	Key Click
718	Last Stack
719	Choose calculator operating mode
71A	Choose number display format
71B	Choose decimal places to display
71C	Choose angle measure
71D	Choose coordinate system
71E	Use comma as fraction mark?
71F	Enable standard beep?
720	Enable key click?
721	Save last stk for UNDO and ANS?
722	CALCULATOR MODES
723	Font:
724	Stack:
725	Small
726	Textbook
727	Edit:
728	Small
729	Full Page
72A	Indent
72B	EQW:
72C	Small
72D	Small Stack Disp
72E	Header:
72F	Clock
730	Analog
731	Choose system font
732	Display stack using small font?
733	Use pretty print in the stack?
734	Edit using small font?
735	Edit in full page?
736	Automatically indent new lines?
737	Edit in EQW using small font?
738	Display EQW using small font?
739	Choose header height
73A	Display ticking clock?
73B	Analog clock?

#n	Message	#n	Message
73C	DISPLAY MODES	76B	Conic
73D	Indep var:	76C	Truth
73E	Modulo:	76D	Histogram
73F	Verbose	76E	Bar
740	Step/Step	76F	Scatter
741	Complex	770	Slopefield
742	Approx	771	Fast3D
743	Incr Pow	772	Wireframe
744	Simp Non-Rational	773	Ps-Contour
745	Rigorous	774	Y-Slice
746	Numeric	775	Gridmap
747	Enter independent variable name	776	Pr-Surface
748	Enter modulo value	777	Deg
749	Display calculus information?	778	Rad
74A	Perform operations step by step?	779	Grad
74B	Allow complex numbers?	77A	Type:
74C	Perform approx calculations?	77B	∡:
74D	Increasing polynomial ordering?	77C	EQ:
74E	Simplify non rational expr?	77D	Indep:
74F	Don't simplify |X| to X?	77E	Connect
750	Replace constants by values?	77F	Simult
751	CAS MODES	780	H-Tick:
752	Goto row:	781	V-Tick:
753	Goto column:	782	Pixels
754	Specify a row to go to	783	Depnd:
755	Specify a column to go to	784	Save Animation
756	Matrix Writer	785	ΣDAT:
757	Bad range value	786	Col:
758	Start:	787	Cols:
759	Step:	788	F:
75A	Type:	789	H-Var:
75B	Zoom:	78A	V-Var:
75C	Small Font	78B	Stiff
75D	File:	78C	∂F∂Y:
75E	Enter starting value	78D	∂F∂T:
75F	Enter increment value	78E	Choose type of plot
760	Choose table format	78F	Choose angle measure
761	Enter zoom factor	790	Enter function(s) to plot
762	Display table using small font?	791	Enter independent variable name
763	Enter a filename to save data	792	Connect plot points?
764	TABLE SETUP	793	Plot functions simultaneously?
765	Automatic	794	Enter horizontal tick spacing
766	Build Your Own	795	Enter vertical tick spacing
767	Function	796	Tick spacing units are pixels?
768	Polar	797	Enter dependent variable name
769	Parametric	798	Save slices animation?
76A	Diff Eq	799	Enter data to plot

#n	Message
79A	Enter col to use for horizontal
79B	Enter col to use for vertical
79C	Enter horizontal variable
79D	Enter vertical variable
79E	Use stiff diff eq solver?
79F	Enter derivative w.r.t. soln
7A0	Enter derivative w.r.t. indep
7A1	PLOT SETUP
7A2	H-View:
7A3	V-View:
7A4	Indep Low:
7A5	High:
7A6	Step:
7A7	Pixels
7A8	Depnd Low:
7A9	High:
7AA	X-Left:
7AB	X-Right:
7AC	Y-Near:
7AD	Y-Far:
7AE	Step Indep:
7AF	Depnd:
7B0	Bar Width:
7B1	Z-Low:
7B2	Z-High:
7B3	XE:
7B4	YE:
7B5	ZE:
7B6	Init:
7B7	Final:
7B8	Init-Soln:
7B9	Tol:
7BA	XXLeft:
7BB	XXRight:
7BC	YYNear:
7BD	YYFar:
7BE	Enter minimum horizontal value
7BF	Enter maximum horizontal value
7C0	Enter minimum vertical value
7C1	Enter maximum vertical value
7C2	Enter minimum indep var value
7C3	Enter maximum indep var value
7C4	Enter indep var increment
7C5	Indep step units are pixels?
7C6	Enter minimum depend var value
7C7	Enter maximum depend var value
7C8	Enter bar width

#n	Message
7C9	Enter minimum Z view-volume val
7CA	Enter maximum Z view-volume val
7CB	Enter X eyepoint coordinate
7CC	Enter Y eyepoint coordinate
7CD	Enter Z eyepoint coordinate
7CE	Enter absolute error tolerance
7CF	Enter minimum XX range value
7D0	Enter maximum XX range value
7D1	Enter minimum YY range value
7D2	Enter maximum YY range value
7D3	PLOT WINDOW
7D4	Default
7D5	FUNCTION
7D6	POLAR
7D7	PARAMETRIC
7D8	DIFF EQ
7D9	CONIC
7DA	TRUTH
7DB	HISTOGRAM
7DC	BAR
7DD	SCATTER
7DE	SLOPEFIELD
7DF	FAST3D
7E0	WIREFRAME
7E1	PS-CONTOUR
7E2	Y-SLICE
7E3	GRIDMAP
7E4	PR-SURFACE
7E5	PLOT WINDOW -␣
7E6	Enter minimum X view-volume val
7E7	Enter maximum X view-volume val
7E8	Enter minimum Y view-volume val
7E9	Enter maximum Y view-volume val
7EA	Enter indep var sample count
7EB	Enter depnd var sample count
7EC	Goto Level:
7ED	Specify a level to go to
7EE	HISTORY
801	Must be >= 0
802	Must be bewteen 0 and 1
803	μ0:
804	$\bar{x}$:
805	N:
806	α:
807	σ:
808	Null hypothesis population mean
809	Sample mean

#n	Message	#n	Message
80A	Sample Size	839	Sample Mean
80B	Significance level	83A	Significance level
80C	Population standard deviation	83B	Sample size
80D	Z-TEST: 1 μ, KNOWN σ	83C	T-TEST: 1 μ, UNKNOWN σ
80E	Alternative Hypothesis	83D	$\bar{x}1$:
80F	$\bar{x}1$:	83E	S1:
810	$\sigma 1$:	83F	N1:
811	N1:	840	α:
812	α:	841	$\bar{x}2$:
813	$\bar{x}2$:	842	S2:
814	$\sigma 2$:	843	N2:
815	N2:	844	Pooled?
816	Sample mean for population 1	845	Sample mean for population 1
817	Std deviation for population 1	846	Std deviation for sample 1
818	Sample size for population 1	847	Sample size for population 1
819	Significance level	848	Significance level
81A	Sample mean for population 2	849	Sample mean for population2
81B	Std deviation for population 2	84A	Std deviation for sample 2
81C	Sample size for population 2	84B	Sample size for population 2
81D	Z-TEST: 2 μ, KNOWN σ	84C	"Pooled" if checked
81E	$\pi 0$:	84D	T-TEST: 2 μ, UNKNOWN σ
81F	x:	84E	$\bar{x}$:
820	N:	84F	σ:
821	α:	850	N:
822	Null hyp. population proportion	851	C:
823	Success count	852	Sample mean
824	Sample size	853	Population standard deviation
825	Significance level	854	Sample size
826	Z-TEST: 1 P	855	Confidence level
827	X1:	856	CONF. INT.: 1 μ, KNOWN σ
828	N1:	857	$\bar{x}1$:
829	α:	858	$\sigma 1$:
82A	X2:	859	N1:
82B	N2:	85A	C:
82C	Success count for sample 1	85B	$\bar{x}2$:
82D	Size of sample 1	85C	$\sigma 2$:
82E	Significance level	85D	N2:
82F	Success count for sample 2	85E	Sample mean for population 1
830	Size of sample 2	85F	Std deviation for sample 1
831	Z-TEST: 2 P	860	Size of sample 1
832	$\bar{x}$:	861	Sample mean for population 2
833	Sx:	862	Std deviation for sample 2
834	$\mu 0$:	863	Size of sample 2
835	α:	864	Confidence level
836	N:	865	CONF. INT.: 2 μ, KNOWN σ
837	Null hypothesis population mean	866	x:
838	Sample Standard deviation	867	N:

#n	Message
868	C:
869	Sample success count
86A	Sample size
86B	Confidence level
86C	CONF. INT.: 1 P
86D	$\bar{x}1$:
86E	N1:
86F	C:
870	$\bar{x}2$:
871	N2:
872	Sample 1 success count
873	Sample 1 size
874	Sample 2 success count
875	Sample 2 size
876	Confidence level
877	CONF. INT.: 2 P
878	$\bar{x}$:
879	Sx:
87A	N:
87B	C:
87C	Sample mean
87D	Sample standard deviation
87E	Sample size
87F	Confidence level
880	CONF. INT.: 1 μ, UNKNOWN σ
881	$\bar{x}1$:
882	S1:
883	N1:
884	C:
885	$\bar{x}2$:
886	S2:
887	N2:
888	Pooled
889	Sample 1 mean
88A	Std deviation for sample 1
88B	Sample 1 size
88C	Sample 2 mean
88D	Std deviation for sample 2
88E	Sample 2 size
88F	Confidence level
890	Pooled if checked
891	CONF. INT.: 2 μ, UNKNOWN σ
892	Search for:
893	Replace by:
894	Case Sensitive
895	Search For:
896	Enter search pattern
897	Enter replace pattern
898	Case sensitive search?
899	Enter search pattern
89A	FIND REPLACE
89B	FIND
89C	Goto Line:
89D	Specify a line to go to
89E	GOTO LINE
89F	Goto Position:
8A0	Specify a position to go to
8A1	GOTO POSITION
8A2	H-Factor:
8A3	V-Factor:
8A4	Recenter on cursor
8A5	Enter horizontal zoom factor
8A6	Enter vertical zoom factor
8A7	Recenter plot on cursor?
8A8	ZOOM FACTOR
8A9	Object:
8AA	Name:
8AB	Directory
8AC	Enter New Object
8AD	Enter variable name
8AE	Create a new directory?
8AF	NEW VARIABLE
8B0	Select Object
901	[not shown because too long]
902	[not shown because too long]
903	[not shown because too long]
904	[not shown because too long]
905	[not shown because too long]
906	[not shown because too long]
907	[not shown because too long]
908	[not shown because too long]
909	[not shown because too long]
90A	[not shown because too long]
90B	[not shown because too long]
90C	[not shown because too long]
90D	Inconclusive result
A01	Bad Guess(es)
A02	Constant?
A03	Interrupted
A04	Zero
A05	Sign Reversal
A06	Extremum
A07	Left
A08	Right

#n	Message
A09	Expr
B01	Invalid Unit
B02	Inconsistent Units
C01	Bad Packet Block Check
C02	Timeout
C03	Receive Error
C04	Receive Buffer Overrun
C05	Parity Error
C06	Transfer Failed
C07	Protocol Error
C08	Invalid Server Cmd.
C09	Port Closed
C0A	Connecting
C0B	Retry #
C0C	Awaiting Server Cmd.
C0D	Sending␣
C0E	Receiving␣
C0F	Object Discarded
C10	Packet #
C11	Processing Command
C12	Invalid IOPAR
C13	Invalid PRTPAR
C14	Low Battery
C15	Empty Stack
C16	Row␣
C17	Invalid Name
D01	Invalid Date
D02	Invalid Time
D03	Invalid Repeat
D04	Nonexistent Alarm
B901	Press [CONT] for menu
B902	reset/delete this field
B903	Reset value
B904	Delete value
B905	Reset all
B906	Valid object types:
B907	Valid object type:
B908	Any object
B909	Real number
B90A	(Complex num)
B90B	"String"
B90C	[Real array]
B90D	[(Cmpl array)]
B90E	{ List }
B90F	Name
B910	« Program »
B911	'Algebraic'
B912	# Binary int
B913	_Unit object
B914	Invalid object type
B915	Invalid object value
B916	Calculator Modes
B917	Number Format:
B918	Angle Measure:
B919	Coord System:
B91A	Beep
B91B	Clock
B91C	FM,
B91D	Choose number display format
B91E	Enter decimal places to display
B91F	Choose angle measure
B920	Choose coordinate system
B921	Enable standard beep?
B922	Display ticking clock?
B923	Use comma as fraction mark?
B924	Standard
B925	Std
B926	Fixed
B927	Fix
B928	Scientific
B929	Sci
B92A	Engineering
B92B	Eng
B92C	Degrees
B92D	Deg
B92E	Radians
B92F	Rad
B930	Grads
B931	Grad
B932	Rectangular
B933	Polar
B934	Spherical
B935	SYSTEM FLAGS
B936	01 General solutions
B937	02 Constant → symb
B938	03 Function → symb
B939	14 Payment at end
B93A	19 →V2 → vector
B93B	20 Underflow → 0
B93C	21 Overflow → ±9E499
B93D	22 Infinite → error
B93E	27 'X+Y*i' → '(X,Y)'
B93F	28 Sequential plot
B940	29 Draw axes too

#n	Message	#n	Message
B941	31 Connect points	B970	95 RPN mode
B942	32 Solid cursor	B971	97 List:horiz disp
B943	35 ASCII transfer	B972	98 Vector:horiz disp
B944	36 RECV renames	B973	99 CAS:quiet
B945	37 Single-space prnt	B974	100 Step by step off
B946	38 Add linefeeds	B975	103 Complex off
B947	39 Show I/O messages	B976	105 Exact mode on
B948	40 Don't show clock	B977	106 Simp. in series
B949	41 12-hour clock	B978	109 Sym. factorize
B94A	42 mm/dd/yy format	B979	110 Normal matrices
B94B	43 Reschedule alarm	B97A	111 Simp non rat.
B94C	44 Delete alarm	B97B	112 i simplified
B94D	51 Fraction mark: .	B97C	113 Linear simp on
B94E	52 Show many lines	B97D	114 Disp 1+x → x+1
B94F	53 No extra parens	B97E	115 SQRT simplified
B950	54 Tiny element → 0	B97F	116 Prefer cos()
B951	55 Save last args	B980	117 CHOOSE boxes
B952	56 Standard beep on	B981	119 Rigorous on
B953	57 Alarm beep on	B982	120 Silent mode off
B954	58 Show INFO	B983	123 Allow Switch Mode
B955	59 Show variables	B984	125 Accur. Sign-Sturm
B956	60 [α][α] locks	B985	126 rref w/ last col
B957	61 [USR][USR] locks	B986	128 Cmplx var allowed
B958	62 User keys off	B987	01 Principal value
B959	63 Custom ENTER off	B988	02 Constant → num
B95A	65 All multiline	B989	03 Function → num
B95B	66 Stack:x lines str	B98A	14 Payment at begin
B95C	67 Digital clock	B98B	19 →V2 → complex
B95D	68 No AutoIndent	B98C	20 Underflow → error
B95E	69 Line edit	B98D	21 Overflow → error
B95F	70 →GROB 1 line str	B98E	22 Infinite → ±9E499
B960	71 Show addresses	B98F	27 'X+Y*i' → 'X+Y*i'
B961	72 Stack:current fnt	B990	28 Simultaneous plot
B962	73 Edit:current font	B991	29 Don't draw axes
B963	74 Right stack disp	B992	31 Plot points only
B964	75 Key click off	B993	32 Inverse cursor
B965	76 Purge confirm	B994	35 Binary transfer
B966	79 Textbook on	B995	36 RECV overwrites
B967	80 EQW cur stk font	B996	37 Double-space prnt
B968	81 GRB Alg cur font	B997	38 No linefeeds
B969	82 EQW edit cur font	B998	39 No I/O messages
B96A	83 Display grobs on	B999	40 Show clock
B96B	85 Normal stk disp	B99A	41 24-hour clock
B96C	90 CHOOSE:cur font	B99B	42 dd.mm.yy format
B96D	91 MTRW:matrix	B99C	43 Don't reschedule
B96E	92 MASD asm mode	B99D	44 Save alarm
B96F	94 Result = LASTCMD	B99E	51 Fraction mark: ,

#n	Message
B99F	52 Show one line
B9A0	53 Show all parens
B9A1	54 Use tiny element
B9A2	55 No last args
B9A3	56 Standard beep off
B9A4	57 Alarm beep off
B9A5	58 Don't show INFO
B9A6	59 Show names only
B9A7	60 [α] locks Alpha
B9A8	61 [USR] locks User
B9A9	62 User keys on
B9AA	63 Custom ENTER on
B9AB	65 Level 1 multiline
B9AC	66 Stk: 1 line str
B9AD	67 Analog clock
B9AE	68 AutoIndent
B9AF	69 Infinite line edit
B9B0	70 →GROB x lines str
B9B1	71 No addresses
B9B2	72 Stack:mini font
B9B3	73 Edit:mini font
B9B4	74 Left stack disp
B9B5	75 Key click on
B9B6	76 No purge confirm
B9B7	79 Textbook off
B9B8	80 EQW mini stk font
B9B9	81 GRB Alg mini font
B9BA	82 EQW edit mini fnt
B9BB	83 Display grobs off
B9BC	85 SysRPL stk disp
B9BD	90 CHOOSE:mini font
B9BE	91 MTRW:list of list
B9BF	92 MASD SysRPL mode
B9C0	94 Result <> LASTCMD
B9C1	95 Algebraic mode
B9C2	97 List:vert disp
B9C3	98 Vector:vert disp
B9C4	99 CAS:verbose
B9C5	100 Step by step on
B9C6	103 Complex on
B9C7	105 Approx. mode on
B9C8	106 !Simp. in series
B9C9	109 Num. factorize
B9CA	110 Large matrices
B9CB	111 !Simp non rat.
B9CC	112 i not simplified
B9CD	113 Linear simp off

#n	Message
B9CE	114 Disp x+1 → 1+x
B9CF	115 SQRT !simplified
B9D0	116 Prefer sin()
B9D1	117 Soft MENU
B9D2	119 Rigorous off
B9D3	120 Silent mode on
B9D4	123 Forb. Switch Mode
B9D5	125 FastSign-no Sturm
B9D6	126 rref w/o last col
B9D7	128 Vars are reals
B9D8	Object:
B9D9	Obs in
B9DA	Name:
BA01	1.Send to HP 49...
BA02	2.Get from HP 49
BA03	3.Print display
BA04	4.Print...
BA05	5.Transfer...
BA06	6.Start Server
BA07	Enter names of vars to send
BA08	Vars in
BA09	SEND TO HP 49
BA0A	Port:
BA0B	Dbl-Space
BA0C	Delay:
BA0D	Xlat:
BA0E	Linef
BA0F	Baud:
BA10	Parity:
BA11	Len:
BA12	Choose print port
BA13	Enter object(s) to print
BA14	Print extra space between lines?
BA15	Enter delay between lines
BA16	Choose character translations
BA17	Print linefeed between lines?
BA18	Choose baud rate
BA19	Choose parity
BA1A	Enter printer line length
BA1B	PRINT
BA1C	Type:
BA1D	OvrW
BA1E	Fmt:
BA1F	Chk:
BA20	Choose transfer port
BA21	Choose type of transfer
BA22	Enter names of vars to transfer

