HP48 Java Library

Richard Steventon, Andre Schoorl, and Will Laughlin v3.0, 29 May 1997

Contents

1 What is Java?

Java is a library that displays an improved 5 level stack. It combines the best of SOL, HPSauce, EQStk, and others. Plus there are several new features and improvements! The intent of Java is to provide a more useful and faster stack environment for HP48 calculators.

2 Features

- Works on S/SX and G/GX
- 5-level stack, and fast 7-level interactive stack
- SOL's famous lowercase menus
- FULL path display with lowercase support in status area
- Improved status area, includes decimal mode, beep flag, and card status
- Option for oval/rectangular menu keys
- Autostart if not in covered memory and flag 64 is set
- Routines to help greatly improve stack speed!
- On-stack equations, arrays, and units with lowercase support!
- Configurable editor (alters keys and interactive stack editor)
- On-stack scrolling of `AGROB` output creates a sort of virtual desktop
- On the fly RAM card lock/unlock without warmstart
- Will run AUTOEXEC if it exists on startup
- Support for System RPL display if a recent copy of Jazz exists

3 Library Info

3.1 Copyright and Disclaimer

The Java library is freely distributed in the hope that it will be useful, but is provided 'as is' and is subject to change without notice. No warranty of any kind is made with regard to the software or documentation. The authors shall not be liable for any errors, incidental or consequential damages arising from use of the software or documentation.

Permission to copy the whole, unmodified Java package is granted provided that the copies are not made or distributed for resale (excepting nominal copying fees). If you put Java on CD-ROM and distribute it, you should send each of the authors a free copy. E-Mail us for our postal addresses.

The HP48 Java library has no connection at all to the Sun Microsystems product of the same name. Since the Java library is not a commercial product, we do not expect this to a problem.

3.2 Flavours

Java is now brewed in four delicious flavours: Espresso, Cappuccino, Mocha, and Decaf. Choose the one that best suites your taste and available memory.

Espresso

- ML Decomp for strings
- AGROB for algebraics
- General purpose full featured library

Cappuccino

- Built in decomp for strings
- Enhanced AGROB for algebraics, with all compiler flags set (includes user functions)
- Intended for math users

Mocha

- ML Decomp for strings
- Algebraics are handled by ML Decomp
- As fast as possible without AGROB

Decaf

- Built in decomp for strings
- Algebraics are handled by built in decomp
- Low size alternative to HPSauce

Since not all flavours are created equal, there will be differences in size and speed between the libraries. Furthermore, not all configuration flags will be meaningful in all flavours, but they are left the same for consistency.

3.3 Sizes and Checksums

- Lib. Number : 908
- Version : 3.0

java30e.lib (Espresso)

- Size : 30851.5 bytes
- Checksum : # F812h

java30c.lib (Cappuccino)

- Size : 24497 bytes

- Checksum : # 7506h

java30m.lib (Mocha)

- Size : 20877.5 bytes
- Checksum : # 615Ch

java30d.lib (Decaf)

- Size : 11807.5 bytes
- Checksum : # A7C3h

Java has been tested on SX ROMs E, J, and GX ROMs M, P, R. Backup your memory if in doubt.

3.4 Changes

New in Version 3.0 (from 2.9a):

- Added ML Decomp to greatly improve stack speed! (Will Laughlin)
- On-stack algebraics now based on EQStk 9.2 - includes cache for speed
- Rewritten port lock/unlock routine should fix previous problems
- Compiled four delicious flavours by popular demand
- Slightly modified orientation of lists in `AGROB`
- SysRPL mode no longer makes a copy of the Jazz tables unnecessarily
- Improved VAR and LIB menus to only redraw the stack when necessary
- Added page preservation code to UPDIR replacement
- Added review key replacement
- Strings and tagged objects are now scrollable
- Built in key support for MATRIX library if it exists
- `STK` and `JAVA` are smarter about when Java is running
- Redid documentation in SGML, allowing text, HTML, and PostScript formats
- Source code now maintained under the Concurrent Version System (CVS)

4 Installing/Deleting Java

4.1 Installing the Library

- Send the library to the HP 48
- Put a copy of the library on the stack.
- You may now `PURGE` the variable containing the library.
- Enter the port number you wish to store the lib (0/1 on GX, 0/1/2 on SX)
- Press `STO`
- Warmstart with `ON-C`, or power-cycle
- Install a UFL library that supports FNT2
- Run `STK` and configure Java. Select "Java Active" or run `JAVA`

4.2 Deleting the Library

- Press the `JAVA` key or clear flag 64 to quit
- The library is still referenced in memory, so warmstart with `ON-C`
- Type `:&: 908` then press `ENTER ENTER`
- Type `DETACH PURGE`
- The display should "flick" slightly - don't worry, it's normal.
- You can also purge the UFL if none of your other programs require it

4.3 Universal Font Library

In order to run Java, you will need need a copy of the Univeral Font Library (UFL) installed that supports FNT2. Java does not require FNT1 because it contains its own customized variable width smallfont.

Currently the only UFL libraries that support FNT2 are levels 2 and 3. As such, you will need to install either 'ufl2.lib' or 'ufl3.lib'. You must obtain the actual UFL package for these libraries as well as usage instructions, access routines, and the font maintenance library.

These should be available at the standard HP48 FTP sites, or at `<http://www.engr.uvic.ca/~aschoorl/ufl/>`

5 Commands

5.1 JAVA

Usage: (->)

Toggles the 5 level stack. Note that Java does not work in a GX covered port. Putting it there will result in an error on execution.

Although previous versions worked in covered ports, it was not recommended as it slowed down the HP48's display updates. This is because the HP48 will copy everything from "covered" memory to "temporary" memory as it cannot execute it otherwise. This resulted in a large performance loss.

5.2 STK

Usage: (->)

`STK` allows you to configure your user flags 57-64 for use with Java:

- 57 : Force small font for algebraics, units, and arrays
- 58 : Use built in presets for `ED/TED` editors (Jazz or TED)
- 59 : Use `EDIT` (either StringWriter or user-defined 'EDIT')
- 60 : Try to use SSTK type display if Jazz exists
- 61 : When set, turns off on-stack equation view
- 62 : Turn oval menus OFF when CST menu shows
- 63 : Oval menus on
- 64 : Java running

Keys while in `STK` are:

- Up and Down Arrow keys to scroll up and down
- `+/-` toggles the flag highlighted by the scroll bar
- `ENTER` exits and saves changes
- `ON` exits without saving changes

Note there is no OFF key capability or 10 minute timeout while in `STK`.

In order to use an alternate editor, you first need to install the editor itself. Otherwise you will only be able to use the HP's built in editor.

If you select both editors, `ED` will be used. To use the built-in editor just make sure neither ED or EDIT are selected. Selecting an editor will automatically alter your EDIT and DownArrow keys (and VISIT on SX) even out of user mode, unless you are in user mode and have your own assignment. It also changes the interactive stack editor.

To use StringWriter or an alternate editor, simply select EDIT in `STK`. Since StringWriter has an xlib called `EDIT`, this is all you have to do. There is no need to run StringWriter's `SWINIT`. Otherwise, just store a reference to your editor in 'EDIT'.

For example, to use HP-Writer:

```
HOME << HPW >> 'EDIT' STO
```

The EDIT variable can be anything suitable for a standard User RPL EVAL. The program should take a string on level 1 as its argument.

5.3 AGROB

Usage: (ob maxsize -> grob)

`AGROB` is the user interface to the algebraic to grob conversion utility. It takes an object from stack, along with the desired maximum font size, and converts the object into a grob. If the object is an algebraic expression or of some other suitable type, equation writer form output is produced.

The size argument should be a real number in the range 1 to 3, where 3 specifies the maximum possible starting font size.

`AGROB` with generate an error and leave the arguments on the stack if there is insufficient memory to create the grob.

5.4 FHOOK

Usage: (->)

`FHOOK` is a hidden command that does not appear in the Java menu, but can still be typed in or put in programs. It forces another check for Jazz Hook (used in SysRPL mode). This is only useful for GX users who use the `JAZZ` command to "uncover" their ports. After you `JAZZ` your port, just run `FHOOK` to update Java's local variables.

Note that `STK` also checks for Jazz Hook on exit, so if you always use `STK` to enable SysRPL mode, you do not need to use `FHOOK`.