#n	Message
BA23	Choose transfer format
BA24	Choose checksum type
BA25	Overwrite existing variables?
BA26	TRANSFER
BA27	Local vars
BA28	Remote PC files
BA29	Files in␣
BA2A	Enter name of dir to change to
BA2B	Choose Remote Directory
BA2C	Infrared
BA2D	IR
BA2E	Wire
BA2F	Kermit
BA30	XModem
BA31	Odd
BA32	Even
BA33	Mark
BA34	Space
BA35	Spc
BA36	ASCII
BA37	ASC
BA38	Binary
BA39	Bin
BA3A	None
BA3B	Newline (Ch 10)
BA3C	Newl
BA3D	Chr 128-159
BA3E	$\rightarrow$159
BA3F	$\rightarrow$255
BA40	Chr 128-255
BA41	One-digit arith
BA42	Two-digit arith
BA43	Three-digit CRC
BB01	1.Single-var…
BB02	2.Frequencies…
BB03	3.Fit data…
BB04	4.Summary stats…
BB05	SINGLE-VARIABLE STATISTICS
BB06	ΣDAT:
BB07	Type:
BB08	Mean
BB09	Std Dev
BB0A	Variance
BB0B	Total
BB0C	Maximum
BB0D	Minimum
BB0E	Enter statistical data
BB0F	Enter variable column
BB10	Choose statistics type
BB11	Calculate mean?
BB12	Calculate standard deviation?
BB13	Calculate variance?
BB14	Calculate column total?
BB15	Calculate column maximum?
BB16	Calculate column minimum?
BB17	Sample
BB18	Population
BB19	FREQUENCIES
BB1A	X-Min:
BB1B	Bin Count:
BB1C	Bin Width:
BB1D	Enter minimum first bin X value
BB1E	Enter number of bins
BB1F	Enter bin width
BB20	FIT DATA
BB21	X-Col:
BB22	Y-Col:
BB23	Model:
BB24	Enter indep column number
BB25	Enter dependent column number
BB26	Choose statistical model
BB27	Correlation
BB28	Covariance
BB29	PREDICT VALUES
BB2A	Y:
BB2B	Enter indep value or press PRED
BB2C	Enter dep value or press PRED
BB2D	SUMMARY STATISTICS
BB2E	Calculate:
BB2F	ΣX
BB30	ΣY
BB31	ΣX2
BB32	ΣY2
BB33	ΣXY
BB34	NΣ
BB35	Calculate sum of X column?
BB36	Calculate sum of Y column?
BB37	Calculate sum of squares of X?
BB38	Calculate sum of squares of Y?
BB39	Calculate sum of products?
BB3A	Calculate number of data points?
BB3B	Linear Fit
BB3C	Logarithmic Fit
BB3D	Exponential Fit

#n	Message
BB3E	Power Fit
BB3F	Best Fit
BB40	5.Hypoth. tests...
BB41	6.Conf. interval...
BC01	1.Browse alarms...
BC02	2.Set alarm...
BC03	3.Set time, date...
BC04	SET ALARM
BC05	Message:
BC06	Time:
BC07	Date:
BC08	Repeat:
BC09	Enter "message" or « action »
BC0A	Enter hour
BC0B	Enter minute
BC0C	Enter second
BC0D	Choose AM, PM, or 24-hour time
BC0E	Enter month
BC0F	Enter day
BC10	Enter year
BC11	Enter alarm repeat multiple
BC12	Enter alarm repeat unit
BC13	SET TIME AND DATE
BC14	Choose date display format
BC15	Monday
BC16	Tuesday
BC17	Wednesday
BC18	Thursday
BC19	Friday
BC1A	Saturday
BC1B	Sunday
BC1C	None
BC1D	AM
BC1E	PM
BC1F	24-hour time
BC20	24-hr
BC21	␣1 January
BC22	␣2 February
BC23	␣3 March
BC24	␣4 April
BC25	␣5 May
BC26	␣6 June
BC27	␣7 July
BC28	␣8 August
BC29	␣9 September
BC2A	10 October
BC2B	11 November
BC2C	12 December
BC2D	Week
BC2E	Day
BC2F	Hour
BC30	Minute
BC31	Second
BC32	Weeks
BC33	Days
BC34	Hours
BC35	Minutes
BC36	Seconds
BC37	Month/Day/Year
BC38	M/D/Y
BC39	Day.Month.Year
BC3A	D.M.Y
BC3B	ALARMS
BD01	1.Integrate...
BD02	2.Differentiate...
BD03	3.Taylor poly...
BD04	4.Isolate var...
BD05	5.Solve quad...
BD06	6.Manip expr...
BD07	INTEGRATE
BD08	Expr:
BD09	Var:
BD0A	Result:
BD0B	Enter expression
BD0C	Enter variable name
BD0D	Enter lower limit
BD0E	Enter upper limit
BD0F	Choose result type
BD10	Choose disp format for accuracy
BD11	DIFFERENTIATE
BD12	Value:
BD13	Enter variable value
BD14	Expression
BD15	TAYLOR POLYNOMIAL
BD16	Order:
BD17	Enter Taylor polynomial order
BD18	ISOLATE A VARIABLE
BD19	Principal
BD1A	Get principal solution only?
BD1B	SOLVE QUADRATIC
BD1C	MANIPULATE EXPRESSION
BD1D	MATCH EXPRESSION
BD1E	Pattern:
BD1F	Replacement:

#n	Message
BD20	Subexpr First
BD21	Cond:
BD22	Enter pattern to search for
BD23	Enter replacement object
BD24	Search subexpressions first?
BD25	Enter conditional expression
BD26	Symbolic
BD27	Numeric
BE01	Plot
BE02	Type:
BE03	∡:
BE04	H-View:
BE05	Autoscale
BE06	V-View:
BE07	Choose type of plot
BE08	Choose angle measure
BE09	Enter function(s) to plot
BE0A	Enter minimum horizontal value
BE0B	Enter maximum horizontal value
BE0C	Autoscale vertical plot range?
BE0D	Enter minimum vertical value
BE0E	Enter maximum vertical value
BE0F	Plot (x(t), y(t))
BE10	Enter complex-valued func(s)
BE11	Plot y'(t)=f(t,y)
BE12	Enter function of INDEP and SOLN
BE13	Enter derivative w.r.t. SOLN
BE14	Enter derivative w.r.t. INDEP
BE15	Use Stiff diff eq solver?
BE16	ΣDat:
BE17	Col:
BE18	Wid:
BE19	Enter data to plot
BE1A	Arrays in
BE1B	Enter column to plot
BE1C	Enter bar width
BE1D	Cols:
BE1E	Enter col to use for horizontal
BE1F	Enter col to use for vertical
BE20	Steps:
BE21	Enter indep var sample count
BE22	Enter dep var sample count
BE23	Plot Options
BE24	Lo:
BE25	Hi:
BE26	Axes
BE27	Simult
BE28	Connect
BE29	Pixels
BE2A	H-Tick:
BE2B	V-Tick:
BE2C	Enter minimum indep var value
BE2D	Enter maximum indep var value
BE2E	Draw axes before plotting?
BE2F	Connect plot points?
BE30	Plot functions simultaneously?
BE31	Enter indep var increment
BE32	Indep step units are pixels?
BE33	Enter horizontal tick spacing
BE34	Enter vertical tick spacing
BE35	Tick spacing units are pixels?
BE36	Depnd:
BE37	Enter dependent var name
BE38	Enter minimum dep var value
BE39	Enter maximum dep var value
BE3A	H-Var:
BE3B	V-Var:
BE3C	Enter max indep var increment
BE3D	Choose horizontal variable
BE3E	Choose vertical variable
BE3F	0 INDEP
BE40	1 SOLN
BE41	␣SOLN(
BE42	X-Left:
BE43	X-Right:
BE44	Y-Near:
BE45	Y-Far:
BE46	Z-Low:
BE47	Z-High:
BE48	Enter minimum X view-volume val
BE49	Enter maximum X view-volume val
BE4A	Enter minimum Y view-volume val
BE4B	Enter maximum Y view-volume val
BE4C	Enter minimum Z view-volume val
BE4D	Enter maximum Z view-volume val
BE4E	XE:
BE4F	YE:
BE50	ZE:
BE51	Enter X eyepoint coordinate
BE52	Enter Y eyepoint coordinate
BE53	Enter Z eyepoint coordinate
BE54	Save Animation
BE55	Save animation data after plot?
BE56	XX-Left:

#n	Message
BE57	XX-Rght:
BE58	YY-Near:
BE59	YY-Far:
BE5A	Enter minimum XX range value
BE5B	Enter maximum XX range value
BE5C	Enter minimum YY range value
BE5D	Enter maximum YY range value
BE5E	XX and YY Plot Options
BE5F	Zoom Factors
BE60	H-Factor:
BE61	V-Factor:
BE62	Recenter at Crosshairs
BE63	Enter horizontal zoom factor
BE64	Enter vertical zoom factor
BE65	Recenter plot at crosshairs?
BE66	Reset plot
BE67	Dflt
BE68	Auto
BE69	Function
BE6A	Polar
BE6B	Conic
BE6C	Truth
BE6D	Parametric
BE6E	Diff Eq
BE6F	Histogram
BE70	Bar
BE71	Scatter
BE72	Slopefield
BE73	Wireframe
BE74	Ps-Contour
BE75	Y-Slice
BE76	Gridmap
BE77	Pr-Surface
BF01	1.Solve equation…
BF02	2.Solve diff eq…
BF03	3.Solve poly…
BF04	4.Solve lin sys…
BF05	5.Solve finance…
BF06	SOLVE EQUATION
BF07	Enter value or press SOLVE
BF08	Eq:
BF09	Enter function to solve
BF0A	Funcs in
BF0B	Solver Variable Order
BF0C	Variables:
BF0D	Enter order of vars to display
BF0E	SOLVE Y'(T)=F(T,Y)
BF0F	f:
BF10	∂f∂y:
BF11	∂f∂t:
BF12	Indep:
BF13	Init:
BF14	Final:
BF15	Soln:
BF16	Tol:
BF17	Step:
BF18	Stiff
BF19	Enter function of INDEP and SOLN
BF1A	Enter derivative w.r.t. SOLN
BF1B	Enter derivative w.r.t. INDEP
BF1C	Enter independent var name
BF1D	Enter initial indep var value
BF1E	Enter final indep var value
BF1F	Enter solution var name
BF20	Enter initial solution var value
BF21	Press SOLVE for final soln value
BF22	Enter absolute error tolerance
BF23	Enter initial step size
BF24	Calculate stiff differential?
BF26	Tolerance
BF27	Solution
BF28	SOLVE AN·X^N+…+A1·X+A0
BF29	Coefficients [an … a1 a0]:
BF2A	Roots:
BF2B	Enter coefficients or press SOLVE
BF2C	Enter roots or press SOLVE
BF2D	Coefficients
BF2E	Roots
BF2F	SOLVE SYSTEM A·X=B
BF30	A:
BF31	B:
BF32	X:
BF33	Enter coefficients matrix A
BF34	Enter constants or press SOLVE
BF35	Enter solutions or press SOLVE
BF36	Constants
BF37	Solutions
BF38	N:
BF39	I%YR:
BF3A	PV:
BF3B	PMT:
BF3C	P/YR:
BF3D	FV:
BF3E	Enter no. of payments or SOLVE

#n	Message
BF3F	Enter yearly int rate or SOLVE
BF40	Enter present value or SOLVE
BF41	Enter payment amount or SOLVE
BF42	Enter no. of payments per year
BF43	Enter future value or SOLVE
BF44	Choose when payments are made
BF45	TIME VALUE OF MONEY
BF47	I%/YR
BF48	PV
BF49	PMT
BF4A	FV
BF4B	End
BF4C	Begin
BF4D	Beg
BF4E	AMORTIZE
BF4F	Payments:
BF50	Principal:
BF51	Interest:
BF52	Balance:
BF53	Enter no. of payments to amort
BF54	Principal
BF55	Interest
BF56	Balance
C001	Unable to find root
DE01	␣denominator(s)␣
DE02	root(s)␣
DE03	last␣
DE04	obvious␣
DE05	factorizing␣
DE06	value␣
DE07	␣test(s)␣
DE08	searching␣
DE09	TAYLR of ↓ at␣
DE0A	nth␣
DE0B	␣is␣
DE0C	␣numerator(s)␣
DE0D	Less than␣
DE0E	multiplicity␣
DE0F	list of␣
DE10	␣at␣
DE11	factor(s)␣
DE12	Eigenvalues␣
DE13	Computing for
DE14	Root mult <
DE15	Numerical to symbolic
DE16	Invalid operator
DE17	Result:
DE18	Pivots
DE19	Press CONT to go on
DE1A	Test␣
DE1B	To be implemented
DE1C	Unable to factor
DE1D	Z is not = 1 mod 4
DE1E	Z is not prime
DE1F	Empty {} of equations
DE20	Not reducible to a rational expression
DE21	Non unary operator
DE22	User function
DE23	Non isolable operator
DE24	Not exact system
DE25	Parameters not allowed
DE26	CAS internal error
DE27	Invalid ˆ for SERIES
DE28	Operator not implemented (SERIES)
DE29	No variable in expr.
DE2A	No solution found
DE2B	Invalid derivation arg
DE2C	No solution in ring
DE2D	Not a linear system
DE2E	Can't derive int. var
DE2F	Diff equation order>2
DE30	INT:invalid var change
DE31	Mode switch cancelled
DE32	No name in expression
DE33	Invalid user function
DE34	Can't find ODE type
DE35	Integer too large
DE36	Unable to find sign
DE37	Non-symmetric matrix
DE38	ATAN insufficent order
DE39	ASIN at infinity undef
DE3A	Unsigned inf error
DE3B	LN[Var] comparison err
DE3C	Undef limit for var
DE3D	Bounded var error
DE3E	Got expr. indep of var
DE3F	Can't state remainder
DE40	LN of neg argument
DE41	Insufficient order
DE42	ABS of non-signed 0
DE43	Numeric input
DE44	Singularity! Continue?
DE45	Cancelled
DE46	Negative integer

#n	Message
DE47	Parameter is cur. var. dependent
DE48	Unsimplified sqrt
DE49	Non polynomial system
DE4A	Unable to solve ODE
DE4B	Array dimension too large
DE4C	Unable to reduce system
DE4D	Complex number not allowed
DE4E	Polyn. valuation must be 0
DE4F	Mode switch not allowed here
DE50	Non algebraic in expression
DE51	Purge current variable␣
DE52	Reduction result
DE53	Matrix not diagonalizable
DE54	Int[u'*F(u)] with u=
DE55	Int. by part u'*v, u=
DE56	Square root␣
DE57	Rational fraction␣
DE58	Linearizing␣
DE59	Risch alg. of tower␣
DE5A	Trig. fraction, u=
DE5B	Unknown operator (DOMAIN)
DE5C	Same points
DE5D	Unsigned inf. Solve?
DE5E	CAS not available
DE5F	Can not store current var
DE60	Not available on the HP40G
DE61	Not available on the HP49G
DE62	SERIES remainder is O(1) at order 3
DE63	Delta/Heaviside not available from HOME
DE64	Warning, integrating in approx mode
DE65	Function is constant
DE66	Can not unbind local vars
DE67	Replacing strict with large inequality
DE68	No valid environment stored
E101	Avogadro's number
E102	Boltzmann
E103	molar volume
E104	universal gas
E105	std temperature
E106	std pressure
E107	Stefan-Boltzmann
E108	speed of light
E109	permittivity
E10A	permeability
E10B	accel of gravity
E10C	gravitation
E10D	Planck's
E10E	Dirac's
E10F	electronic charge
E110	electron mass
E111	q/me ratio
E112	proton mass
E113	mp/me ratio
E114	fine structure
E115	mag flux quantum
E116	Faraday
E117	Rydberg
E118	Bohr radius
E119	Bohr magneton
E11A	nuclear magneton
E11B	photon wavelength
E11C	photon frequency
E11D	Compton wavelen
E11E	1 radian
E11F	2π radians
E120	∡ in trig mode
E121	Wien's
E122	k/q
E123	ε0/q
E124	q*ε0
E125	dielectric const
E126	SiO2 dielec cons
E127	ref intensity
E128	CONSTANTS LIBRARY
E129	Undefined Constant
E401	Invalid Mpar
E402	Single Equation
E403	EQ Invalid for MINIT
E404	Too Many Unknowns
E405	All Variables Known
E406	Illegal During MROOT
E407	Solving for␣
E408	Searching
E601	No Solution
E602	Many or No Solutions
E603	I%YR/PYR $\leq$ -100
E604	Invalid N
E605	Invalid PYR
E606	Invalid #Periods
E607	Undefined TVM Variable
E608	END mode
E609	BEGIN mode
E60A	␣payments/year
E60B	Principal

#n	Message	#n	Message
E60C	Interest	10105	Val betw 0-15 expected
E60D	Balance	10106	Val betw 1-16 expected
E701	NEAR␣	10107	Label Expected
E702	␣MINE␣␣	10108	Hexa Expected
E703	␣MINES␣␣	10109	Decimal Expected
E704	␣␣SCORE:␣	1010A	Can't Find
E705	YOU MADE IT!!	1010B	Label already defined
E706	YOU BLEW UP!!	1010C	{ expected
10001	Invalid $ROMID	1010D	} expected
10002	Invalid $TITLE	1010E	(expected
10003	Invalid $MESSAGE	1010F	Forbidden
10004	Invalid $VISIBLE	10110	Bad Expression
10005	Invalid $HIDDEN	10111	Jump too Long
10006	Invalid $EXTPRG	10112	Val betw 1-8 expected
10101	Invalid File	10113	Insuffisant Memory
10102	Too Many	10114	Matrix Error
10103	Unknown Instruction	10115	Define Error
10104	Invalid Field	31401	No Message here

Part VI

Index

Appendix F
Entries sorted by Name

The entries in this index have been sorted alphabetically, ignoring case. Leading characters ˆ, ˜, and x have no influence on the position of an entry. Entries starting with a digit or a symbol are at the end of the index. Note that for technical reasons, the page number given may be off by one for a few percent of the entries. If the page reference is 167, the entry may actually be the first entry on page 168.