5.5 CLRCACHE

Usage: (->)

`CLRCACHE` is another hidden command that does not appear in the Java menu. If Java is running, `CLRCACHE` clears the contents of the current AGROB cache. This is useful if you are changing the algebraic size flag manually or from within a program instead of using STK. If you always use STK to change Java's settings, you do not need to use `CLRCACHE`.

`CLRCACHE` is only available in the Java flavours that support AGROB.

6 Java's Features

6.1 Speed Enhancements

6.1.1 ML Decomp

The ML Decomp is a machine language routine by Will Laughlin that converts an object into a string - in other words, it "decompiles" the object. We call it the ML Decomp for short.

The reason Java needs such a routine is so it can display objects on the stack. When you push an object on the stack (say just the number 1) how does it get displayed? Every object on the stack is first converted to a string and then it is displayed. The binary representation of each object is totally independent of how it appears on the stack. Of course, things get a bit more complicated than this with the introduction of `AGROB`, which creates grobs instead of strings.

There are built in routines to convert objects to strings, but they are written in System RPL and are quite slow. The ML Decomp is written entirely in machine language so that it is fast. However, to accomplish this takes quite a lot of code, so the routine is fairly large. For this reason some flavours of Java contain the ML Decomp (Espresso, Mocha) and others don't (Cappuccino, Decaf).

6.1.2 Graphic Cache

Java also includes a 10 entry cache based on EQStk to speed up display of objects displayed with `AGROB`. This takes advantage of the fact that for the most part, objects on the stack don't change very much.

For example, suppose you have two large equations on the stack. The first on level 1 and the other on level 2. If you SWAP them, the objects themselves haven't changed - only the order on the stack has changed. A similar argument holds for duplicating or dropping objects.

Once an object on the stack has been converted into a grob for display, it is added to the cache. The next time it needs to be displayed, the grob can be obtained immediately from the cache, avoiding the overhead of running `AGROB` on it again. All flavours of Java that support `AGROB` (Espresso, Cappuccino) also include the cache.

The cache is automatically cleared by Java when a system flag that can affect display of algebraics is changed. This includes change of decimal mode (SCI/FIX/ENG) or angle mode. Java also keeps track of the CRC of every object in the cache.

6.2 Improved Status Area

Most people don't like seeing the user flags 1 to 5, so the status area has some new soft annunciators, as follows:

Miscellaneous

<bell>
: Error beep on

FIX/SCI/ENG
: Indicates decimal mode setting, blank for STD

NORM

Indicates user mode is OFF (normal keys)

Ports

I

ROM or write protected RAM card in port one

I (With box around)

RAM card in port one

I (With inverted box)

Merged RAM card in port one

The same goes for port 2, except it will look like a "II". Obviously, if there is no card in a port, you will not see anything.

The directory path starts with a triangle like in SOL. The rest of the directory path is separated with the '/' character. Ellipses (...) separate the path if it is very long. If selected, the ticking clock is shown as normal in the upper right area of the status area.

6.3 Scrolling of Level One Object

Java has the ability to "scroll" large objects on level one that normally would go off the side or top of the screen. This support uses `AGROB` to convert the object to a graphic and then uses a special on-stack viewer. Currently we support the following object types:

- Equations (Symbolics), Arrays (Matrices, Vectors), Units
- Lists, Grobs, Libraries, Strings

For other object types the key won't do anything. The scroller is accessed via the right shift picture key `RS-Picture`. On a S/SX this is `RS-Graph` (also known as key 34.3).

Just put one of the above objects on level one of the stack, and press the scroller key. Keys while in the scroller are:

- Arrow keys to scroll left/right/up/down
- `A - F` adjusts scroll speed (default speed is B)
- `ON` exits and returns to the normal stack

Note that list support in the scroller can be useful even if the list does not exceed the screen size. This is because the list is formatted by `AGROB`, making it a more appropriate for viewing output of utilities like QPI. The scroller is also handy for viewing graphics objects.

Even though `AGROB` is much faster than the Equation-Writer display, there can be a slight delay before the scroller is activated on very large objects. For this reason the scroll routine turns on both the left and right shift annunciators to indicate when it is active and ready for scrolling.

The scroller will generate an error if there is insufficient memory to make the grob.

6.4 Dynamic Port Lock/Unlock Feature

Note: This routine has been rewritten for Java 3.0

Provided you are in the stack environment, you can change the read/write switch on a free RAM card without turning your HP off! This is especially useful for people with large RAM cards (like 1MB) for eliminating long warmstarts. Of course, this doesn't apply to merged cards.

This feature should work for all cards 128K or larger. It will not work for 32K cards because the HP treats all cards as 128K by default (which is also why cards greater than 128K have to be bank switched).

Note that the port annunciators in the status area won't be updated until you press a key, such as `ON`. If you have sound turned on you will hear a slight "chirp" when the status area is updated.

A word of warning that applies to all stack programs: Do not try to free a merged card from which Java is currently running! The sudden change in address can cause unexpected results, including memory clear. If you wish to free a merged card, be sure to quit Java first. After the card is free you can start Java again.

6.5 Running of AUTOEXEC

Java will run a program called 'AUTOEXEC' if it exists in the user's HOME directory when it starts up, including warmstarts. This program can contain anything valid for a standard User RPL EVAL. If it does not exist, nothing will be run.

When building your program, remember that there is nothing on the stack. If AUTOEXEC generates an error, Java will not start properly and you will be left with the built in stack. In this case you will have to fix your AUTOEXEC and warmstart before you can run Java again.

Going into user mode and the VAR menu on startup is the default. If this is a problem for you, you can use the AUTOEXEC feature to override this at startup. For example, if you want the CST menu to come up on warmstarts, you can do:

```
HOME << 1 MENU >> 'AUTOEXEC' STO
```

6.6 UPDIR Replacement

The UPDIR key has been altered to give it page preservation abilities. This is similar to the UPD package by Joe Horn, or the UP command by Mika Heiskanen, but has been customized for Java.

Suppose you are in the VAR menu and press NXT several times, then enter a directory. If you press UPDIR in the built-in stack you will end up at page one and will have to press NXT several times to get back to the same page you entered the directory in the first place. With Java, if you press UPDIR you will return to the menu page you just came from.

6.7 Interactive Stack Menu Keys

Two new functions have been added to the IStack menu keys. You can find them on the second page of the menu:

->PRG

Nice little feature that puts a program together from the components on the stack. Using this, you can recreate programs that were exploded onto the stack with OB->.

REV

As its name implies, it will reverse the order of the stack objects between the cursor and level 1.

6.8 SysRPL Mode with Jazz

Note: This feature is intended for HP48 Hackers who use the Jazz library!

For SysRPL mode to work properly, you need to use Jazz 5.9 or higher. However, use of Jazz 6.4 or higher is recommended because of its support for the Universal Font Library (UFL), saving you some memory. At the time of this writing, Jazz 6.5 is the latest version.

Once you install Jazz, you will be able to display System RPL objects on the stack just as SSTK does without leaving Java or running multiple stacks on top of each other. Furthermore, your editor keys and review key will be enhanced.

6.8.1 Setup

First, install Jazz 5.9 or higher. See the documentation that comes with Jazz on how to do this. Be sure to also install the tables library (993) or you will just get addresses instead of names. The latest version of Jazz is available from `<http://www.hut.fi/~mheiskan/>`

"Light" versions of Jazz that contain the assembler, disassembler, a few other commands, and support for Java are available from `<http://www.engr.uvic.ca/~aschoorl/>`

Next, install Java normally as described above.

Java checks for the existence of Jazz on startup, so if you install Jazz after you install Java, use the `FHOOK` command to force another check, or just warmstart your HP with `ON-C`. Users with covered ports can use `FHOOK` after they `JAZZ` their port.

Once you have done this, you can enable or disable the SSTK type display at any time by using `STK`, or in your own programs by setting or clearing flag 60 respectively. If Jazz does not exist, or if you are using an old version, the SSTK type display will not be used.

6.8.2 Using the SSTK Type Display

Using the SSTK display does not in any way affect the on-stack algebraics, so you can have your algebraics and matrices symbolically at the same time as your programs are handled with Jazz. If you wish to have all objects on the stack to be handled with Jazz, just use `STK` to disable equation view (flag 61).

Note that if Jazz exists, Java will bind a few local variables to routines in Jazz. This means you probably will not be able to purge Jazz while Java is running.