Addr.	Name	Page	Addr.	Name	Page
02F0B3	~BBReDrawBackgr	245	33567	BINT112	15
03B0B3	~BBRereadChkEnbl	246	33571	BINT113	15
0250B3	~BBReReadCoords	245	3357B	BINT114	15
03C0B3	~BBRereadFullScr	246	33585	BINT115	15
0240B3	~BBReReadHeight	245	3358F	BINT116	15
03E0B3	~BBReReadNElems	246	33599	BINT117	15
0230B3	~BBReReadPageSize	245	335A3	BINT118	15
0260B3	~BBReReadWidth	245	335AD	BINT119	15
0290B3	~BBRunCanclAction	245	3317F	BINT12	11
0280B3	~BBRunENTERAction	245	335B7	BINT120	15
0220B3	~BBRunEntryProc	245	335C1	BINT121	15
0540B3	~BBSpace	246	335CB	BINT122	15
0520B3	~BBUpArrow	246	335D5	BINT123	15
39765	xBEEP	454	335DF	BINT124	15
071A2	BEGIN	150	335E9	BINT125	15
08F006	^Berlekamp	373	335F3	BINT126	16
08E006	^BerlekampP	373	335FD	BINT127	16
3EA006	^BESTDIV2	357	33607	BINT128	16
3E2C1	xBESTFIT	454	33611	BINT129	16
169006	^BESTMATRIXTYPE	65	33189	BINT13	11
019100	xBetaTesting	479	3361B	BINT130	16
4C8006	^BEZOUTMSOLV	396	3361B	BINT130d	16
0CF006	^BFactor	331	33625	BINT131	16
280006	^BICARREE?	382	33625	BINT131d	16
25FB3	BIGDISPN	281	33193	BINT14	11
25FB8	BIGDISPROW1	281	3319D	BINT15	11
25FBD	BIGDISPROW2	281	331A7	BINT16	11
25FC2	BIGDISPROW3	281	331B1	BINT17	11
25FC7	BIGDISPROW4	281	331BB	BINT18	11
3B655	xBIN	454	331C5	BINT19	11
074D0	BIND	116	3311B	BINT2	10
3E171	xBINS	454	331CF	BINT20	11
33107	BINT0	10	331D9	BINT21	11
33111	BINT1	10	331E3	BINT22	11
3316B	BINT10	10	331ED	BINT23	11
334EF	BINT100	15	331F7	BINT24	11
334F9	BINT101	15	33201	BINT25	11
33503	BINT102	15	3371F	BINT253	16
3350D	BINT103	15	33729	BINT255d	17
33517	BINT104	15	3320B	BINT26	11
33521	BINT105	15	33215	BINT27	11
3352B	BINT106	15	3321F	BINT28	11
33535	BINT107	15	33229	BINT29	11
3353F	BINT108	15	33125	BINT3	10
33549	BINT109	15	33233	BINT30	12
33175	BINT11	10	3323D	BINT31	12
33553	BINT110	15	33247	BINT32	12
3355D	BINT111	15	33251	BINT33	12

Addr.	Name	Page
2EFAC	bitAND	58
2EFB8	bitASR	58
2EFAF	bitNOT	58
2EFAD	bitOR	58
2EFB6	bitRL	58
2EFB7	bitRLB	58
2EFB4	bitRR	58
2EFB5	bitRRB	58
2EFB0	bitSL	58
2EFB1	bitSLB	58
2EFB2	bitSR	58
2EFB3	bitSRB	58
2EFAE	bitXOR	58
3C70A	xBLANK	454
2F16D	Blank$	48
2EF5E	BlankDA1	277
2EE5C	BlankDA12	277
2F31B	BlankDA2	277
2F31C	BlankDA2a	277
25E7E	BLANKIT	277
2F31D	BOTROW	279
3C6E0	xBOX	454
25E7F	Box/StdLabel	95
25E80	Box/StdLbl:	95
0D5006	^BRabin	332
0100E0	~BRbrowse	
0A5003	^BRDispItems	
0A4003	^BRdone	
0D0006	^BrentPow	331
03D0B3	~BReReadMenus	246
0AB003	^BRGetItem	
0A6003	^BRinverse	
0130E0	~BRoutput	
070004	^BrowseMem.1	
0190E0	~BRRclC1	246
0180E0	~BRRclCurRow	
0030E0	~BRStoC1	
0A7003	^BRViewItem	
3EE47	xBUFLEN	454
2F31E	BUILDKPACKET	
39480	xBYTES	455
261D4	C%%0=	38
27E2E	C%%1	36
05DBC	C%%>%%	37
261D9	C%%>C%	37
261DE	C%%CHS	38
261E3	C%%CONJ	38
51E006	^C%%SQRT	38

Addr.	Name	Page
27DBF	C%-1	36
2B2A7	C%-1=case	143
27DE4	C%0	36
261E8	C%0=	38
2B15D	C%0=case	142
27E09	C%1	36
25E81	C%1/	37
2B1C1	C%1=case	142
2B22A	C%2=case	143
2F31F	C%>#	22
05D2C	C%>%	31
25E82	C%>%%	37
25E83	C%>%%SWAP	37
18B006	^C%>C%%	37
25E84	C%ABS	37
25E85	C%ACOS	38
25E86	C%ACOSH	38
25E87	C%ALOG	38
25E88	C%ARG	38
25E89	C%ASIN	38
25E8A	C%ASINH	38
25E8B	C%ATAN	38
25E8C	C%ATANH	38
25E8F	C%C^C	37
25E90	C%C^R	37
261ED	C%CHS	37
261F2	C%CONJ	37
25E8D	C%COS	38
25E8E	C%COSH	38
25E91	C%EXP	38
25E92	C%LN	38
25E93	C%LOG	38
25E94	C%R^C	37
25E95	C%SGN	37
25E96	C%SIN	38
25E97	C%SINH	38
25E98	C%SQRT	37
25E99	C%TAN	38
25E9A	C%TANH	38
188006	^C2C%%	37
01E0DE	xC2P	455
25E9B	C>Im%	37
3C58E	xC>PX	456
3BAF5	xC>R	456
25E9C	C>Re%	37
34D00	CACHE	117
2EF72	CacheStack	
2EF91	CAL_CURS_POS	301

Addr.	Name	Page
33DBE	CHR_;	41
33DC5	CHR_<	41
33F4D	CHR_<<	43
33FBD	CHR_<=	43
33FCB	CHR_<>	43
33DCC	CHR_=	41
33DD3	CHR_>	41
33FC4	CHR_>=	43
33F54	CHR_>>	43
33FA1	CHR_[	42
33FA8	CHR_]	42
33E90	CHR_a	42
33DDA	CHR_A	41
33F5B	CHR_Angle	43
33DE1	CHR_B	41
33E97	CHR_b	42
33DE8	CHR_C	41
33E9E	CHR_c	42
33EA5	CHR_d	42
33DEF	CHR_D	42
33D39	CHR_DblQuote	41
33F62	CHR_Deriv	43
33EAC	CHR_e	42
33DF6	CHR_E	42
33EB3	CHR_f	42
33DFD	CHR_F	42
33E04	CHR_G	42
33EBA	CHR_g	42
33EC1	CHR_h	42
33E0B	CHR_H	42
33E12	CHR_I	42
33EC8	CHR_i	42
33F69	CHR_Integral	43
33E19	CHR_J	42
33ECF	CHR_j	42
33ED6	CHR_k	42
33E20	CHR_K	42
33EDD	CHR_l	42
33E27	CHR_L	42
33F70	CHR_LeftPar	41
33E2E	CHR_M	42
33EE4	CHR_m	43
33EEB	CHR_n	43
33E35	CHR_N	42
33F77	CHR_Newline	41
33E3C	CHR_O	42
33EF2	CHR_o	43
33E43	CHR_P	42
33EF9	CHR_p	43
33F7E	CHR_Pi	43
33E4A	CHR_Q	42
33F00	CHR_q	43
33F07	CHR_r	43
33E51	CHR_R	42
33F85	CHR_RightPar	41
33F0E	CHR_s	43
33E58	CHR_S	42
33F8C	CHR_Sigma	43
33F93	CHR_Space	41
33F15	CHR_t	43
33E5F	CHR_T	42
33E66	CHR_U	42
33F1C	CHR_u	43
33F9A	CHR_UndScore	42
33E6D	CHR_V	42
33F23	CHR_v	43
33F2A	CHR_w	43
33E74	CHR_W	42
33F31	CHR_x	43
33E7B	CHR_X	42
33E82	CHR_Y	42
33F38	CHR_y	43
33E89	CHR_Z	42
33F3F	CHR_z	43
33FAF	CHR_{	43
33FB6	CHR_}	43
01D0DE	xCIRC	455
2F215	CircleB	93
2F216	CircleG1	93
2F217	CircleG2	93
2F218	CircleW	93
2F219	CircleXor	93
2F13D	CK#-	23
51D006	^CK%%SQRT	34
203006	^CK&CONV2INT	328
202006	^CK&CONVINT	328
25EA7	Ck&DecKeyLoc	205
2631E	CK&DISPATCH0	197
26328	CK&DISPATCH1	197
26323	CK&DISPATCH2	197
007002	^Ck&DoMsgBox	283
25EA8	Ck&Freeze	
262B0	CK0	196
25E9D	CK0ATTNABORT	207
26292	CK0NOLASTWD	196
262B5	CK1	196

Addr.	Name	Page
2EE62	DA1OK?	275
2BF3A	DA1OK?NOTIT	275
2EEB1	DA2aBad?	276
2EE66	DA2aLess1OK?	275
2BF53	DA2aOK?NOTIT	275
2EEB2	DA2bBad?	276
2EE7E	DA2bIsEdL?	276
2EEB7	DA2bNoCh?	276
2BF6C	DA2bOK?NOTIT	275
2EEA6	DA2bTemp?	
2EEB4	DA3Bad?	276
2EE63	DA3OK?	275
2BF85	DA3OK?NOTIT	275
29EE9	DaDGNTc	
0610AB	xDARCY	456
36E07	dARRYcase	145
39078	xDATE	456
2EED0	DATE	173
39238	xDATE+	456
2EED2	DATE+DAYS	173
2F329	Date>d$	173
2F1AB	Date>hxs13	173
2DEBB	DAY#	
2DD27	Day>Date	
0690AB	xdB	456
0150DD	xDBUG	456
2F190	DcompWidth@	51
2EED1	DDAYS	173
39218	xDDAYS	456
00C007	^DEB.MATRIX	
00D007	^DEB.MATRIXTYPE	
3B670	xDEC	456
0B1006	^DeCntMulti	375
2F32A	DECODE	
2A893	Decomp#Disp	52
2A878	Decomp#Line	52
25EAA	DECOMP$	53
2F1BF	Decomp%Short	54
2A842	Decomp1Line	52
2A8AE	DecompEcho	54
2A85D	DecompEdit	53
2A8C9	DecompStd1Line	52
2A8E4	DecompStd1Line32	52
3E576	xDECR	456
0370DE	xDEDICACE	
041ED	DEEPSLEEP	178
0250DE	xDEF	
3E85C	xDEFINE	456
3B549	xDEG	456
288006	^DEG1	382
285006	^DEG2ext	382
0360DE	xDEGREE	
3E0006	^DEGREEext	378
2EF95	DEL_CMD	303
391D8	xDELALARM	457
3D1C7	xDELAY	457
29D6A	delimcase	
3EF3B	xDELKEYS	457
0F3007	^DELTAPSOLVE	385
0BA006	^DemonicLf	375
2A5006	^DENOLCMext	358
3C51F	xDEPND	457
3DCA7	xDEPTH	457
0314C	DEPTH	107
4F2006	^DEPTHext	398
4F3006	^DEPTHOBJext	398
80EDC	DEPTHSAVE	
3D258	xDER	
1B3006	^DERARG	390
00E314	~xDERIV	457
1A5006	^DERIV	388
1A1006	^DERIVext	387
1A3006	^DERIVIDNT	388
1A4006	^DERIVIDNT1	388
2C0ED	derprod1	
2C0A7	derquot	
1DD006	^DERVX	412
003314	~xDERVX	457
00F314	~xDESOLVE	457
07E007	^DESOLVE	391
3B1BA	xDET	457
3EB84	xDETACH	457
2F7006	^deuxipi	420
03B0AB	xDIAG→	457
00C0DE	xDIAGMAP	
36E93	dIDNTNcase	145
084314	~xDIFF	457
00E0AB	xDIFFEQ	457
16E006	^DIMLIMITS	64
345006	^DIMRANM	337
004007	^DIMS	
38BAE	xDIR	
39725	xDISP	457
25EBC	Disp5x7	282
25FB8	DISP@01	281
25FBD	DISP@09	281

Addr.	Name	Page	Addr.	Name	Page
25EC2	DoDelims	302	2EECC	DOPRLCD	
25EB4	DODISP	281	2EF01	DOPX>C	318
2604E	DOENG	177	30A66	DORANDOMIZE	33
2EEF9	DOERASE	273	25EB5	DORCLE	318
39527	xDOERR	457	2F2E9	DOREPL	308
2F2DD	DoFarBS	303	2F2EB	DOREPLACE	308
2F2DE	DoFarDel	303	2F2EC	DOREPLACE/NEXT	308
2F2E8	DOFIND	308	2AC0E	DoRunSafe	177
2EEBD	DOFINISH	180	2EEC6	DOSBRK	180
25EC3	DoFirstRow	294	26058	DOSCI	177
26053	DOFIX	177	25ECC	DoSolvrMenu	
2EF60	DOGRAPHIC	318	2EEC7	DOSRECV	180
25EC4	DoHere:	168	2605D	DOSTD	177
2EFA5	DOHEX	177	2565A	DoStdKeys?	224
2C371	DoInputForm	257	3B76C	DOSTOALLF2	176
2EEC4	DOKERRM	180	25EB6	DOSTOE	318
0AF002	^DoKeyCancel		2F23E	DOSTOSYSF	175
0B5002	^DoKeyEdit		25EB7	DOSTR>	50
25EC5	DoKeyOb	206	2EFAA	dostws	57
0B4002	^DoKeyOK		0540AB	xDOSUBS	458
25886	DoLabel	293, 95	3B1E1	xDOT	458
2EF03	DOLCD>	92	2F2DB	DOTEXTINFO	312
05B0AB	xDOLIST	457	2EEC3	DOTRANSIO	180
0B2006	^DoLS	375	25EB8	DOTVARS%	168
0210DE	xDOMAIN		0BD002	^DOTVARS{}	169
25EC6	DoMenuKey	291	25EB9	DOVARS	168
2589A	DoMenuKeyNS	293	2F2D4	dowait	172
026FE	DOMINIFONT		2EE60	DoWarning	282
0AE002	^DoMKeyOK		25EBA	DPRADIX?	178
0000B1	~DoMsgBox		3C484	xDRAW	458
25EC7	DoNameKeyLRS		06B0AB	xDRAW3DMATRIX	458
25EC8	DoNameKeyRS		2F32C	DRAWBOX#	92
2F192	DoNewEqw		2F13F	DRAWLINE#3	92
2F142	DoNewMatrix		2F32D	drax	
008007	^DoNewMatrixCplx		3C4BA	xDRAX	458
007007	^DoNewMatrixReal		36EA7	dREALNcase	145
00B007	^DoNewMatrixRealOrCplx		0230DE	xDROITE	
2F2EA	DONEXT	308	3DC3B	xDROP	458
25EC9	DoNextRow		03244	DROP	106
2EFA7	DOOCT	177	3683D	DROP#1-	24
2F13C	DoOldMatrix		282CC	DROP%0	31
00A007	^DoOldMatrixCplx		36996	DROP'	130
009007	^DoOldMatrixReal		3DC56	xDROP2	458
2EEC0	DOOPENIO		2F1C6	DROP3PICK	109
2EEC2	DOPARITY	180	28ACE	DROP?symcomp	86
2EEBE	DOPKT	180	25ECD	DropBadKey	207
25ECA	DoPlotMenu		2F32E	DROPDEADTRUE	130
25ECB	DoPrevRow		357CE	DROPDUP	106

Addr.	Name	Page
35190	DUPTYPELIST?	199
350AF	DUPTYPELNGCMP?	200
350A0	DUPTYPELNGREAL?	200
35118	DUPTYPEREAL?	199
35145	DUPTYPEROMP?	200
35154	DUPTYPERRP?	200
0CC006	^DupTypeS?	335
35163	DUPTYPESYMB?	199
3519F	DUPTYPETAG?	200
183006	^DUPTYPEZ?	200
35091	DUPTYPEZINT?	200
3457F	DUPUNROT	106, 107
36AD6	DUPZERO	21
10A006	^DupZIsEven?	334
0FB006	^DupZIsNeg?	334
0F9006	^DupZIsOne?	334
109006	^DupZIsTwo?	334
36A77	dvarlsBIND	117
00003	DZP	
18C006	^E%%>C%%	37
2C121	easyabs	
2EEE4	Echo$Key	302
2F11C	Echo$NoChr00	302
25ED1	Echo2Macros	
2EEE3	EchoChrKey	
039EF	ECUSER	
25F10	ederr	158
0070DD	xEDIT	458
0090DD	xEDITB	458
25ECE	EDITDECOMP$	53
25F11	editdecomp$w	53
36E57	EditExstCase	146
2F1A9	EDITF	
2F1A8	EditFont	
2EEE5	EditLevel1	311
257A2	EditLExists?	300
806FD	EDITLINE	
2EEEB	EDITLINE$	300
25ED2	EditMenu	311
2EEEC	EDITPARTS	
2F2DF	EditSelect	309
2EEE9	EditString	310
02E314	~xEGCD	458
3D8006	^EGCDext	380
3DB006	^EGCDNEWG	
3DA006	^EGCDSWAP	
02C0AB	xEGV	458
02D0AB	xEGVL	458
33157	EIGHT	10
331BB	EIGHTEEN	11
342BB	EIGHTROLL	108
33427	EIGHTY	14
33431	EIGHTYONE	14
33175	ELEVEN	10
4F9006	^ELMGext	399
3805D	xELSE	458
371B3	Embedded?	69
2F330	ENCODE	
2F331	ENCODE1PKT	
38A54	xENDDO	458
0570AB	xENDSUB	458
38A14	xENDTIC	
3B5DA	xENG	458
088314	~xEPSX0	458
03B2E	EQ	137
3663A	EQ:	137
3BDE6	xEQ>	458
34999	EQcase	144
34920	EQcasedrop	144
2F332	EQCURSOR?	
36EBB	EQIT	144
36F01	EQITE	144
25ED3	EqList?	73
37829	EQLookup	70
3664E	EQOR	137
3607F	EQOVER	137
03B97	EQUAL	137
36D62	EQUALcase	144
2AD81	EQUALcasedrop	145
36D08	EQUALcasedrp	144
3660D	EQUALNOT	137
36E7F	EQUALNOTcase	144
36662	EQUALOR	137
3C0006	^EQUALPOS2META	78
3776B	EQUALPOSCOMP	69
3BF006	^EQUALPOSMETA	78
25ECF	EQUATION	318
462006	^EQUATION?	86
4BF006	^EQUIV!	387
00B0DD	xEQW	458
010004	^EQW3	
01D004	^EQW3Code	
01C004	^EQW3CursorOff	
01B004	^EQW3CursorOn	
011004	^EQW3Edit	311
019004	^EQW3GROB	98