If you compile your own light version of Jazz and you don't want all of SSTK just to use this feature, all you need is:

- stk.s : StkDis1, JavaDis1
- jazz.s : JazzHook

6.8.3 Editor Changes in SysRPL Mode

If you are in SysRPL mode (flag 60 set) the down arrow and interactive stack editors will call `DIS` and `ASS` (also in Jazz) automatically when starting and exiting your chosen editor. The exceptions are for strings (which don't need to be disassembled) or if `DIS` and `ASS` do not exist. If you wish to edit without disassembling, just use the EDIT key instead of down arrow, or temporarily exit SysRPL mode.

Also, if there is a string on level 2 and a real number on level 1 and you are using `ED`, you can just press down arrow to use `ED`'s offset feature to jump into the string. This is most typically used with the Jazz assembler to jump directly to an error location.

6.9 External Libraries

The most obvious external library supported by Java is the Universal Font Library (UFL), which Java requires for accessing FNT2.

The libraries directly supported for editing are Jazz, TED, and StringWriter. Other editors can be specified using the 'EDIT' variable. Jazz is also used in SysRPL mode.

Finally, the MATRIX library will be called for all matrix viewing/editing if it exists. Just install it and as long as Java is running all the necessary keys will automatically run it instead of the built-in MatrixWriter.

6.10 Accessing String to Grob Routines

You can access Java's string to grob routines (both smallfont and FNT2) by passing your string and the appropriate size to the `AGROB` xlib above. For example:

- `"Testing smallfont" 1 AGROB`
- `"Testing UFL FNT2" 2 AGROB`

Of course, since the UFL's fonts are configurable, the resulting graphic objects created by `AGROB` can also change.

7 On-Stack Algebraics

By default, algebraics will be converted to grobs and display on the stack in a similar form as equation writer, except much faster! They can be disabled in the `STK` screen if you like the standard form.

This is based on EQStk 9.2 but has a few differences:

- Slightly different grobs for AND, XOR, OR, EXP, summation
- Modified brackets and orientation for arrays and lists
- Disabled lists, grobs, lams, ids, and user functions on-stack since they were not suitable for a general use stack like Java. However with the exception of user functions, these types are still available in all flavours of Java via both the scroller and the `AGROB` xlib. User functions are only available in the Cappuccino flavour of Java.

- Support for vectors and cylindrical and spherical coordinates
- In low memory, objects will be passed to the string decomp instead of `AGROB`
- Smaller array size cap (20)

Java's `AGROB` uses the string decomp routine to decompile objects, with the following exceptions:

7.1 Tagged Objects

The tag is ignored, and thus the test defaults to testing for the following lists of exceptions.

7.2 Grobs

Grobs already are in the required form, so nothing is done.

7.3 Global name

Global names are displayed with the " delimiters. Note that 'Phi.2' is considered an ID and is displayed as in the normal stack. The following subscripts and special grobs only to apply to algebraics, such as 'Phi.2/3'.

Subscripts in algebraics can be generated as follows:

- By having an embedded dot character in the identifier. The first part of the name forms the actual displayed name. The second part of the name forms the subscript which is displayed in a smaller font. The second part can also have further dots to generate a subscript for the subscript itself.
- By having trailing decimal digits in the name. The digits will form the subscript in a smaller font.

In the following examples the decreasing font size is left for the imagination of the reader:

```
X.Y.Z   ==>     X                X1      ==>     X
                 Y                                1
                  Z
```

Due to the restricted internal font, and for easy access to some symbols the following names create special grobs:

- Psi, Phi, phi, Gamma, GAMMA, Planck, Nabla, Blank, Cont

The user should try typing above names when Java is active to see what the results are. Planck's constant is actually Dirac's constant (H-Bar) but is left the same for EQStk compatibility.

7.4 Local variable names

Local variable names are displayed without the " delimiters.

7.5 Units

Units are shown as in the built-in Equation Writer. For example:

```
                   m
1_m/s   ==>     1_--
                   s
```

7.6 Arrays

Arrays having more than two dimensions are shown using the current string decomp. For one or two dimensions the objects in the array are decompiled using the general rules for AGROB, the results are then arranged in a grid and are surrounded with brackets. One dimensional arrays are shown as column vectors.

Arrays containing more than 20 objects will not be passed to the AGROB in the stack, but you can still use the scroller or AGROB xlib.

7.7 Lists

Note: Java's list support only applies in the scroller or direct AGROB calls.

An empty list is converted using the current string decomp.

A list of lists all of equal length is considered a two dimensional array, and execution is similar as for arrays with the exception of using surrounding parenthesis instead of brackets.

Otherwise the list is generated as a column vector with surrounding parentheses.

7.8 Algebraics

All objects inside the algebraic are displayed using the regular rules. The functions in the algebraic then cause the grobs to be merged as is proper for the function. By default all functions simply display the name of the function followed by the arguments in parenthesis, with the following exceptions:

```
+       Arguments displayed with + in the middle
-       Arguments displayed with - in the middle
NEG     Argument  displayed with - in the front
*       Arguments displayed with a dot in the middle
/       Arguments displayed on top of each other w/ a line in between
^       Second argument displayed as exponent
INV     Argument displayed with "-1" exponent
SQ      Argument displayed with "2" exponent
EXP     Argument displayed as exponent for "e"
ALOG    Argument displayed as exponent for "10"
CONJ    Argument displayed with "*" exponent
ABS     Argument displayed with surrounding ||
SQRT    Argument displayed with surrounding square root
XROOT   Arguments displayed as written on paper
!       Argument displayed with following "!"
MOD     Arguments displayed with " mod " in the middle
```

```
NOT     Argument displayed with a special NOT symbol in front
AND     Arguments displayed with a special AND symbol in the middle
OR      Arguments displayed with a special OR symbol in the middle
XOR     Arguments displayed with a special XOR symbol in the middle
=       Arguments displayed with "=" in the middle
==      Arguments displayed with "==" in the middle
<>      Arguments displayed with "<>" in the middle
<       Arguments displayed with "<" in the middle
>       Arguments displayed with ">" in the middle
<=      Arguments displayed with "<=" in the middle
>=      Arguments displayed with ">=" in the middle
```

The following cases are a bit more complex:

Derivative

A series of derivatives with respect to X is causes the argument to be shown with the proper number of ticks (') as the 'exponent'.

A series of derivatives with respect to mixed variables is shown as taking partial derivatives of the argument.

Examples: (d denotes the derivative function)

```
                              ''   '
'dX(dX(Y))+a*dX(Y)+b=0'  ==>  Y  +aY +b=0

                               d
'dY(dX(F(X,Y)))=0'       ==>  ----(F(X,Y))=0
                              dYdX
```

Integral

Integral function is displayed as written on paper. As a special feature '?' as an integration limit is not displayed.

Examples: (S denotes the integral function, 00 infinity)

```
'S(0,00,EXP(-SQ(X)/2),X))=SQRT(PI/2)'

         00
          /  2
         | -x
         | --      ---
         |  2      |PI
==>      |e  dX =  |--
        /         \|2
        0

'S(?,?,S(?,?,S(?,?,1/(X^2+Y^2+Z^2),X),Y),Z)'

          ///
         |||   1
==>      |||--------dXdYdZ
```

```
 ||| 2  2  2
 |||X +Y +Z
///
```

Summation

Summation function is displayed as written on paper.

Example: (Z denotes the summation function)

```
'Z(I=1,n,SQ(I))=1/3*n^3+1/2*n^2+1/6*n)'
```

```
 n
--   2    1 3   1 2   1
>  (I ) = -n  + -n  + -n
--        3     2     6
I=1
```

| function

|is displayed as in the built-in Equation Writer.

7.9 User Functions

Note: User functions are only available in the Cappuccino flavour of Java.