Addr.	Name	Page
27F006	^FRND	382
267006	^FROMLISText	418
042314	~xFROOTS	460
21E006	^FROOTS	415
3B509	xFS?	460
3B615	xFS?C	460
2621F	FSCANFONT	
001004	^FSTR1	54
00A004	^FSTR10	
00B004	^FSTR11	
00C004	^FSTR12	
00D004	^FSTR13	55
002004	^FSTR2	
003004	^FSTR3	54
004004	^FSTR4	54
005004	^FSTR5	54
006004	^FSTR6	54
007004	^FSTR7	55
008004	^FSTR8	
009004	^FSTR9	55
0FE007	^FTAYL	393
0AE007	^FULLDATA	407
3C955	xFUNCTION	460
06B314	~xFXND	460
164006	^FXNDext	380
0070DE	xGAMMA	460
05F42	GARBAGE	178
383006	^GAUSS	346
04D314	~xGAUSS	460
095006	^GBASIS	384
26076	GBUFF	272
25ED5	GBUFFGROBDIM	273
02C314	~xGCD	460
24A006	^GCD1MOD	
2B4006	^GCDext	
0EC006	^GCDHEUext	399
075314	~xGCDMOD	460
2F105	GDISPCENTER	318
3C1C7	xGET	460
2F2F3	GET.W->	308
2F2F4	GET.W<-	308
34504	get1	77
2F2F6	GET_CUR_FONT.EXT	312
314E4	GETAB0	
314CA	GETAB1	
001102	xGETADR	479
0371D	GETATELN	64
2DDD5	getBPOFF	
31518	GETCD0	
2F338	GetChkPRTPAR	
04A41	GETDF	288
26224	GetElemBotVStack	163
26229	GetElemTopVStack	163
2F339	GetEqN	318
57D006	^GETERABLEMSG	403
04E07	GETEXITMSG	156
0BB002	^GetFieldVals	
26238	GetFontCmdHeight	
2623D	GetFontHeight	
26242	GetFontStkHeight	283
26247	GetHeader	273
3C22D	xGETI	460
2F106	GETINDEP	317
2EEBF	GetIOPAR	181
2F33A	GetKermPkt#	
25ED6	GETKEY	206
25ED7	GETKEY*	206
25ED9	GetKeyOb	206
2F33B	GETKP	
075A5	GETLAM	117
3483E	GETLAMPAIR	119
2EF6D	GetLastEdit	
2F33C	getmatchtok	51
25EDA	GetMenu%	289
2624C	GetMetaVStack	162
25F17	GetMetaVStackDROP	161
002102	xGETNAME	479
2EEBC	GETNAME	180
003102	xGETNAMES	479
004102	xGETNEAR	479
25EDB	GetNextToken	50
2F33D	GETPARAM	316
2F107	GETPMIN&MAX	317
04A0B	GETPROC	288
2EEF5	GETPTYPE	317
2F10D	GETRES	317
2F10A	GetRes	
2F108	GETRHS	
07A007	^GetRoot	419
2F33E	GETSCALE	317
2F33F	GETSERIAL	
04D64	GETTHEMESG	157
04714	GETTOUCH	206
25967	GetUserKeys	208
25F18	GetVStack	161
26233	GetVStackProtectWord	164

Addr.	Name	Page
2EFCA	HXS>%	31
2EFCE	HXS>=HXS	59
2EFCD	HXS>HXS	59
336C5	HXSREAL	16
417006	^HYP2EXPext	356
02B0DE	xHYPERBOLIC	
104007	^HYPERGEO	393
39B3B	xi	462
3F0B7	xI>R	462
20C006	^IABCUV	414
031314	~xIABCUV	461
0120DE	xIBASIS	
10D007	^IBERNOULLI	394
0060DE	xIBERNOULLI	461
00B314	~xIBP	461
2C5006	^IBP	390
03B314	~xICHINREM	461
217006	^ICHINREM	414
2F0EC	ICMPDRPRTDRP	71
33143	id	10
05BE9	ID>$	46
2E11B	Id>Menu	293, 95
05F2E	ID>TAG	61
272F3	ID_EQ	116
27937	ID_SIGMADAT	116
3E9006	^IDIV2	
027314	~xIDIV2	461
33887	IDLISTOB	19
3C02E	xIDN	462
33143	idnt	10
2B0CC	idntcase	145
45C006	^IDNTEXEC	423
191006	^IDNTLAM?	201
2B0EF	idntlamcase	145
334D1	IDREAL	15
3387D	IDREALOB	18
0716B	IDUP	128, 150
20A006	^IEGCD	
02F314	~xIEGCD	462
3D6006	^IEGCDext	330
37F48	xIF	462
034004	^IfCheckFieldtype	259
033004	^IfCheckSetValue	259
049004	^IfCreateTitleGrob	
031004	^IfDisplayFromData	259
046004	^IfDisplayFromData2	
0BC002	^IFEDispField	
3807D	xIFEND	
03D004	^IfEnterKeyPress	260
387AC	xIFERR	462
01B0AB	xIFFT	462
027004	^IfGetCurrentFieldValue	258
02C004	^IfGetFieldChooseData	258
02D004	^IfGetFieldChooseDecomp	259
02B004	^IfGetFieldDecompObject	258
030004	^IfGetFieldInternalVa..	259
028004	^IfGetFieldMessageHan..	258
02A004	^IfGetFieldObjectsType	258
045004	^IfGetFieldPos	261
02E004	^IfGetFieldResetValue	259
029004	^IfGetFieldType	258
026004	^IfGetFieldValue	258
032004	^IfGetNbFields	259
04C004	^IfGetPrlgFromTypes	259
04A004	^IfInitDepth	261
03A004	^IfKeyCalc	260
037004	^IfKeyChoose	259
038004	^IfKeyEdit	260
03B004	^IfKeyInvertCheck	260
039004	^IfKeyTypes	260
020004	^IfMain	257
042004	^IfMain2	261
0050B0	~IFMenuRow1	257
0060B0	~IFMenuRow2	257
03C004	^IfONKeyPress	260
043004	^IfPutFieldsOnStack	261
035004	^IfReset	259
048004	^IfSetAllHelpStrings	261
047004	^IfSetAllLabelsMessages	261
025004	^IfSetCurrentFieldValue	258
036004	^IfSetField	259
044004	^IfSetFieldPos	261
02F004	^IfSetFieldResetValue	259
024004	^IfSetFieldValue	258
021004	^IfSetFieldVisible	258
023004	^IfSetGrob	258
03F004	^IfSetHelpString	260
022004	^IfSetSelected	258
040004	^IfSetTitle	260
041004	^IfSetTitle2	
396A4	xIFT	462
395F3	xIFTE	462
04B004	^IfTet	
011314	~xILAP	462
08A007	^ILAPDELTA	
08B007	^ILAPEXP	392

Addr.	Name	Page
101007	^LINEARAPPLY	425
102007	^linearapply	
2556A	LineB	93
2F353	LINECHANGE	
2556F	LineG1	93
25574	LineG2	93
2EFA0	LINEOFF	92
2EFA3	LINEOFF3	92
2EF9F	LINEON	92
2EFA2	LINEON3	92
25565	LineW	93
25579	LineXor	93
31B006	^LINEXPA	355
300006	^LINEXPext	355
3E214	xLINFIT	463
0150AB	xLININ	463
080007	^LINSOLV	342
052314	~xLINSOLVE	463
33139	list	10
2F354	List	
2DC006	^LIST10-10	
2DB006	^LIST1i-1-i	
17A006	^LIST2MATRIX	338
3BAC1	xLIST>	463
3343B	LISTCMP	14
45F006	^LISTEXEC	421
460006	^LISTEXEC1	421
2DA006	^LISTIRRQ	422
3346D	LISTLAM	14
104006	^LISTMAXext	399
4EE006	^LISTOPext	398
4ED006	^LISTOPRAC	398
4EF006	^LISTOPSQRT	398
0FD006	^ListPos	69
2F24E	LISTRCL	166
33431	LISTREAL	14
26B006	^LISTSECOext	421
001007	^ListToArry	65
4E5006	^LLVARDext	397
3AA01	xLN	464
45A006	^LN2ATAN	352
456006	^LN2ext	357
06D314	~xLNAME	464
527006	^LNATANext	349
31D006	^LNCOLCext	355
016314	~xLNCOLLECT	464
316006	^LNEXPA	355
318006	^LNEXPA*	369
319006	^LNEXPA/	369
31A006	^LNEXPA^	369
522006	^xLNext	349
4AB006	^LNOBJ!	
3AB2F	xLNP1	464
41A006	^LNP12LN	351
25EE8	LoadTouchTbl	292
25EE9	LockAlpha	278
3AA73	xLOG	464
41B006	^LOG2LN	351
3E239	xLOGFIT	464
377C5	Lookup	70
377DE	Lookup.1	70
07334	LOOP	151
05149	Loop	
269006	^LOP1ext	421
26A006	^LOPAext	421
3F2006	^LOPDext	420
10F006	^LOPMext	421
01118	LowBat?	178
2B1006	^LPGCDext	358
49D006	^LPROF!	
0320AB	xLQ	464
3DF83	xLR	464
4AE006	^LRDM!	
3DE006	^LRDMext	378
015100	xLR~R	479
02B0AB	xLSQ	464
0300AB	xLU	464
06A314	~xLVAR	464
4F0006	^LVARDext	398
4E7006	^LVARext	397
4C0006	^LVARXNX2!	387
4E3006	^LVARXNX2ext	
4DF006	^LVARXNXext	397
386006	^m-1&m+1	359
2AE32	M-1stcasechs	143
35AE2	MACRODCMP	
051314	~xMAD	464
321006	^MADD	339
242006	^MADDTMOD	416
37E006	^MADJ	345
229006	^MADNOCK	415
07F314	~xMAIN	
377006	^make2dmatrix	337
376006	^MAKE2DMATRIX	337
373006	^MAKEARRY	337
2E189	MakeBoxLabel	95

Addr.	Name	Page
36725	OVER#>	26
369AA	OVER'	130
36CF4	OVER5PICK	109
36183	OVERARSIZE	64
35CF4	OVERDUP	109
36482	OVERINDEX@	152
05622	OVERLEN$	47
35D6C	OVERSWAP	109, 109
35D6C	OVERUNROT	109, 109
351FA	OverWrF/TLp	
01F0DE	xP2C	
0D8007	^P2P#	374
215006	^PA2B2	330
039314	~xPA2B2	466
2EF62	palparse	50
35B46	PALPTRDCMP	
35B82	palrompdcmp	55
3C98B	xPARAMETRIC	466
3EDEC	xPARITY	466
0CD007	^PARITYTEST	
2B475	ParOuterLoop	223
2EF6A	Parse.1	
2EF6B	Parse.2	
2EF6E	ParseFail	51
2EF70	ParseFail2	
0090AB	xPARSURFACE	466
3D2006	^PARTFRAC	391
034314	~xPARTFRAC	466
3D4006	^PARTFRACRAT	
0F2007	^PASCAL_NEXTLINE	385
2EF94	PASTE.EXT	307
393EA	xPATH	466
25EF1	PATHDIR	169
228006	^PCAR	346
04F314	~xPCAR	466
0450AB	xPCOEF	466
00D0AB	xPCONTOUR	466
01F0AB	xPCOV	466
02E0E7	~PCunpack	
3DC006	^PDer	387
3C4F5	xPDIM	466
3F8006	^PDIV2ext	372
0C6006	^PDivLk	377
011100	xPEEK	478
209006	^PEGCD	414
3B477	xPERM	466
37B006	^PEVAL	345
0460AB	xPEVAL	466
2C1006	^PEval	424
0EA006	^PEvalFast?	380
0E1006	^PEvalMod	395
0DA007	^PEvalN/D	394
0AC006	^PFactor	374
0C5006	^PFactPowCnt	377
0C3006	^PFactTriv	376
268006	^PFEXECext	421
3D5006	^PFext	
3EAA7	xPGDIR	466
0AE006	^PHFctr	374
0B0006	^PHFctr0	374
0AF006	^PHFctr1	374
2EA006	^pi	420
39AC7	xPI	466
30017	PI/180	30
438006	^pi/2-acos	366
43B006	^pi/2-asin	366
439006	^pi/2-meta	366
3DCFD	xPICK	466
032E2	PICK	109
3F27F	xPICK3	466
3C72A	xPICT	466
2F258	PICTRCL	316
28A006	^PIext	73
2F6006	^pifois2	420
06A0AB	xPINIT	466
2E9006	^pisur-2	420
2E8006	^pisur2	420
354006	^PIVOTFLOAT	341
353006	^PIVOTNORM	
3C662	xPIX?	466
3C638	xPIXOFF	466
260DF	PIXOFF	91
260D5	PIXOFF3	91
3C60E	xPIXON	466
260E4	PIXON	91
260DA	PIXON3	91
260EE	PIXON?	91
260E9	PIXON?3	92
255BF	PixonB	94
255C4	PixonG1	94
255C9	PixonG2	94
255BA	PixonW	94
255CE	PixonXor	94
3EE9D	xPKT	466
3FA006	^PLCZ	
36FE2	plDRPpZparg	88

Addr.	Name	Page
420006	^TAN2EXP	351
421006	^tan2exp	366
01F314	~xTAN2SC	473
427006	^TAN2SC	351
021314	~xTAN2SC2	473
440006	^TAN2SC2	352
43F006	^TAN2SC2ext	357
426006	^TAN2SCext	356
40E006	^TAN2TAN/2	351
40F006	^tan2tan/2	366
52E006	^xTANext	349
3A70C	xTANH	473
44D006	^tanh2exp	367
44C006	^TANH2EXPext	352
531006	^xTANHext	350
1E1006	^TAYLOR0	412
006314	~xTAYLOR0	473
3E6CA	xTAYLR	473
00116	TBR	
302006	^TCHEBext	378
233006	^TCHEBNOCK	415
05B314	~xTCHEBYCHEFF	473
01A314	~xTCOLLECT	473
2FE006	^TCOLLECT	355
00112	TCS	
0640AB	xTDELTA	473
2F09A	TempConv	82
3316B	TEN	10
2E1006	^TESTINFINI	419
02E0DE	xTESTS	
26170	TestSysFlag	175
26175	TestUserFlag	175
065314	~xTEVAL	473
31F006	^TEXPAext	356
013314	~xTEXPAND	473
3C8FA	xTEXT	473
37F7F	xTHEN	473
38B43	xTHENCASE	
33189	THIRTEEN	11
33233	THIRTY	12
33283	THIRTYEIGHT	12
33265	THIRTYFIVE	12
3325B	THIRTYFOUR	12
3328D	THIRTYNINE	12
3323D	THIRTYONE	12
33279	THIRTYSEVEN	12
33251	THIRTYTHREE	12
33247	THIRTYTWO	12
2F158	THISCHAR	300
33125	THREE	10
33319	THREEFIVE	13
36216	THREE{}N	72
09D006	^THREE{}POLY	377
39093	xTICKS	473
2F003	Ticks>Date	173
2F004	Ticks>Rpt	173
2F002	Ticks>TOD	173
34BEF	ticR	128
3905D	xTIME	473
26161	TIMEOUT?	
0012E	TIMERCTRL.1	
0012F	TIMERCTRL.2	
2EED3	TIMESTR	49, 173
0650AB	xTINC	473
019314	~xTLIN	473
3C6B6	xTLINE	473
3E97B	xTMENU	473
26166	TOADISP	272
2EECF	TOD	172
2F381	TOD>t$	173
2616B	TOGDISP	272
2F382	TOGGLELINE#3	92
2577F	TogInsert	302
25F2D	TogInsertKey	
2EFA1	TOGLINE	92
2EFA4	TOGLINE3	92
2F21B	ToGray	95
33AA7	tok$	45
33AB3	tok&	45
33B85	tok'	44
33BD9	tok*	45
33BF1	tok+	45
33B91	tok,	44
33BFD	tok-	44
272D9	tok->	44
33B9D	tok.	44
33BE5	tok/	45
33C4D	tok0	44
33C59	tok1	44
33C65	tok2	45
33C71	tok3	45
33C7D	tok4	45
33C89	tok5	45
33C95	tok6	45
33CA1	tok7	45
33CAD	tok8	44

Addr.	Name	Page
331ED	TWENTYTHREE	11
331E3	TWENTYTWO	11
3311B	TWO	10
09E006	^TWO::POLY	377
36202	TWO{}N	72
09C006	^TWO{}POLY	377
3BC39	xTYPE	474
03C64	TYPE	199
350D2	TYPEAPLET?	200
3513B	TYPEARRY?	199
350F0	TYPEBINT?	200
352AD	TYPECARRY?	199
3503C	TYPECHAR?	200
03F95	TYPECMP	20
3512C	TYPECMP?	199
35177	TYPECOL?	200
3510E	TYPECSTR?	199
03FDB	TYPEEREL	20
03FE5	TYPEEXT	20
351B3	TYPEEXT?	200
35087	TYPEFLASHPTR?	200
350C3	TYPEFONT?	200
114007	^TYPEGAUSSINT?	200, 333
35186	TYPEGROB?	199
350FF	TYPEHSTR?	199
03FA9	TYPEIDNT	20
3504B	TYPEIDNT?	199
194006	^TYPEIDNTLAM?	199
167006	^TYPEIRRQ?	326, 421
03FD1	TYPELAM	20
350E1	TYPELAM?	199
03F9F	TYPELIST	20
35195	TYPELIST?	199
350B4	TYPELNGCMP?	200
350A5	TYPELNGREAL?	200
35292	TYPERARRY?	199
03F8B	TYPEREAL	20
3511D	TYPEREAL?	199
196006	^TYPEREALZINT?	201
3514A	TYPEROMP?	200
20D6F	TYPERRP	20
35159	TYPERRP?	200
03FBD	TYPESYMB	20
35168	TYPESYMB?	199
351A4	TYPETAGGED?	200
182006	^TYPEZ?	200
35096	TYPEZINT?	200
2F07B	U>nbr	82
2F099	U>NCQ	82
25F06	UART?	
2F387	UARTBUFLEN	
25F07	UARTxcp	
38FD7	xUBASE	474
3900B	xUFACT	474
0B7006	^UFactor	375
0B8006	^UFactor1	375
0BD006	^UFactorDeg2	376
0140DD	xUFL1→MINIF	474
2F07C	UM#?	83
2F07D	UM%	83
2F07E	UM%CH	83
2F07F	UM%T	83
2F080	UM*	83
2D74F	um*	81
2F081	UM+	83
2F082	UM-	83
2D759	um/	81
2F083	UM/	83
2F085	UM<=?	83
2F086	UM<?	83
2F087	UM=?	83
2F088	UM>=?	83
2F089	UM>?	83
2F07A	UM>U	82
2D763	um^	81
2F08A	UMABS	83
2D999	UMCEIL	83
2F08B	UMCHS	83
2F08C	UMCONV	82
2F08D	UMCOS	83
2D777	umEND	81
2D985	UMFLOOR	83
2D971	UMFP	83
2D95D	UMIP	83
2F08E	UMMAX	83
2F08F	UMMIN	83
2D76D	umP	81
2D9CB	UMRND	83
2F090	UMSI	82
2D949	UMSIGN	83
2F091	UMSIN	83
2F092	UMSQ	83
2F093	UMSQRT	83
2F094	UMTAN	83
2D9EE	UMTRC	83
2F095	UMU>	82

Addr.	Name	Page	Addr.	Name	Page
346B1	17PUTLAM	118	2D7006	^2DROPTRUE	136
346C0	18GETLAM	118	392006	^2DROPZ0	327
346BB	18PUTLAM	118	031AC	2DUP	106
346CA	19GETLAM	118	3674D	2DUP#+	24
346C5	19PUTLAM	118	358C2	2DUP#<	25
362A2	1_#1-SUB	48	358DC	2DUP#=	25
362A2	1_#1-SUB$	48	358F8	2DUP#>	25
25E6C	1A/LockA		36CA4	2DUP5ROLL	106
35DEE	1ABNDSWAP	119	36621	2DUPEQ	137
364FF	1GETABND	119	35D30	2DUPSWAP	106, 106
34616	1GETLAM	117	3370B	2EXT	16
2F318	1GETLAMSWP1+	119	3632E	2GETEVAL	119
35F42	1GETSWAP	119	34620	2GETLAM	117
36518	1LAMBIND	116	336E3	2GROB	16
3A4006	^1metainf#	367	336CF	2HXS	16
3A1006	^1metaundef#	367	2F17E	2HXSLIST?	22
2B3A6	1NULLLAM{}	119	155006	^2LAMBIND	116
34611	1PUTLAM	118	33459	2LIST	14
2F180	1REV	30	3A3006	^2metainf#	368
25E6D	1stkdecomp$w	51	3A0006	^2metaundef#	367
36676	2#0=OR	26	2825E	2NELCOMPDROP	68
30E5B	2%%>%	31	271F4	2NULLLAM{}	119
30E47	2%>%%	31	37087	2Ob>Seco	73
34B4F	2'RCOLARPITE	138	37046	2OVER	109
441006	^2*1-cos/sin	366	4E1006	^2POLYNOMIAL?	397
443006	^2*sin/1+cos	367	3461B	2PUTLAM	118
346D4	20GETLAM	118	343E1	2RDROP	129
346CF	20PUTLAM	118	331B1	2REAL	11
346DE	21GETLAM	118	3570C	2skipcola	131
346D9	21PUTLAM	119	35018	2SWAP	107
346E8	22GETLAM	118	12A006	^2SYMBINCOMP	72, 325
346E3	22PUTLAM	119	40D006	^2x/1+x^2	365
346F2	23GETLAM	118	34BBB	3@REVAL	128
346ED	23PUTLAM	119	341D2	3DROP	106
346FC	24GETLAM	118	2D5006	^3DUP	106
346F7	24PUTLAM	119	3462A	3GETLAM	117
34706	25GETLAM	118	156006	^3LAMBIND	116
34701	25PUTLAM	119	27208	3NULLLAM{}	119
34710	26GETLAM	118	34485	3PICK	109
3470B	26PUTLAM	119	36789	3PICK#+	25
3471A	27GETLAM	118	36CCC	3PICK3PICK	109
34715	27PUTLAM	119	360F7	3PICKOVER	109
34BAB	2@REVAL	128	35F1A	3PICKSWAP	109
173006	^2DMATRIX?	339	34625	3PUTLAM	118
03258	2DROP	106	343F3	3RDROP	129
355A5	2DROP00	21	33765	3REAL	17
25E6E	2DropBadKey	207	35703	3skipcola	131
35B32	2DROPFALSE	136	3422B	3UNROLL	108, 108

Appendix G
Entries sorted by Address

The entries in this index are sorted by address. Six-digit addresses are always sorted after five-digit addresses. The six-digit addresses for rompointers and flashpointers consist of the pointer number (first three digits) and the flashbank/library id (last three digits). Sorting of these addresses uses first the flashbank/library id and then the pointer number, so 000123 will be sorted after FFF122. Note that for technical reasons, the page number given may be off by one for a few percent of the entries. If the page reference is 167, the entry may actually be the first entry on page 168.

Addr.	Name	Page
03B46	AND	136
03B75	OR	136
03B97	EQUAL	137
03C64	TYPE	199
03CA6	#0=	25
03CC7	#0<>	25
03CE4	#<	25
03D19	#=	25
03D4E	#<>	25
03D83	#>	25
03DBC	#+	23
03DC7	#PUSHA-	
03DE0	#-	23
03DEF	#1+	23
03E0E	#1-	23
03E2D	#2+	23
03E4E	#2-	23
03E6F	#2*	23
03E8E	#2/	23
03EB1	#AND	25
03EC2	#*	23
03EF7	#/	23
03F8B	TYPEREAL	20
03F95	TYPECMP	20
03F9F	TYPELIST	20
03FA9	TYPEIDNT	20
03FA9	#2E48	20
03FBD	TYPESYMB	20
03FD1	TYPELAM	20
03FDB	TYPEEREL	20
03FE5	TYPEEXT	20
03FEF	#2614	20
03FF9	#2686	20
041A7	TurnOff	178
041ED	DEEPSLEEP	178
0426A	ShowInvRomp	178
04708	CHECKKEY	205
04714	GETTOUCH	206
047C7	REPKEY?	206
047CF	adrDISABLE_K	
047DD	adrKEYBUFFER	
04A0B	GETPROC	288
04A41	GETDF	288
04CE6	ERROR@	156
04D0E	ERRORSTO	156
04D33	ERRORCLR	156
04D3E	DROPNULL$	46
04D64	GETTHEMESG	157
04D87	JstGetTHEMESG	157
04D87	JstGETTHEMSG	157
04E07	GETEXITMSG	156
04E37	EXITMSGSTO	156
04E5E	ERRSET	157
04E66	addrTEMPENV	
04EA4	ABORT	156
04EB8	ERRTRAP	157
04ED1	ERRJMP	156
04FB6	SETMEMERR	157
04FF2	SETPORTNOTAV	157
05016	SETROMPERR	157
05040	ATTNFLG@	207
05068	ATTNFLGCLR	207
05089	CARCOMP	68
050ED	CAR$	47
05149	Loop	
05153	CDRCOMP	68
0516C	CDR$	47
0518A	&HXS	57
05193	&$	49
0521F	&COMP	68
0525B	>H$	49
052C6	>HCOMP	68
052EE	>T$	49
052FA	>TCOMP	68
05445	::N	71
05459	{}N	71
0546D	SYMBN	71, 86
05481	EXTN	81, 71
054AF	INNERCOMP	71
0556F	NULL$?	55
055B7	NULLCOMP?	68
055D5	NULLHXS	57
055DF	NULL$	43
055E9	NULL{}	72
055FD	NULL::	73
05616	LENHXS	57
05622	OVERLEN$	47
05636	LEN$	47
0567B	LENCOMP	68
056B6	NTHELCOMP	68
05733	SUB$	48
05815	SUBHXS	57
05821	SUBCOMP	69
05902	OSIZE	178
05944	OCRC	179
059CC	#>HXS	56

Addr.	Name	Page	Addr.	Name	Page
2557E	Sub	93	25790	INSERT?	302
25583	Repl	93	25795	INSERT_MODE	302
25588	Gor	93	2579A	REPLACE_MODE	
2558D	Gxor	94	257A2	EditLExists?	300
25592	SubRepl	93	257BE	ClrNewEditL	
25597	SubGor	93	257E2	NoIgnoreAlm	
2559C	SubGxor	93	2580E	SetRebuild	288
255A1	Grey?	94	25845	MenuDef@	289
255A6	ZoomX		25863	MenuRowAct!	289
255AB	ZoomY		25877	LabelDef!	290
255B0	ScrollVGrob	94	25886	DoLabel	293, 95
255B5	Distance	95	2588B	MenuKeyNS!	290
255BA	PixonW	94	25890	MenuKeyNS@	290
255BF	PixonB	94	2589A	DoMenuKeyNS	293
255C4	PixonG1	94	2589F	MenuKeyLS!	290
255C9	PixonG2	94	258B3	MenuKeyRS!	290
255CE	PixonXor	94	258C7	ReviewKey!	291
255D3	FBoxW	94	258EF	ExitAction!	291
255D3	FBoxG1	94	25908	LastMenuDef!	289
255D8	FBoxG2	94	2590D	LastMenuDef@	289
255DD	FBoxB	94	2593F	KeyOb0	
255E2	FBoxXor	94	25949	KeyOb!	
255E7	LBoxW	94	2594E	KeyOb@	
255EC	LBoxG1	94	25967	GetUserKeys	208
255F1	LBoxG2	94	2597B	CtlAlarm!	
255F6	LBoxB	94	25980	CtlAlarm@	
255FB	LBoxXor	94	25E67	!*triand	50
25617	SetNUsrKeyOK	208	25E68	!*trior	50
2561C	ClrNUsrKeyOK	208	25E69	%+SWAP	31
25621	NonUsrKeyOK?	208	25E6A	'DoBadKey	130
25636	HISTON?		25E6B	'DoBadKeyT	130
2564D	SetNAppKeyOK	224	25E6C	1A/LockA	
2565A	DoStdKeys?	224	25E6D	1stkdecomp$w	51
2565F	SetDoStdKeys	224	25E6E	2DropBadKey	207
25671	SetAppSuspOK	224	25E6F	>Review$	52
25676	ClrAppSuspOK	224	25E70	?ATTNQUIT	207
25683	SetBadPOLUI		25E70	?ATTN_QUIT	207
25690	AppMode?	224	25E71	?BlinkCursor	179
25695	SetAppMode	224	25E72	?CaseKeyDef	142
2569A	ClrAppMode	224	25E73	?CaseRomptr@	142
256A2	UNDO_ON?		25E74	?ClrAlg	278
256A7	UNDO_ON		25E75	?ClrAlgSetPr	278
256AC	UNDO_OFF		25E76	?Key>UKeyOb	
256BE	NOBLINK	179	25E77	?OKINALG	201
256EA	AlgEntry?	278	25E78	?PURGE_HERE	167
25719	SetAlgEntry	278	25E79	XEQSTOID	167
2571E	ClrAlgEntry	278	25E79	?STO_HERE	167
2577F	TogInsert	302	25E7A	ALARMxcp	

Addr.	Name	Page	Addr.	Name	Page
25EDD	H/WKey>KeyOb		25F0E	XYGROBDISP	91
25EDE	HARDHEIGHT	273	25F0F	a%>$	46
25EDF	ImmedEntry?	278	25F0F	a%>$,	46
25EE0	InitMenu	291	25F10	ederr	158
25EE1	InitMenu%	291	25F11	editdecomp$w	53
25EE2	InitTrack:	289	25F12	sstDISP	280
25EE3	KEYINBUFFER?	206	25F13	stkdecomp$w	51
25EE4	KeepUnit	82	25F14	XEQPGDIR	168
25EE5	Key>StdKeyOb	208	25F15	>FONT	283
25EE6	Key>U/SKeyOb	208	25F16	DISP_LINE	
25EE7	LastNonNull	168	25F17	GetMetaVStackDROP	161
25EE8	LoadTouchTbl	292	25F18	GetVStack	161
25EE9	LockAlpha	278	25F19	PopMetaVStack	162
25EEA	ModifierKey?	205	25F1A	PopMetaVStackDROP	160
25EEB	NEXTLIBBAK	99	25F1B	PopVStack	160
25EEC	NULL$TEMP	46	25F1C	PopVStackAbove	162
25EED	NoAttn?Semi	207	25F1D	PushMetaVStack&Drop	161
25EEE	NoEdit?case	146	25F1E	PushVStack	160
25EEF	NoExitAction	291	25F1F	PushVStack&Clear	160
25EF0	OR$	50	25F20	PushVStack&Keep	162
25EF1	PATHDIR	169	25F21	PushVStack&KeepDROP	162
25EF2	PrevNonNull	168	25F22	RestoreSysFlags	176
25EF3	RAD?	178	25F23	SaveSysFlags	176
25EF4	RECLAIMDISP	273	25F24	RIGHT$3x6	96
25EF5	REPEATER		25F25	CKNNOLASTWD	197
25EF6	REPEATERCH		25F29	EvalNoCK:	198
25EF7	SAFE@_HERE	166	25F2A	Keyword?	
25EF8	SEP$NL	48	25F2B	CHECKMENU	293
25EFA	SLEEPxcp		25F2C	Find1stT.1	
25EFB	SaveLastMenu	289	25F2D	TogInsertKey	
25EFC	SetKeysNS	290	25F63	!REDIMUSER	
25EFD	SetSomeRow	293	25F68	!REDIMTEMP	
25EFE	SetThisRow	292	25F6D	realPAcode	
25EFF	SolvMenuInit	293	25F72	#:>$	46
25F00	StartMenu	292	25F77	#>$	46
25F01	Std/BoxLabel	95	25F7C	$>GROB	96
25F02	StdMenuKeyLS	290	25F81	$>grob	96
25F03	StdMenuKeyNS	290	25F86	$>GROBCR	96
25F04	SuspendOK?	224	25F8B	$>grobCR	96
25F05	TurnOffKey		25F90	>LANGUAGE	179
25F06	UART?		25F95	LANGUAGE>	179
25F07	UARTxcp		25F9A	0LastRomWrd!	197
25F08	UnLockAlpha	278	25F9A	0LASTOWDOB!	197
25F09	UserKeys?	208	25F9F	?ACCPTR>	
25F0A	VLM		25FA4	ABUFF	272
25F0B	WaitForKey	206	25FA9	ALARM?	174
25F0C	XEQStoKey	167	25FAE	ATTN?	207
25F0D	XOR$	50	25FB3	DISPN	281

Addr.	Name	Page
26175	TestUserFlag	175
2617F	WINDOWCORNER	279
26184	WINDOWDOWN	279
26189	WINDOWLEFT	279
2618E	WINDOWRIGHT	279
26193	WINDOWUP	279
26198	WINDOWXY	279
2619D	addtics	
261A2	rstfmt1	178
261A7	savefmt1	177
261AC	setbeep	178
261B1	stackitw	
261B6	subpdcdptch	
261BB	tok8trior	50
261C0	CLCD10	277
261C5	CLEARLCD	277
261CA	FLUSHKEYS	205
261CA	FLUSH	205
261CF	%%>C%	37
261D4	C%%0=	38
261D9	C%%>C%	37
261DE	C%%CHS	38
261E3	C%%CONJ	38
261E8	C%0=	38
261ED	C%CHS	37
261F2	C%CONJ	37
261F7	DISPROW6	281
261FC	Re>C%	37
26201	nextsym'R	
26206	tok8cktrior	50
2620B	>MINIFONT	283
26210	CHECK_SCAN_FONT	
26215	DropVStack	162
2621A	FONT>	283
2621F	FSCANFONT	
26224	GetElemBotVStack	163
26229	GetElemTopVStack	163
2622E	SetVStackProtectWord	164
26233	GetVStackProtectWord	164
26238	GetFontCmdHeight	
2623D	GetFontHeight	
26242	GetFontStkHeight	283
26242	StackFontHeight	283
26247	GetHeader	273
2624C	GetMetaVStack	162
26251	InitVirtualStack	
26256	INITMKFONT	
2625B	MINIFONT>	283
26260	nDISPSTACK	275
26265	PushMetaVStack	161
2626A	PutElemBotVStack	163
2626F	PutElemTopVStack	163
26274	RCL_NB_AFF_LGN	
26279	RCL_NB_AFF_LGNSTK	
2627E	SCANFONT	
26283	SetHeader	273
26288	StackLineHeight	283
2628D	STO_ML_DISP_SIZE	
26292	CK0NOLASTWD	196
26297	CK1NOLASTWD	197
2629C	CK2NOLASTWD	197
262A1	CK3NOLASTWD	197
262A6	CK4NOLASTWD	197
262AB	CK5NOLASTWD	197
262B0	CK0	196
262B5	CK1	196
262BA	CK2	196
262BF	CK3	196
262C4	CK4	196
262C9	CK5	196
262CE	CKN	196
262D8	SETSIZEERR	158
262DD	SETTYPEERR	157
262E2	SETSTACKERR	157
262E7	SETNONEXTERR	158
262EC	%ABSCOERCE	22
262F1	COERCE	22
262F6	UNCOERCE	31
262FB	COMPEVAL	128
26300	CK1&Dispatch	197
26305	CK2&Dispatch	197
2630A	CK3&Dispatch	197
2630F	CK4&Dispatch	197
26314	CK5&Dispatch	197
26319	EvalNoCK	198
2631E	CK&DISPATCH0	197
26323	CK&DISPATCH2	197
26328	CK&DISPATCH1	197
2632D	#*OVF	23
2639B	MATATLOOP	
263D2	!MATTRNnc	
26459	setStdWid	51
2645E	setStdEditWid	51
2649F	ClrBusyAnn	278
264CC	FIRSTC@	300
264D1	SETFIRSTC_0	

Addr.	Name	Page
27EAF	<SkipKey	314
27EFB	>SkipKey	314
27F47	<DelKey	314
27F9A	>DelKey	314
27FED	NullMenuKey	291
28001	ROT#1+UNROT	24
28071	pull	24, 77
28071	#1-SWAP	24, 77
28085	pullrev	77
28099	SWAP#1+SWAP	24
280AD	SWAP#1-SWAP	24
280C1	ORDERXY#	92
280F8	ORDERXY%	92
2812F	SWAPTRUE	136
28143	NDUPN	106
28187	reversym	107
281D5	#1-UNROT	24
28211	NDROPFALSE	75, 136
28225	9UNROLL	109
2825E	2NELCOMPDROP	68
282CC	DROP%0	31
283E8	FalseFalse	136
283FC	ISTOP-INDEX	152
286E7	symcomp	86
28989	SWAPDROP#1-	24
28ACE	DROP?symcomp	86
293A3	?symcomp	86
293F8	P{}N	71
2962A	RDROPFALSE	136
2963E	psh&	76
29693	psh1top&	77
296A7	top&	76
2973B	pshtop&	76
29754	pullpsh1&	77
29786	'RSWAP#1+	24
29821	psh1&	77
298C0	psh1&rev	77
29972	pshzer	77
29986	pshzerpsharg	88
29A5D	psh	76
29A8F	roll2ND	76
29B12	unroll2ND	76
29CB9	uncrunch	87
29D18	ONE{}N	72
29D6A	delimcase	
29E29	ngsizecase	
29E67	nultrior	50
29E99	tok=casedrop	145
29ED0	'Rapndit	131
29EE9	DaDGNTc	
29F25	AppDisplay!	223
29F35	AppDisplay@	223
29F55	AppKeys!	223
29F75	AppKeys0	223
2A055	AppExitCond!	223
2A065	AppExitCond@	223
2A085	Clipboard!	307
2A095	Clipboard@	307
2A0A5	Clipboard0	307
2A0B5	Clipboard?	307
2A145	AppError!	223
2A158	AppError@	223
2A4FC	WaitTbz0	
2A5CA	SUB$1#	48
2A7A7	CkSecoType	
2A7F7	DispTimeReq?	274
2A842	Decomp1Line	52
2A85D	DecompEdit	53
2A878	Decomp#Line	52
2A893	Decomp#Disp	52
2A8AE	DecompEcho	54
2A8C9	DecompStd1Line	52
2A8E4	DecompStd1Line32	52
2A904	RPNDecomp1Line	52
2A924	RPNDecompEdit	53
2A944	RPNDecomp#Line	53
2A964	RPNDecomp#Disp	52
2A984	RPNDecompEcho	54
2A9A4	RPNDecompStd1Line	52
2A9C4	RPNDecompStd1Line32	52
2A9E9	RunRPN:	198
2AA43	AlgDecomp	53
2AA70	CASEVAL	
2AAE0	SimplifyExpression	
2AB69	RunInApprox	176
2ABD7	RunSafeFlagsNoError	177
2ABF0	RunSafeFlags	176
2AC0E	DoRunSafe	177
2AC72	need'case	
2ACA9	addrTEMPTOP	
2ACB0	?TogU/LCase	
2AD81	EQUALcasedrop	145
2ADBD	nonopcase	145
2ADE0	numb1stcase	145
2AE32	M-1stcasechs	143
2AF37	AEQ1stcase	143