The user can create their own functions for example by typing `'F(X,Y)'` directly or executing { `X Y` } `'F'` APPLY. The `AGROB` parser utilizes this feature for executing various functions on the arguments as follows:

```
'Mul(expr1,expr2)'      ==>  Multiplication with no mult. mark
'Der(expr,n)'           ==>  nth derivative of expr (tick form)
'DER(expr,n)'           ==>  nth derivative of expr (d/dx form)
'der(expr,n1,n2..)'     ==>  nth partial derivatives of expr
'Grad(expr)'            ==>  Gradient
'Div(expr)'             ==>  Divergence
'Curl(expr)'            ==>  Curl
'Lap(expr)'             ==>  Laplacian
'LapT(expr)'            ==>  Laplace transform
'FouT(expr)'            ==>  Fourier transform
'ILapT(expr)'           ==>  Inverse Laplace transform
'IFouT(expr)'           ==>  Inverse Fourier transfrom
'Cross(expr1,expr2)'    ==>  expr1 x expr2
'If(expr1,expr2)'       ==>  expr2 ,expr1
'So(expr1,expr2)'       ==>  expr1 ==> expr2
'Equ(expr1,expr2)'      ==>  expr1 <==> expr2
'PM(expr1,expr2)'       ==>  expr1 plus/minus expr2
'MP(expr1,expr2)'       ==>  expr1 minus/plus expr2
'pm(expr)'              ==>  plus/minus expr
'mp(expr)'              ==>  minus/plus expr
```

The following are more about formatting than actually providing some mathematical functions:

```
'Par(expr)'            ==>  Parenthesis are drawn around the argument
'Vector(expr)'         ==>  An arrow is drawn above the argument
'Top(expr1..exprN)'    ==>  The arguments are shown on top of each
                            other, aligned to left border.
'Lt(expr1..exprN)'     ==>  The argument is given a left brace "{"
                            For multiple arguments Top is called 1st
'Rt(expr1..exprN)'     ==>  The argument is given a right brace "}"
                            For multiple arguments Top is called 1st
'fm(expr)'             ==>  Decrement font size for expr
'fp(expr)'             ==>  Increment font size for expt
```

The user should note that all of above are just pseudo-functions, the only thing they do is to arrange the arguments in a convenient way. Thus for example true functions do not 'see' the pseudo-functions to have any special priorities, and thus parenthesis may not be shown when needed. The `Par` pseudo-function is provided for use in such cases.

To support the common notation for special functions the following patterns at the end of the function name are recognized:

```
'.n'    ==> First argument is shown as subindex.
            Remaining arguments, if any, are shown in parenthesis.
'.nm'   ==> First argument is shown as subindex, second as topindex.
            Remaining arguments, if any, are shown in parenthesis.
```

Examples:

```
'J.n(n,x)'      ==> J (x)       (Bessel function)
                     n

                     m
'P.nm(n-1,m,x)' ==> P   (x)     (Associated Legendre function)
                     n-1
```

8 Trying the Samples Directory

In the Java distribution you should find a file called 'samples.dir'. This is just a regular directory which you can download to your HP48. It demonstrates what `AGROB` and the scroller are capable of.

- Size : 2688 bytes
- Checksum : # 54E3h

Simply transfer it to your HP, store it in a variable, and enter the directory. Make sure Java is running and that on-stack algebraics aren't disabled in `STK`. Then just press any variable name in the menu in order to see the object in it.

If you notice an equation extends either off the right or top of the display, you can enter the scroller with `RS-Picture` (`RS-Graph` on S/SX) and use the arrow keys to move around. Similarly, to view lists of numbers in a more intuitive format, you can also use the scroller.

9 Using Java's Code

Remember that the Java library and its source can only be used for non- commercial purposes. This includes restrictions on using Java's code in shareware packages. See the copyright at the beginning of this document.

9.1 Notes on Hacking

If you use parts of Java in your own library then:

- Appropriate credit should be given in the documentation!
- Please drop us a note on how you like it!

If you use a library splitter, note the size of the resulting directory! There is no intentional library protection. Just remember that:

- Source is available on request
- Use SAD by Mika

If you plan on using `AGROB` or the ML Decomp routines, you should consult the respective authors in case they have different distribution restrictions.

9.2 Source Code Availability

Java requires GNU-Tools 2.1.9 or higher for proper compilation. But the source code is over 534 K uncompressed, so it is not included here. However, if you wish to have the source just mail one of the authors and you will be sent a zipped archive of the source attached with either Base64 or uuencode.

10 Frequently Asked Questions

I try to run JAVA, but I just get the error: UFL not Found

You need to install the Universal Font Library (UFL). See <http://www.engr.uvic.ca/ aschoorl/ufl/> to download the library and installation instructions.

Why aren't symbolic lists displayed like in EQStk?