Addr.	Name	Page
2C2AA	D/DSUM	
2C2B5	D/DAPPLY	
2C2C0	nCustomMenu	293
2C2CB	nCOLCTQUOTE	
2C2D6	SPLITWHERE	
2C2F9	DispStsBound	274
2C305	DispStatus	274
2C311	?DispStatus	274
2C341	?DispStack	274
2C371	DoInputForm	257
2C37D	PTYPE>PINFO	
2C388	MOVEVAR	
2C393	COPYVAR	
2C4D2	#AAA0	20
2D74F	um*	81
2D759	um/	81
2D763	um^	81
2D76D	umP	81
2D777	umEND	81
2D848	tok_g	44
2D86D	tok_m	44
2D8AD	tok_s	44
2D949	UMSIGN	83
2D95D	UMIP	83
2D971	UMFP	83
2D985	UMFLOOR	83
2D999	UMCEIL	83
2D9CB	UMRND	83
2D9EE	UMTRC	83
2DA11	cfF	30
2DA2B	cfC	30
2DCB5	FLOAT	
2DD27	Day>Date	
2DDD5	getBPOFF	
2DE26	mpop1%	
2DEBB	DAY#	
2DFCC	?DispMenu	292, 274
2DFE0	DispMenu	292, 274
2DFF4	DispMenu.1	292, 274
2E0D5	Grob>Menu	293, 95
2E0F3	Str>Menu	293, 95
2E107	Seco>Menu	293, 95
2E11B	Id>Menu	293, 95
2E166	MakeStdLabel	95
2E189	MakeBoxLabel	95
2E1EB	MakeDirLabel	95
2E24D	MakeInvLabel	95
2E25C	InvLabelGrob	90
2E2AA	MakeLabel	293, 96
2EE5A	DispEditLine	274
2EE5B	DispTime?	
2EE5C	BlankDA12	277
2EE5D	?FlashAlert	178
2EE5E	SysErrorTrap	
2EE5F	SysErrorTrapConfirm	
2EE60	DoWarning	282
2EE61	FlashWarning	282
2EE62	DA1OK?	275
2EE63	DA3OK?	275
2EE64	SetDAsTemp	275
2EE65	SetDA12a3NoCh	276
2EE65	SetDA12a3NCh	276
2EE66	DA2aLess1OK?	275
2EE67	SetDA1Valid	275
2EE68	SetDA2bValid	275
2EE69	SetDA1Temp	275
2EE6A	SetDA2bTemp	275
2EE6B	SetDA3Temp	275
2EE6C	SetDA2aEcho	276
2EE6D	ClrDAsOK	275
2EE6E	ClrDA3OK	275
2EE6F	SetDA12NoCh	276
2EE70	SetDA13NoCh	276
2EE71	SetDA12Temp	275
2EE72	SetDA1NoCh	276
2EE73	SetDA2aNoCh	276
2EE74	ClrDA1Bad	276
2EE75	ClrDA2aBad	276
2EE76	SetDA2bNoCh	276
2EE77	SetDA3NoCh	276
2EE78	SetDA1Bad	276
2EE79	SetDA2aBad	276
2EE7A	SetDA2bBad	276
2EE7B	SetDA3Bad	276
2EE7C	SetDAsNoCh	276
2EE7D	ClrDA1IsStat	273
2EE7E	DA2bIsEdL?	276
2EE7F	SetDA2bIsEdL	276
2EE80	ClrDA2bIsEdL	276
2EE81	ClrDA2bNoCh	276
2EE8A	SetDA2aTemp	275
2EE8B	MENoP&FixDA1	
2EE8D	ClrDA1OK	275
2EE8E	ClrDA2aOK	275
2EE8F	ClrDA2bOK	275
2EE90	ClrDA2OK	275

Addr.	Name	Page
2EF5A	apndvarlst	73
2EF5E	BlankDA1	277
2EF5F	InputLine	212
2EF60	DOGRAPHIC	318
2EF61	WINDOW#	281
2EF62	palparse	50
2EF66	SysMenuCheck	292
2EF67	SysDisplay	274
2EF68	ClrDouseAlm	
2EF69	EvalParsed	
2EF6A	Parse.1	
2EF6B	Parse.2	
2EF6C	AtUserStack	197
2EF6D	GetLastEdit	
2EF6E	ParseFail	51
2EF6F	DispBadToken	51
2EF70	ParseFail2	
2EF71	DispBadToken2	
2EF72	CacheStack	
2EF73	?Space/Go>	311
2EF74	CMD_PLUS	301
2EF75	AddTrailingSpace	312
2EF76	AddLeadingSpace	312
2EF77	CMD_PAGEL	305
2EF78	CMD_PAGER	305
2EF79	CMD_PAGEU	305
2EF7A	CMD_PAGED	305
2EF7B	CMD_BAK	305
2EF7C	CMD_NXT	305
2EF7D	CMD_DEB_LINE	305
2EF7E	CMD_END_LINE	305
2EF7F	CMD_UP	305
2EF80	CMD_DOWN	305
2EF81	CMD_DROP	302
2EF82	CMD_DEL	302
2EF83	CMD_STO_DEBUT	306
2EF84	CMD_STO_FIN	306
2EF85	RCL_CMD_DEB	306
2EF86	RCL_CMD_FIN	306
2EF87	RCL_CMD_POS	300
2EF88	CMD_CUT	306
2EF8A	CMD_COPY	306
2EF8B	STO_CURS_POS	304
2EF8C	STO_CURS_POS2	304
2EF8D	STO_CURS_POS3	304
2EF8E	STO_CURS_POS4	304
2EF8F	STO_CURS_POS_VIS	304
2EF90	CAL_CURS_POS_VIS	301
2EF91	CAL_CURS_POS	301
2EF92	XLINE_SIZE?	314
2EF93	VERIF_SELECTION	306
2EF94	PASTE.EXT	307
2EF95	DEL_CMD	303
2EF96	NO_AFFCMD	312
2EF97	InsertEcho	302
2EF98	SetDA2aValid	275
2EF99	SetDA3Valid	275
2EF9A	CommandLineHeight	312
2EF9F	LINEON	92
2EFA0	LINEOFF	92
2EFA1	TOGLINE	92
2EFA2	LINEON3	92
2EFA3	LINEOFF3	92
2EFA4	TOGLINE3	92
2EFA5	DOHEX	177
2EFA6	DOBIN	177
2EFA7	DOOCT	177
2EFA8	DODEC	177
2EFAA	dostws	57
2EFAC	bitAND	58
2EFAD	bitOR	58
2EFAE	bitXOR	58
2EFAF	bitNOT	58
2EFB0	bitSL	58
2EFB1	bitSLB	58
2EFB2	bitSR	58
2EFB3	bitSRB	58
2EFB4	bitRR	58
2EFB5	bitRRB	58
2EFB6	bitRL	58
2EFB7	bitRLB	58
2EFB8	bitASR	58
2EFB9	bit+	57
2EFBA	bit-	57
2EFBC	bit*	57
2EFBD	bit/	57
2EFBE	WORDSIZE	57
2EFBF	BASE	177
2EFC0	HXS>$	46
2EFC1	hxs>$	46
2EFC2	bit%#/	57
2EFC3	bit#%/	58
2EFC4	bit%#*	57
2EFC5	bit#%*	57
2EFC6	bit%#-	57
2EFC7	bit#%-	57

Addr.	Name	Page	Addr.	Name	Page
2F366	PUTXMIN	316	2F94C	%1	28
2F367	PUTYMAX	317	2F961	%2	28
2F368	PUTYMIN	317	2F976	%3	28
2F369	RECORDX&YC%		2F98B	%4	28
2F36A	REMAP		2F9A0	%5	28
2F36B	RIGHTCOL	279	2F9B5	%6	28
2F36C	Rows8-15	279	2F9CA	%7	28
2F36D	SCROLLDOWN	279	2F9DF	%8	28
2F36E	SCROLLLEFT	280	2F9F4	%9	28
2F36F	SCROLLRIGHT	280	2FA09	%-1	28
2F370	SCROLLUP	279	2FA1E	%-2	28
2F371	SENDACK		2FA33	%-3	28
2F372	SENDEOT		2FA48	%-4	28
2F373	SENDERROR		2FA5D	%-5	28
2F374	SENDNAK		2FA72	%-6	28
2F375	SENDNULLACK		2FA87	%-7	28
2F376	SENDPKT		2FA9C	%-8	28
2F377	SendSetup		2FAB1	%-9	28
2F378	SetCursor	304	2FAC6	%PI	28
2F379	SetDA123NoCh	276	2FADB	%%PI	30
2F37A	SetDA2OKTemp	275	2FAF5	%MAXREAL	30
2F37B	SetIOPARErr	158, 181	2FB0A	%-MAXREAL	28
2F37C	SETLOOPENV		2FB1F	%MINREAL	28
2F37D	SetServMode		2FB34	%-MINREAL	28
2F37E	SORTASLOW	172	2FB49	%%0	30
2F37F	STOALM	174	2FB63	%%1	30
2F380	SysSTO	167	2FB7D	%%2	30
2F381	TOD>t$	173	2FB97	%%3	30
2F382	TOGGLELINE#3	92	2FBB1	%%4	30
2F383	TOP16	279	2FBCB	%%5	30
2F384	TOP8	279	2FBE5	%%.1	30
2F385	TOPROW	279	2FBFF	%%.5	30
2F386	TRPACKETFAIL		2FC19	%%10	30
2F387	UARTBUFLEN		2FC7D	%1200	29
2F388	VerifyTOD	172	2FC92	%2400	29
2F389	VERSTRING	181	2FCA7	%4800	29
2F38A	WINDOWBOT?	280	2FCBC	%9600	29
2F38B	WINDOWLEFT?	280	2FCD1	%15360	29
2F38C	WINDOWRIGHT?	280	2FCE6	%11	28
2F38D	WINDOWTOP?	280	2FCFB	%12	28
2F38E	xnsgeneral	77	2FD10	%13	29
2F38F	xsngeneral	77	2FD25	%14	29
2F390	xssgeneral		2FD3A	%15	29
2F3A9	STOALLF	176	2FD4F	%16	29
2F3AA	STOALLF2	176	2FD64	%17	29
2F3B3	AsnKey	208	2FD79	%18	29
2F458	SETIVLERR	158	2FD8E	%19	29
2F937	%0	28	2FDA3	%20	29

Addr.	Name	Page
350CD	DUPTYPEAPLET?	200
350D2	TYPEAPLET?	200
350DC	DUPTYPELAM?	199
350E1	TYPELAM?	199
350EB	DUPTYPEBINT?	200
350F0	TYPEBINT?	200
350F5	#37258	21
350FA	DUPTYPEHSTR?	199
350FF	TYPEHSTR?	199
35109	DTYPECSTR?	199
35109	DUPTYPECSTR?	199
3510E	TYPECSTR?	199
35118	DTYPEREAL?	199
35118	DUPTYPEREAL?	199
3511D	TYPEREAL?	199
35127	DUPTYPECMP?	199
3512C	TYPECMP?	199
35136	DUPTYPEARRY?	199
35136	DTYPEARRY?	199
3513B	TYPEARRY?	199
35145	DUPTYPEROMP?	200
3514A	TYPEROMP?	200
35154	DUPTYPERRP?	200
35159	TYPERRP?	200
35163	DUPTYPESYMB?	199
35168	TYPESYMB?	199
35172	DTYPECOL?	200
35172	DUPTYPECOL?	200
35177	TYPECOL?	200
35181	DUPTYPEGROB?	199
35186	TYPEGROB?	199
35190	DUPTYPELIST?	199
35190	DTYPELIST?	199
35195	TYPELIST?	199
3519F	DUPTYPETAG?	200
351A4	TYPETAGGED?	200
351AE	DUPTYPEEXT?	200
351B3	TYPEEXT?	200
351FA	OverWrF/TLp	
35268	OVER#=	25
35280	DROPTRUE	136
35289	DROPFALSE	136
35292	TYPERARRY?	199
352AD	TYPECARRY?	199
352BD	DUP#0=	25
352E0	#3=	25
352F1	#2=	25
352FE	#1=	25
3530D	#1<>	25
3531C	DUP#1=	26
3532B	DUP#0<>	26
3533C	!insert$	49
35346	SWAP&$	49
35369	!!append$?	49
353CD	!append$	49
353EB	!!insert$	49
353F7	!!append$	49
354CB	'RSaveRomWrd	197
354CB	'RSAVEWORD	197
35511	#MIN	25
3551D	#MAX	25
35552	#-#2/	24
3558C	DROPZERO	21
355A5	2DROP00	21
355D0	#6-	23
355D5	#5-	23
355DA	#4-	23
355DF	#3-	23
355FD	#3+	23
35602	#4+	23
35607	#5+	23
3560C	#6+	23
35611	#7+	23
35616	#8+	23
3561B	#9+	23
35620	#10+	23
3562A	#12+	23
35675	#10*	23
3569B	#8*	23
356B8	#6*	23
356D5	5skipcola	131
35703	3skipcola	131
3570C	2skipcola	131
35715	skipcola	131
3571E	DUP#2+	24
35733	DROPSWAP	106
3574D	XYZ>Y	106, 107
3574D	ROT2DROP	106, 107
3574D	DROPSWAPDROP	106, 107
3576E	SWAPDUP	107
3579C	ROTDUP	107
357BB	SWAP#-	24
357CE	DROPDUP	106
357E2	DUPLEN$	47
357FC	#+DUP	24
3581F	#-DUP	24

Addr.	Name	Page
36057	4UNROLLROT	108
3606B	DROPOVER	106
3607F	EQOVER	137
36093	#+OVER	24
360A7	#-OVER	24
360BB	ZEROOVER	22
360CF	UNROTOVER	108
360E3	4ROLLOVER	108
360F7	3PICKOVER	109
3610B	4PICKOVER	109
3611F	DUPPICK	106
36133	DUPROLL	106
36147	OVER#2+UNROL	77
3615B	8UNROLL	109
3616F	10UNROLL	109
36183	OVERARSIZE	64
3619E	'ERRJMP	130
361B2	caseERRJMP	146
361C6	?CARCOMP	68
361DA	NEWLINE$&$	47
361DA	NEWLINE&$	47
361EE	#1-{}N	72
36202	TWO{}N	72
36216	THREE{}N	72
3622A	DUPINCOMP	71
3623E	SWAPINCOMP	71
36252	DUPNULL$?	55
36266	DUPNULLCOMP?	68
3627A	DUPLENCOMP	68
3628E	#1-SUB$	48
362A2	1_#1-SUB$	48
362A2	1_#1-SUB	48
362B6	LAST$	48
362CA	#1+LAST$	48
362DE	DUP$>ID	116
362F2	SWAP%>C%	37
36306	'NOP	130
3631A	::NEVAL	74
3632E	2GETEVAL	119
36342	DROPRDROP	129
3635B	SWAPCOLA	131
3636F	XYZ>ZCOLA	131
36383	#0=?SKIP	141
3639C	#1=?SKIP	142
363B5	#=?SKIP	140
363CE	ONE_EQ	25
363E2	#>?SKIP	140
363FB	COLASKIP	131
3640F	NOT_UNTIL	151
36428	NOT_WHILE	151
36441	DUP#0<>WHILE	151
3645A	DUPINDEX@	152
3646E	SWAPINDEX@	152
36482	OVERINDEX@	152
36496	SWAPLOOP	151
364AF	DROPLOOP	151
364C8	DUP#0_DO	151
364E1	toLEN_DO	151
364FF	1GETABND	119
36513	DUP1LAMBIND	116
36518	1LAMBIND	116
3652C	caseTRUE	140
36540	TrueFalse	136
36540	TRUEFALSE	136
36554	FalseTrue	136
36554	FALSETRUE	136
36568	ZEROFALSE	22
3657C	ONEFALSE	22
36590	#=casedrpfls	141
365B3	casedrpfls	139
365CC	case2drpfls	140
365E5	caseFALSE	140
365F9	ORNOT	136
3660D	EQUALNOT	137
36621	2DUPEQ	137
36635	DUPEQ:	137
3663A	EQ:	137
3664E	EQOR	137
36662	EQUALOR	137
36676	2#0=OR	26
36694	OVER#0=	25
366A8	OVER#<	25
366BC	#<3	25
366D0	DUP#<7	26
366E9	INNER#1=	71
366FD	#5=	25
36711	#2<>	25
36725	OVER#>	26
36739	ONE#>	25
36739	#>1	25
3674D	DUP3PICK#+	24
3674D	2DUP#+	24
36761	ROT#+	24
36775	OVER#+	24
36789	3PICK#+	25
3679D	4PICK#+	25

Addr.	Name	Page
36F8D	top&Cr	
36FA6	metaROTDUP	76
36FBA	ROTUntop&	76
36FCE	roll2top&	76
36FCE	rolltwotop&	76
36FE2	plDRPpZparg	88
36FF6	&$SWAP	49
3700A	SWAPCKREF	171
3701E	pZpargSWAPUn	88
37032	DROPNDROP	75, 106
37046	2OVER	109
3705A	?Ob>Seco	73
37073	Ob>Seco	73
37087	2Ob>Seco	73
3709B	ZEROISTOPSTO	152
3709B	ExitAtLOOP	152
370AF	RclHiddenVar	170
370C3	WithHidden	170
37104	StoHiddenVar	170
37118	PuHiddenVar	170
3712C	SaveVarRes	169
3714A	SetHiddenRes	170
37186	RestVarRes	169
371B3	Embedded?	69
371F9	UStackDepth	
3721C	Sig?ErrJmp	158
37258	DupAndThen	
37287	ZEROZERO	21
37294	#ZERO#ONE	21
37305	#ZERO#SEVEN	21
37315	#ONE#27	21
37328	#TWO#ONE	21
3733A	#TWO#TWO	21
3734A	#TWO#FOUR	21
3735C	#THREE#FOUR	21
3736E	#FIVE#FOUR	21
37380	ZEROZEROZERO	21
37394	ZEROZEROONE	21
373A8	ZEROZEROTWO	21
374BE	SWAPDROPTRUE	136
3760D	SubMetaOb	77
37685	SubMetaOb1	77
376B7	matchob?	69
376C1	matchob?Lp	
376EE	POSCOMP	69
37702	nextpos	
37752	#=POSCOMP	69
3776B	EQUALPOSCOMP	69
37784	NTHOF	69
37798	Find1stTrue	70
377C5	Lookup	70
377DE	Lookup.1	70
37829	EQLookup	70
378FA	POS$	47
378FA	POSCHR	47
37906	POSCHRREV	47
37906	POS$REV	47
37AA5	CHR>$	47
37ABE	STRIPTAGS	61
37AEB	STRIPTAGSl2	62
37B04	TAGOBS	61
37B54	NEXTCOMPOB	70
37C06	>LASTRAM-WORD	
37F48	xIF	462
37F7F	xTHEN	473
3805D	xELSE	458
3807D	xIFEND	
38093	xALG->	
380DB	xWHILE	475
38105	xREPEAT	469
3816B	xDO	457
38195	xUNTIL	474
381AB	xSTART	471
38252	xSTARTVAR	459
38266	#FFFF	21
38275	#BB	16
3831C	xNEXT	465
3851F	xSTEP	472
387AC	xIFERR	462
3880D	xHALT	461
38837	xSILENT'	
3885C	xRPN->	465
38999	x>>ABND	
389B9	x<<	
389D4	x>>	
389EF	x'	
38A14	xENDTIC	
38A2F	xWHILEEND	
38A54	xENDDO	458
38ABA	xERRTHEN	
38B28	xCASE	455
38B43	xTHENCASE	
38BAE	xDIR	
38BBF	xPROMPT	467
38C00	DISPST2&FREEZE	282
38C1B	xGROB	

Addr.	Name	Page	Addr.	Name	Page
3D1E7	xPRLCD	467	3DE90	xSUMX2	461
3D202	x∂	457	3DEAB	xSUMY2	461
3D258	xDER		3DEC6	xSUMXY	461
3D2B4	CKSYMBTYPE	199	3DEE1	xMAXSIGMA	464
3D393	xRCEQ	468	3DEFC	xMEAN	464
3D3AE	xSTEQ	472	3DF17	xMINSIGMA	464
3D3CE	xROOT	469	3DF32	xSDEV	470
3D434	x$\int$	462	3DF4D	xTOT	473
3D47E	xINTEGRAL		3DF68	xVAR	475
3D503	xSUM	460	3DF83	xLR	464
3D50D	#A110	20	3DFDD	xPREDV	467
3D51C	#AA10	20	3DFFD	xPREDY	467
3D52B	#A1A0	20	3E01D	xPREDX	467
3D56B	x|	460	3E03D	xXCOL	475
3D605	xWHERE		3E05D	xYCOL	476
3D6F6	xQUOTE	468	3E07D	xUTPC	474
3D7AC	xAPPLY	453	3E09D	xUTPN	474
3D81D	xFCNAPPLY		3E0BD	xUTPF	474
3DA3E	x->Q	468	3E0DD	xUTPT	474
3DA63	x->QPI	468	3E0FD	xSIGMACOL	455
3DAD0	xMATCHUP	464	3E127	xSCLSIGMA	470
3DB04	xMATCHDN	475	3E156	xSIGMALINE	460
3DB62	xFORMUNIT		3E171	xBINS	454
3DBCA	xPREDIV	468	3E17B	#111	17
3DBEA	xDUP	458	3E196	xBARPLOT	454
3DC05	xDUP2	458	3E1CA	xHISTPLOT	461
3DC20	xSWAP	473	3E1EF	xSCATRPLOT	470
3DC3B	xDROP	458	3E214	xLINFIT	463
3DC56	xDROP2	458	3E239	xLOGFIT	464
3DC71	xROT	469	3E25E	xEXPFIT	459
3DC8C	xOVER	466	3E283	xPWRFIT	
3DCA7	xDEPTH	457	3E2C1	xBESTFIT	454
3DCC7	xDROPN	458	3E331	xSINV	471
3DCE2	xDUPN	458	3E35B	xSNEG	471
3DCFD	xPICK	466	3E385	xSCONJ	470
3DD18	xROLL	469	3E3AF	xSTO+	472
3DD33	xROLLD	469	3E406	xSTO-	472
3DD4E	xCLEAR	455	3E46C	xSTO/	472
3DD6E	xSTOSIGMA	472	3E4D2	xSTO*	472
3DD8E	xCLSIGMA	455	3E54C	xINCR	462
3DDA9	xRCLSIGMA	468	3E576	xDECR	456
3DDC4	xSIGMA+	460	3E5A0	xCOLCT	455
3DDEE	xSIGMA-	460	3E5E9	xEXPAN	459
3DE09	xNSIGMA	465	3E632	xRULES	
3DE24	xCORR	456	3E648	xISOL	462
3DE3F	xCOV	456	3E66F	xQUAD	468
3DE5A	xSUMX	461	3E696	xSHOW	471
3DE75	xSUMY	461	3E6CA	xTAYLR	473

Addr.	Name	Page
088002	^SaveHARDBUFF	234
089002	^RestoreHARDBUFF	234
09D002	^DoCKeyOK	244
09E002	^DoCKeyCancel	244
09F002	^DoCKeyCheck	244
0A0002	^DoCKeyChAll	244
0AE002	^DoMKeyOK	
0AF002	^DoKeyCancel	
0B0002	^DoCKeyUnChAll	244
0B1002	^LEDispItem	245
0B2002	^LEDispList	245
0B3002	^LEDispPrompt	244
0B4002	^DoKeyOK	
0B5002	^DoKeyEdit	
0BB002	^GetFieldVals	
0BC002	^IFEDispField	
0BD002	^DOTVARS{}	169
0BE002	^ChangeFocus	
06C003	^laDELROW	
06D003	^laINSROW	
06E003	^laGPROW	
09A003	^WRAP$	48
0A4003	^BRdone	
0A5003	^BRDispItems	
0A6003	^BRinverse	
0A7003	^BRViewItem	
0AB003	^BRGetItem	
0AC003	^SWAPROWS	344
001004	^FSTR1	54
002004	^FSTR2	
003004	^FSTR3	54
004004	^FSTR4	54
005004	^FSTR5	54
006004	^FSTR6	54
007004	^FSTR7	55
008004	^FSTR8	
009004	^FSTR9	55
00A004	^FSTR10	
00B004	^FSTR11	
00C004	^FSTR12	
00D004	^FSTR13	55
00E004	^algparse	
00F004	^algunwrap	
010004	^EQW3	
011004	^EQW3Edit	311
012004	^EQW3StartEdit	
013004	^EQW3ViewMargin	
014004	^EQW3ViewLeftX	
015004	^EQW3ViewRightX	
016004	^EQW3ViewLeft	
017004	^EQW3ViewRight	
018004	^EQW3ViewRightRPL	
019004	^EQW3GROB	98
01A004	^EQW3GROBStk	98
01B004	^EQW3CursorOn	
01C004	^EQW3CursorOff	
01D004	^EQW3Code	
01E004	^EQW3GROBsys	98
01F004	^EQW3GROBmini	98
020004	^IfMain	257
021004	^IfSetFieldVisible	258
022004	^IfSetSelected	258
023004	^IfSetGrob	258
024004	^IfSetFieldValue	258
025004	^IfSetCurrentFieldValue	258
026004	^IfGetFieldValue	258
027004	^IfGetCurrentFieldValue	258
028004	^IfGetFieldMessageHan..	258
029004	^IfGetFieldType	258
02A004	^IfGetFieldObjectsType	258
02B004	^IfGetFieldDecompObject	258
02C004	^IfGetFieldChooseData	258
02D004	^IfGetFieldChooseDecomp	259
02E004	^IfGetFieldResetValue	259
02F004	^IfSetFieldResetValue	259
030004	^IfGetFieldInternalVa..	259
031004	^IfDisplayFromData	259
032004	^IfGetNbFields	259
033004	^IfCheckSetValue	259
034004	^IfCheckFieldtype	259
035004	^IfReset	259
036004	^IfSetField	259
037004	^IfKeyChoose	259
038004	^IfKeyEdit	260
039004	^IfKeyTypes	260
03A004	^IfKeyCalc	260
03B004	^IfKeyInvertCheck	260
03C004	^IfONKeyPress	260
03D004	^IfEnterKeyPress	260
03F004	^IfSetHelpString	260
040004	^IfSetTitle	260
041004	^IfSetTitle2	
042004	^IfMain2	261
043004	^IfPutFieldsOnStack	261
044004	^IfSetFieldPos	261
045004	^IfGetFieldPos	261

Addr.	Name	Page
19E006	^CLEANIDLAM	326
19F006	^ssSYMDER	
1A0006	^SYMDER	
1A1006	^DERIVext	387
1A2006	^siSYMDER	
1A3006	^DERIVIDNT	388
1A4006	^DERIVIDNT1	388
1A5006	^DERIV	388
1A6006	^METADERIV	388
1A7006	^DO>STRID	54
1A8006	^METADEROP	
1A9006	^METADER+	388
1AA006	^METADER-	388
1AB006	^METADER*	388
1AC006	^METADER/	388
1AD006	^METADER^	388
1AE006	^METADERFCN	388
1AF006	^METADERDER	388
1B0006	^METADERI4	388
1B1006	^METADERI3	388
1B2006	^METADERIFTE	388
1B3006	^DERARG	390
1B4006	^METADEREXP	388
1B5006	^METADERLN	388
1B6006	^METADERLNP1	388
1B7006	^METADERLOG	389
1B8006	^METADERALOG	389
1B9006	^METADERABS	389
1BA006	^METADERINV	389
1BB006	^METADERNEG	389
1BC006	^METADERSQRT	389
1BD006	^METADER&NEG	388
1BE006	^METADERSQ	389
1BF006	^METADERSIN	389
1C0006	^METADERCOS	389
1C1006	^METADERTAN	389
1C2006	^METADERSINH	389
1C3006	^METADERCOSH	389
1C4006	^METADERTANH	389
1C5006	^METADERASIN	389
1C6006	^METADERACOS	389
1C7006	^METADERATAN	389
1C8006	^METADERASH	389
1C9006	^METADERACH	389
1CA006	^METADERATH	390
1CB006	^pshder*	390
1CC006	^SQRTINVpshd*	390
1CD006	^ckaddt*	361
1CE006	^ckaddt+	360
1CF006	^ckaddt-	361
1D0006	^VERNUMext	408
1D1006	^MENUXYext	409
1D2006	^SAVECASFLAGS	408
1D3006	^SAFEPURGE	167
1D4006	^RESTORECASFLAGS	408
1D5006	^CASFLAGEVAL	408
1D6006	^FLAGEXPAND	411
1D7006	^EXPANDBOTH	
1D8006	^FLAGFACTOR	411
1D9006	^FLAGLISTEXEC	411
1DA006	^FLAGSYMBEXEC	411
1DB006	^FLAGIDNTEXEC	411
1DC006	^FLAGINTVX	411
1DD006	^DERVX	412
1DE006	^SOLVEXFLOAT	412
1DF006	^SYMLIMIT	412
1E0006	^FLAGMATRIXLIMIT	412
1E1006	^TAYLOR0	412
1E2006	^FLAGSERIES	412
1E3006	^PLOTSTK	
1E4006	^PLOTADD	412
1E5006	^FLAGIBP	412
1E6006	^FLAGPREVAL	412
1E7006	^MATRIXRISCH	412
1E8006	^FLAGRISCH	412
1E9006	^FLAGDERIV	412
1EA006	^FLAGLAP	412
1EB006	^FLAGILAP	412
1EC006	^FLAGDESOLVE	412
1ED006	^FLAGLDSSOLV	412
1EE006	^FLAGLDECSOLV	
1EF006	^FLAGTEXPAND	412
1F0006	^FLAGLIN	412
1F1006	^FLAGTSIMP	413
1F2006	^FLAGLNCOLLECT	413
1F3006	^FLAGEXPLN	413
1F4006	^FLAGSINCOS	413
1F5006	^FLAGTLIN	413
1F6006	^FLAGTCOLLECT	413
1F7006	^FLAGTRIG	413
1F8006	^FLAGTRIGCOS	413
1F9006	^FLAGTRIGSIN	413
1FA006	^FLAGTRIGTAN	413
1FB006	^FLAGTAN2SC	413
1FC006	^FLAGHALFTAN	413
1FD006	^FLAGTAN2SC2	413

Addr.	Name	Page	Addr.	Name	Page
25E006	^EvalNoCKxMOD	417	28E006	^RFACT2ext	383
25F006	^EvalNoCKxPERM	418	28F006	^RFACTSTEP3	383
260006	^EvalNoCKxCOMB	418	290006	^RFACTSTEP5	383
261006	^EvalNoCKxOR	418	291006	^METASOLV	383
262006	^EvalNoCKxAND	418	292006	^METASOLVOUT	
263006	^EvalNoCKxXOR	418	293006	^METASOLV2	383
264006	^EvalNoCKxXROOT	418	294006	^METASOLV4	383
265006	^TABVALext	418	295006	^ADDMULTIPL	384
266006	^TOLISText	418	296006	^FACTOOBJext	384
267006	^FROMLISText	418	297006	^SLVARext	353
268006	^PFEXECext	421	298006	^SIMPLIFY	354
269006	^LOP1ext	421	299006	^SIMP1ext	354
26A006	^LOPAext	421	29A006	^SYMEXPAN	354
26B006	^LISTSECOext	421	29B006	^SIMPVAR	354
26C006	^rpnQOBJext	423	29D006	^SIMPIDNT	423
26D006	^CK1TONOext	421	29E006	^RCLALLIDNT	
26E006	^COLCext	355	29F006	^RCL1IDNT	424
26F006	^SYMCOLCT		2A0006	^SIMPSYMBS	354
270006	^COLC1		2A1006	^SYMINTEGRAL	
271006	^COLC2		2A2006	^SIMPUSERFCN	354
272006	^MULMULText	381	2A3006	^EVALUSERFCN	354
273006	^METAMULMULT		2A4006	^SIMP|	354
274006	^METAMM2	381	2A5006	^DENOLCMext	358
275006	^COMPLISText	381	2A6006	^METADENOLCM	358
276006	^METACOMPRIM	381	2A7006	^SWPSIMPNDXF	424
277006	^METACOMP0		2A8006	^SIMPNDXFext	424
278006	^METACOMP1	381	2A9006	^SIMPext	354
279006	^ADDLISText	381	2AA006	^SIMPEXTOK	
27A006	^DIVISext	381	2AB006	^MAKEPROFOND	378
27B006	^FACT1ext	381	2AC006	^SLOWSIMP2L	
27C006	^FACTOext	381	2AD006	^SIMPGCDext	354
27D006	^ZFACTO	381	2AE006	^SIMP3ext	354
27E006	^SOLVext	382	2AF006	^SIMP3LISText	
27F006	^FRND	382	2B0006	^SIMP3LSTSLOW	
280006	^BICARREE?	382	2B1006	^LPGCDext	358
281006	^REALBICAR	382	2B2006	^SLOWGCDext	358
282006	^FEVIDENText	377	2B3006	^QGcd	358
283006	^EVIDENText	382	2B4006	^GCDext	
284006	^EVIDSOLV	382	2B5006	^CGCDext	333
285006	^DEG2ext	382	2B6006	^CMODext	424
286006	^METADEG2	382	2B7006	^ZGCDext	329
287006	^METADEG1	382	2B8006	^ZGcd	329
288006	^DEG1	382	2B9006	^TSIMP2ext	354
289006	^FDEG2ext	382	2BA006	^TSIMPext	354
28A006	^PIext	73	2BB006	^TSIMP3ext	355
28B006	^RACTOFACext	383	2BC006	^LASTCOMP	68
28C006	^FACTORACext	383	2BD006	^SQFF2ext	424
28D006	^RFACText	383	2BE006	^PPZ	424

Addr.	Name	Page	Addr.	Name	Page
2BF006	^PZHSTR	424	2EF006	^ipi	420
2C0006	^HORNER1ext	424	2F0006	^metaipi	420
2C1006	^PEval	424	2F1006	^meta-pi	420
2C2006	^RISCHext		2F2006	^metapi/2	420
2C3006	^risch/		2F3006	^metapi/4	420
2C4006	^rischABS		2F4006	^meta-pi/2	420
2C5006	^IBP	390	2F5006	^meta-pi/4	420
2C6006	^SQRT_IN?	424	2F6006	^pifois2	420
2C7006	^IS_SQRT?	424	2F7006	^deuxipi	420
2C8006	^XROOT_IN?		2F8006	^metapi*2	420
2C9006	^IS_XROOT?	424	2F9006	^base_ln	420
2CA006	^STOPRIMIT	424	2FA006	^meta_e	420
2CB006	^CONTAINS_LN?	424	2FB006	^NEXTPext	72
2CC006	^ISNT_IDNT?		2FC006	^INSERT{}N	72
2CD006	^RISCHPF		2FD006	^COMPRIMext	73
2CE006	^RISCHRAT		2FE006	^TCOLLECT	355
2CF006	^rischlogpart		2FF006	^SIGMAEXPext	355
2D0006	^PREVALext	390	300006	^LINEXPext	355
2D1006	^WARNSING	390	301006	^SIGMAEXP2ext	355
2D2006	^INText	390	302006	^TCHEBext	378
2D3006	^INT3	390	303006	^SINEXPA	355
2D4006	^FOURIERext	424	304006	^METASINEXPA	368
2D5006	^3DUP	106	305006	^SINEXPA+	368
2D6006	^#3+ROLL	108	306006	^SINEXPA-	368
2D7006	^2DROPTRUE	136	307006	^SINEXPA*	369
2D8006	^IRRQ#ULTIMATE	422	308006	^SINEXPA*1	369
2D9006	^LESSCOMPLEX?	424	309006	^COSEXPA	355
2DA006	^LISTIRRQ	422	30A006	^METACOSEXPA	369
2DB006	^LIST1i-1-i		30B006	^COSEXPA+	369
2DC006	^LIST10-10		30C006	^COSEXPA-	369
2DD006	^TABLECOSext	425	30D006	^COSEXPA*	369
2DE006	^TABLETANext	425	30E006	^COSEXPA*1	369
2DF006	^DROPZ1	327	30F006	^EXPEXPA	355
2E0006	^DROPZ0	327	310006	^METAEXPEXPA	369
2E1006	^TESTINFINI	419	311006	^EXPEXPA+	369
2E2006	^INFINIext	419	312006	^EXPEXPA-	369
2E3006	^MINUSINFext	419	313006	^EXPEXPA*	369
2E4006	^PLUSINFext	419	314006	^EXPEXPANEG	369
2E5006	^?ext	419	315006	^EXPEXPA*1	369
2E6006	^POSINFext	419	316006	^LNEXPA	355
2E7006	^POSUNDEFext	420	317006	^METALNEXPA	369
2E8006	^pisur2	420	318006	^LNEXPA*	369
2E9006	^pisur-2	420	319006	^LNEXPA/	369
2EA006	^pi	420	31A006	^LNEXPA^	369
2EB006	^metapi	420	31B006	^LINEXPA	355
2EC006	^'xPI	420	31C006	^MTRIG2SYMB	355
2ED006	^metai	420	31D006	^LNCOLCext	355
2EE006	^'xi	420	31E006	^METATANEXPA	369

Addr.	Name	Page	Addr.	Name	Page
3DF006	^RRDMext	378	40F006	^tan2tan/2	366
3E0006	^DEGREEext	378	410006	^addtTAN/2	366
3E1006	^FHORNER	378	411006	^TRIGTAN	356
3E2006	^HORNext	378	412006	^COS2TAN	351
3E3006	^HORN1		413006	^cos2tan	366
3E4006	^MHORNext	378	414006	^SIN2TAN	351
3E5006	^PTAYLext	372	415006	^sin2tan	366
3E6006	^LAGRANGEext	379	416006	^TRIGext	356
3E7006	^PSEUDOPREP	422	417006	^HYP2EXPext	356
3E8006	^PSEUDODIV	357	418006	^EXPLNext	356
3E9006	^IDIV2		419006	^SERIESEXPLN	356
3EA006	^BESTDIV2	357	41A006	^LNP12LN	351
3EB006	^CDIV2ext		41B006	^LOG2LN	351
3EC006	^QUOText	357	41C006	^ALOG2EXP	351
3ED006	^NEWDIVext	357	41D006	^EXPM2EXP	351
3EE006	^QDivRem	372	41E006	^SQRT2LNEXP	351
3EF006	^DIV2LISText	372	41F006	^sqrt2lnexp	351
3F0006	^DIVOBJext	420	420006	^TAN2EXP	351
3F1006	^DIVMETAOBJ	88, 361	421006	^tan2exp	366
3F2006	^LOPDext	420	422006	^ASIN2LN	351
3F3006	^QUOTOBJext	357	423006	^asin2ln	366
3F4006	^DIVISIBLE?	357	424006	^ACOS2LN	351
3F5006	^QDiv?	357	425006	^acos2ln	366
3F6006	^FastDiv?	357	426006	^TAN2SCext	356
3F7006	^POTENCEext	358	427006	^TAN2SC	351
3F8006	^PDIV2ext	372	428006	^sin/cos	366
3F9006	^PSetSign	372	429006	^SIN2TCext	356
3FA006	^PLCZ		42A006	^SIN2TC	351
3FB006	^HSECO2RCext	422	42B006	^cos*tan	366
3FC006	^SECO2CMPext	422	42C006	^COS2ext	351
3FD006	^SECO2CMPPOL		42D006	^sqrt1-sin^2	366
3FE006	^SECO2CMPCART		42E006	^SIN2ext	351
3FF006	^VALOBJext	423	42F006	^sqrt1-cos^2	366
400006	^R2SYM	325	430006	^ATAN2Sext	356
401006	^VAL2ext	423	431006	^ATAN2ASIN	352
402006	^INVAL2	423	432006	^atan2asin	366
403006	^METAVAL2	423	433006	^ASIN2Text	356
404006	^VAL1	423	434006	^ASIN2ATAN	352
405006	^VAL1M	423	435006	^asin2atan	366
406006	^addt0meta	77	436006	^ASIN2Cext	357
407006	^HALFTAN	356	437006	^ASIN2ACOS	352
408006	^COS2TAN/2	351	438006	^pi/2-acos	366
409006	^cos2tan/2	365	439006	^pi/2-meta	366
40A006	^1-x^2/1+x^2	365	43A006	^ACOS2Sext	357
40B006	^SIN2TAN/2	351	43B006	^pi/2-asin	366
40C006	^sin2tan/2	365	43C006	^ACOS2ASIN	352
40D006	^2x/1+x^2	365	43D006	^ATAN2LNext	352
40E006	^TAN2TAN/2	351	43E006	^atan2ln	366