Java is meant to be a general purpose stack, whereas EQStk is designed for math users. As such it is generally not appropriate to always display lists with `AGROB`. You can use the scroller to display lists and other objects with `AGROB`. The Cappuccino flavour of Java 3.0 will display objects like EQStk does, and is intended for math users.

Why doesn't Java run a program like AUTOEXEC at power on?

Java runs 'AUTOEXEC' on warm starts only. If you want to run something at every power on, it is trivial to assign a small program to your OFF key that first runs OFF and then the program you want to run. For example, if you want to go to the VAR menu every power on, you could do `<< OFF 1 MENU >> 91.3 ASN`

Why isn't there a Java without the status area?

As mentionned earlier, Java is intended to be a general purpose stack. For most users, the status area is more useful than another two lines of stack display. Since the status area is not updated in the interactive stack, it is a full seven lines.

Why doesn't Java show the available memory like HPSauce?

The reason Java does not show memory in the status area is because in order to provide an accurate reading for available memory, a garbage collection must be forced. This is inherently slow. Java is meant to improve stack performance, so we were unwilling to include a garbage collection every status update. The HPSauce library is still maintained separately for those who prefer it.

Java causes my HP to make clicking noises, then crashes.

You are using an old version of Java on a Rom R machine. There are known problems with the dynamic port lock/unlock routine (GetKeyOb). Either use the "light" version of Java 2.9a, or upgrade to Java 3.0.

Why doesn't Java support a full screen scroller?

The problem with this has to do with how to redraw the stack before and after invocation of the scroller. Do you get rid of the status area, forcing a full stack redraw? Seems slow. What about tiny objects that don't even reach the status area? Do you provide two versions of the scroller for such cases? Seems bloated. If you require a full screen scroller, you can use `AGROB` to convert an object to a grob, then use an external scroller like `VV` to view it.

Why doesn't Java have more configuration options?

The main reason for not adding more configuration options is because of the `STK` configuration program. The current configuration flags occupy the entire screen - so to add more we would either have to use multiple screens or make `STK` scrollable. Both ways would make `STK` much larger. So unless there is significant reason to do so, we have been relucant to add additional configuration flags.

Why does Java have its own smallfont instead of using FNT1?

Java has its own smallfont because it is variable width, just like the built-in one. FNT1, on the other hand, is a fixed width font which is fine for most purposes but is unattractive for use in menus. Some people may not mind this, so some day there may be a Java that uses FNT1. However, a number of core routines will have to be rewritten.

Java makes a copy of my Jazz tables in a covered port.

Java 2.9a had a problem in that it made a copy of the Jazz tables if they were in a covered port even if you did not have the SysRPL mode flag set. Java 3.0 will no longer make a copy if you do not set the SysRPL mode flag. However, it still has to make a copy if you use SysRPL mode because of the bank switching inherent with covered ports.

Can I obtain the PostScript documentation for A4 paper?

The PostScript provided with Java is formatted for letter size paper. SGML source for all the documentation is included with Java's source code, so if you request a copy of that you can use the sgml-tools package to make any format you like. An easier way is just to print the HTML version.

11 Thanks

(In alphabetical order)

Paquot Christophe

For some optimizations to the display routines.

Hervi Couplet

For the original GetKeyOb replacement.

C.N. Fiechter

For the original `AGROB` (from AVW48).

Rick Grevelle

For help with bitmap shifting.

Mika Heiskanen

For Jazz in HP mnemonics, for EQStk and for use of some of its documentation.

Detlef Mueller

For rewriting the GetKeyOb replacement and for SysRPL mode improvements.

Joe Horn

For suggestions about CST menus and ovals (flag 62).

Dan Kirkland

For immortalizing the words: "Mine WILL be better!"

Mozgy

For telling me I *have* to add a 5 level IStack!

Jack Levy

For testing and ideas on the status area and scroller.

As well as Jeff Sketchley, Keith Farmer, Othniel Rawlins, Alfred Arduengo, and others for beta testing.

Everybody else on IRC channel #hp48 for comments etc... And all those on `<news:comp.sys.hp48>` who gave suggestions and posted bugs!

12 E-Mail Addresses

We can be reached via electronic mail at the following addresses:

Andre Schoorl

<`aschoorl@engr.uvic.ca`>

Richard Steventon

<`rsteven@engmlab.uct.ac.za`>

Will Laughlin

<laughw@rpi.edu>