Addr.	Name	Page	Addr.	Name	Page
43F006	^TAN2SC2ext	357	470006	^SYMPAPRX	
440006	^TAN2SC2	352	471006	^TRUNCDL	386
441006	^2*1-cos/sin	366	472006	^LIMSERIES!	386
442006	^TAN2CS2	352	473006	^LIMITX!	
443006	^2*sin/1+cos	367	474006	^LIMITNOVX!	
444006	^SIN2EXPext	352	475006	^LIMERR0!	
445006	^sin2exp	367	476006	^LIMERR1!	
446006	^COS2EXPext	352	477006	^LIMIT!	386
447006	^cos2exp	367	478006	^LIMSTEP1!	386
448006	^SINH2EXPext	352	479006	^LIMSTEP2!	
449006	^sinh2exp	367	47A006	^LIMSTEP3!	
44A006	^COSH2EXPext	352	47B006	^LIMSTEP4!	
44B006	^cosh2exp	367	47C006	^LIMLIM!	386
44C006	^TANH2EXPext	352	47D006	^n{}N	
44D006	^tanh2exp	367	47E006	^LIMLIM1!	
44E006	^ASINH2LNext	352	47F006	^LIMCMPL!	386
44F006	^asinh2ln	367	480006	^LIMEQUFR!	386
450006	^ACOSH2LNext	352	481006	^LIMEQU!	386
451006	^acosh2ln	367	482006	^LIMEQU0!	
452006	^ATANH2LNext	352	483006	^LIM+-!	387
453006	^atanh2ln	367	484006	^LIMERR10!	
454006	^XROOT2ext	352	485006	^LIMNEG!	
455006	^xroot2expln	367	486006	^LIMRAC!	
456006	^LN2ext	357	487006	^LIMINV!	
458006	^exp2sincos	367	488006	^LIM/!	
459006	^metai*	360	489006	^LIMPOW!	
45A006	^LN2ATAN	352	48A006	^LIMSQ!	
45B006	^VAR=LIST	353	48B006	^LIM*!	
45C006	^IDNTEXEC	423	48C006	^LIMDIVPC!	387
45D006	^SYMISOL		48D006	^DIVPC!	
45E006	^SYMQFORM		48E006	^LIMPROFEND!	387
45F006	^LISTEXEC	421	48F006	^LIMPROF!	
460006	^LISTEXEC1	421	490006	^LIM%#!	387
461006	^SECOEXEC	421	491006	^LIMPROF0!	
462006	^EQUATION?	86	492006	^LIMPROF1!	
463006	^USERFCN?	86	493006	^LIMPROF2!	
464006	^SYMBEXEC	353	494006	^LIMINVLN!	
465006	^MEVALext	353	495006	^LIMLN!	
466006	^CASNUMEVAL	353	496006	^LIMEXP!	
467006	^CASCOMPEVAL	353	497006	^LIMSINCOS!	
468006	^REPLACE2BY1	353	498006	^LIMATAN!	
469006	^NR_REPLACE	353	499006	^LIMASIN!	
46A006	^SYMBWHERE		49A006	^LIMSQRT!	
46B006	^CASCRUNCH	353	49B006	^LIMFLOOR!	
46C006	^APPROXCOMPEVAL	353	49C006	^LIMABS!	
46D006	^LIMIText		49D006	^LPROF!	
46E006	^REWRITEIFINF		49E006	^LIM#VARX!	387
46F006	^SYMTAYLOR	386	49F006	^LIMBETA!	

Addr.	Name	Page
4A0006	^LIMALPHA!	
4A1006	^HORNEXP!	387
4A2006	^HORNCOS!	
4A3006	^HORNSIN!	
4A4006	^LIMSC0!	
4A5006	^LIMSC1!	
4A6006	^HORNATAN!	
4A7006	^LIMATAS!	
4A8006	^HORNASIN!	
4A9006	^HORNASIN1!	
4AA006	^HORNLN!	
4AB006	^LNOBJ!	
4AC006	^NEWLIMHORN	
4AD006	^LIMHORN!	
4AE006	^LRDM!	
4AF006	^LIMDL!	
4B0006	^LIMDLINF!	
4B1006	^LIMINFSIGN!	
4B2006	^LIMMAX!	
4B3006	^LIMCOMP!	
4B4006	^VARCOMP2!	
4B5006	^LIMSORT!	
4B6006	^VARCOMP!	387
4B7006	^VARCOMPLN!	
4B8006	^VARCOMP3!	
4B9006	^VARCOMP31!	
4BA006	^VARCOMP32!	387
4BB006	^VARCOMP33!	
4BC006	^LIMERR6!	
4BD006	^LIMVALOBJ!	387
4BE006	^LIMVAL!	387
4BF006	^EQUIV!	387
4C0006	^LVARXNX2!	387
4C1006	^SIMP1!	
4C2006	^FindCurVar	387
4C3006	^LIMVAR!	387
4C4006	^VAR%	
4C5006	^ISOL1	396
4C6006	^ISOLALL	396
4C7006	^ISOL2ext	396
4C8006	^BEZOUTMSOLV	396
4C9006	^ROOT{}N	396
4CA006	^MHORNER	396
4CB006	^MHORNER1	396
4CC006	^SQFFext	396
4CD006	^MSQFF	396
4CE006	^%1TWO	396
4CF006	^MZSQFF	396
4D0006	^MZSQFF1	396
4D1006	^MSECOSQFF	425
4D2006	^MLISTSQFF	397
4D3006	^METASQFFext	397
4D4006	^SECOSQFFext	333, 422
4D5006	^CSQFFext	333
4D6006	^SUMSQRext	333
4D7006	^VXXLext	325
4D8006	^METALISTVXXL	325
4D9006	^VXXLFext	325
4DA006	^VXXL1ext	326
4DB006	^VXXL0	326
4DC006	^VXXL2NR	326
4DD006	^VXXL2	326
4DE006	^LIDNText	397
4DF006	^LVARXNXext	397
4E0006	^ISPOLYNOMIAL?	397
4E1006	^2POLYNOMIAL?	397
4E2006	^VXINDEP?	397
4E3006	^LVARXNX2ext	
4E4006	^RLVARext	397
4E5006	^LLVARDext	397
4E6006	^VXLVARext	397
4E7006	^LVARext	397
4E8006	^VX>LVARext	398
4E9006	^VX>	398
4EA006	^VX!	398
4EB006	^prepvarlist	73
4EC006	^LIDNTLVAR	398
4ED006	^LISTOPRAC	398
4EE006	^LISTOPext	398
4EF006	^LISTOPSQRT	398
4F0006	^LVARDext	398
4F1006	^>VARLIST	
4F2006	^DEPTHext	398
4F3006	^DEPTHOBJext	398
4F4006	^TRIMext	378
4F5006	^PTrim	378
4F6006	^TRIMOBJext	398
4F7006	^NEWTRIMext	399
4F8006	^>POLYTRIM	399
4F9006	^ELMGext	399
4FA006	^SWAPRNEG	348
4FB006	^QNeg	347
4FC006	^RNEGext	347
4FD006	^SWAPRRE	348
4FE006	^RREext	348
4FF006	^SWAPRIM	348

Addr.	Name	Page	Addr.	Name	Page
560006	^addtCEIL	365	001007	^ListToArry	65
561006	^xSYMCEIL	350	002007	^ArryToMatrix	65
562006	^addtIP	365	003007	^ArryToList	338
563006	^xSYMIP	350	004007	^DIMS	
564006	^addtFP	365	005007	^RunDoOldMatrix	
565006	^xSYMFP	350	006007	^RunDoNewMatrix	
566006	^addtXPON	365	007007	^DoNewMatrixReal	
567006	^xSYMXPON	350	008007	^DoNewMatrixCplx	
568006	^addtMANT	365	009007	^DoOldMatrixReal	
569006	^xSYMMANT	350	00A007	^DoOldMatrixCplx	
56A006	^addtLNP1	365	00B007	^DoNewMatrixRealOrCplx	
56B006	^xSYMLNP1	350	00C007	^DEB.MATRIX	
56C006	^addtLOG	365	00D007	^DEB.MATRIXTYPE	
56D006	^xSYMLOG	350	073007	^QpiZ	418
56E006	^addtALOG	365	074007	^QPI	418
56F006	^xSYMALOG	350	075007	^QpiSym	419
570006	^addtEXPM	365	076007	^QpiArry	419
571006	^xSYMEXPM1	350	077007	^QpiList	419
572006	^factorial	350	078007	^Qpi	419
573006	^facts	350	079007	^Qpi%	419
574006	^addtFACT	365	07A007	^GetRoot	419
575006	^xSYMFACT	350	07B007	^Approx	419
576006	^factzint	330	07C007	^#FACT	330
577006	^addtNOT	365	07D007	^CHECKSING	402
578006	^xSYMNOT	350	07E007	^DESOLVE	391
579006	^Verbose1	417	07F007	^ODE_INT	390
57A006	^Verbose2	417	080007	^LINSOLV	342
57B006	^Verbose3	417	081007	^LDECSOLV	391
57C006	^VerboseN	417	082007	^LDEGENE	391
57D006	^GETERABLEMSG	403	083007	^LDEPART	391
57E006	^ERABLEERROR	403	084007	^LDSSOLVext	391
57F006	^CANTFACTOR	403	085007	^ODETYPESTO	391
580006	^TRANSCERROR	403	086007	^ODE_SEPAR	392
581006	^NONUNARYERR	403	087007	^LAPext	392
582006	^INTERNALERR	403	088007	^ILAPext	392
583006	^INVALIDOP	403	089007	^ILAPRAText	
584006	^ISOLERR	404	08A007	^ILAPDELTA	
585006	^NONINTERR	404	08B007	^ILAPEXP	392
586006	^INTVARERR	404	08C007	^ILAPEXPSC	
587006	^Z>#ERR	404	08D007	^MENUext	409
588006	^Z<0ERR	404	08E007	^WRITEMENU	293
589006	^VXINDEPERR	404	08F007	^CFGDISPLAY	405
58A006	^NONPOLYSYST	404	090007	^NEWVX	405
58B006	^COMPLEXERR	404	091007	^NEWMODULO	405
58C006	^VALMUSTBE0	404	092007	^SWITCHON	405
58D006	^SWITCHNOTALLOWED	404	093007	^SWITCHOFF	405
58E006	^ERR$EVALext	404	094007	^FLAGNAME	405
58F006	^Sys1IT	404	095007	^COMPLEXON	406

Addr.	Name	Page	Addr.	Name	Page
096007	^COMPLEXOFF	406	0C6007	^STOMODULO	408
097007	^EXACTON	406	0C7007	^RCLEPS	408
098007	^EXACTOFF	406	0C8007	^ISIDREAL?	408
099007	^COMPLEXMODE	406	0C9007	^ADDTOREAL	408
09A007	^SETCOMPLEX	406	0CA007	^RESETCASCFG	408
09B007	^COMPLEX?	406	0CB007	^FRACPARITY	425
09C007	^REALMODE	406	0CC007	^POLYPARITY	380
09D007	^CLRCOMPLEX	406	0CD007	^PARITYTEST	
09E007	^EXACTMODE	406	0CE007	^COSTEST	
09F007	^SETEXACT	406	0CF007	^SHRINKEVEN	379
0A0007	^NUMMODE	406	0D0007	^SINTEST	
0A1007	^CLREXACT	406	0D1007	^SHRINK2SYM	379
0A2007	^EXACT?	406	0D2007	^SHRINKSYM	379
0A3007	^STEPBYSTEP	407	0D3007	^SHRINK2ASYM	379
0A4007	^NOSTEPBYSTEP	407	0D4007	^SHRINKASYM	379
0A5007	^VERBOSEMODE	407	0D5007	^FR2ND%	425
0A6007	^SILENTMODE	407	0D6007	^POLYSYM	380
0A7007	^RECURMODE	407	0D7007	^POLYASYM	380
0A8007	^NONRECMODE	407	0D8007	^P2P#	374
0A9007	^PLUSAT0	407	0D9007	^NDEvalN/D	394
0AA007	^SETPLUSAT0	407	0DA007	^PEvalN/D	394
0AB007	^PLUSATINFTY	407	0DB007	^POSITIFext	401
0AC007	^CLRPLUSAT0	407	0DC007	^SIGNE1ext	400
0AD007	^SPARSEDATA	407	0DD007	^SIGNEext	
0AE007	^FULLDATA	407	0DE007	^SIGNUNDEF	400
0AF007	^RIGORMODE	407	0DF007	^SIGNPLUS	400
0B0007	^SLOPPYMODE	407	0E0007	^SIGNMOINS	400
0B1007	^SLOPPY?	408	0E1007	^SIGNELN	401
0B2007	^MENUCHOOSE?	409	0E2007	^SIGNEEXP	401
0B3007	^MENUCHOOSE	409	0E3007	^SIGNESIN	401
0B4007	^MENUGENE1	409	0E4007	^SIGNECOS	401
0B5007	^MENUBASE1	409	0E5007	^SIGNETAN	401
0B6007	^MENUCMPLX1	409	0E6007	^SIGNEATAN	401
0B7007	^MENUTRIG1	409	0E7007	^SIGNESQRT	401
0B8007	^MENUMAT1	409	0E8007	^SUBSIGNE	401
0B9007	^MENUARIT1	409	0E9007	^SIGNERIGHT	401
0BA007	^MENUSOLVE1	410	0EA007	^SIGNELEFT	401
0BB007	^MENUEXPLN1	410	0EB007	^>SIGNE	401
0BC007	^MENUDIFF1	410	0EC007	^SIGNE>	401
0BD007	^PROMPTSTO1	167	0ED007	^SIGNMULText	401
0BE007	^XGROBext	98	0EE007	^ZSIGNECK	401
0BF007	^GROBADDext	92	0EF007	^SIGNEERROR	404
0C0007	^DISPLAYext	98	0F0007	^ZSIGNE	401
0C1007	^SCROLLext	281	0F1007	^zsigne	402
0C2007	^RCLMODULO	408	0F2007	^PASCAL_NEXTLINE	385
0C3007	^RCLPERIOD	408	0F3007	^DELTAPSOLVE	385
0C4007	^RCLVX	408	0F4007	^SOLVEMETASYST	342
0C5007	^STOVX	408	0F5007	^REDUCEMETASYST	342

Addr.	Name	Page	Addr.	Name	Page
0F6007	^REDUCEMETAPSYST	342	0080AB	xWIREFRAME	475
0F7007	^SOLVECRAMER	342	0090AB	xPARSURFACE	466
0F8007	^QUOTExSIGMA		00A0AB	xGRIDMAP	460
0F9007	^SUM	393	00B0AB	xYSLICE	476
0FA007	^FLAGSUM		00C0AB	xSLOPEFIELD	471
0FB007	^SUMVX	393	00D0AB	xPCONTOUR	466
0FC007	^FLAGSUMVX		00E0AB	xDIFFEQ	457
0FD007	^RATSUM	393	00F0AB	xVERSION	475
0FE007	^FTAYL	393	0110AB	xRECT	469
0FF007	^CSTFRACTION?	393	0120AB	xCYLIN	456
100007	^NONRATSUM	393	0130AB	xSPHERE	471
101007	^LINEARAPPLY	425	0150AB	xLININ	463
102007	^linearapply		0160AB	xLIBEVAL	463
103007	^meta_cst?	393	0170AB	xFLASHEVAL	459
104007	^HYPERGEO	393	0180AB	xCONLIB	456
105007	^fk+1/fk		0190AB	xCONST	456
106007	^A/B2PQR	425	01A0AB	xFFT	459
107007	^GOSPER?	425	01B0AB	xIFFT	462
108007	^ZEILBERGER	393	01C0AB	xNDIST	465
109007	^SYMPSI	393	01D0AB	xPSDEV	467
10A007	^sympsi		01F0AB	xPCOV	466
10B007	^SYMPSIN	394	0200AB	xRKF	469
10C007	^sympsin		0210AB	xRKFSTEP	469
10D007	^IBERNOULLI	394	0220AB	xRKFERR	469
10E007	^FLAGRESULTANT	380	0230AB	xRRK	470
10F007	^RESULTANT	379	0240AB	xRRKSTEP	470
110007	^RESULTANTLP	379	0250AB	xRSBERR	470
111007	^RESPSHIFTQ	379	0260AB	xCOND	456
112007	^ADDONEVAR	379	0270AB	xTRACE	474
113007	^IROOTS	382	0280AB	xSRAD	471
114007	^TYPEGAUSSINT?	200, 333	0290AB	xSNRM	471
115007	^DTYPEGAUSSINT?	200, 333	02A0AB	xRANK	468
116007	^DUPTYPEGAUSSINT?	201, 333	02B0AB	xLSQ	464
117007	^PPZZ	424	02C0AB	xEGV	458
118007	^DISTRIB*	368	02D0AB	xEGVL	458
119007	^NONALGERR	404	02E0AB	xSVD	473
11A007	^ALGCASCOMPEVAL	353	02F0AB	xSVL	473
11C007	^%%PSI	394	0300AB	xLU	464
1DC007	^PUSHFLAGS	405	0310AB	xQR	468
1DD007	^POPFLAGS	405	0320AB	xLQ	464
0000AB	xXVOL	476	0330AB	xSCHUR	470
0010AB	xYVOL	476	0340AB	xRREF	470
0020AB	xZVOL	477	0350AB	xRANM	468
0030AB	xXXRNG	476	0360AB	x→ROW	470
0040AB	xYYRNG	477	0370AB	xROW→	470
0050AB	xEYEPT	459	0380AB	x→COL	455
0060AB	xNUMX	465	0390AB	xCOL→	455
0070AB	xNUMY	465	03A0AB	x→DIAG	457