LittleSmalltalk – The Primitives

Author: Danny Reinhold
Date: March, 30th 2007
Status: Published
Version:0.1
Describes LST: 5.0 alpha 009

Summary

This documents lists and roughly describes the list of primitives in the LittleSmalltalk system.
Please note that this set of primitives is subject to change and may be extended from release to release.
If you need to use primitives in your Smalltalk code please encapsulate the primitive calls in a central place so you can change the calls if necessary.
Currently I don't plan changes – only additions – but I don't guarantee that I will never change the currently present primitives!

What is a primitive in LittleSmalltalk?

When you write programs with LittleSmalltalk you will usually do this by writing code in the Smalltalk dialect provided by LittleSmalltalk. But sometimes (well, currently often) you need to communicate with the virtual machine (VM) or the operating system etc. more directly.
For doing so the LittleSmalltalk system provides a mechanism called primitives.
A primitive is a piece of ANSI-C code in the virtual machine that can be called and parameterized by your Smalltalk code and that can create results that you Smalltalk code then can evaluate.

How does the primitive mechanism work?

To understand how the primitive mechanism is realized in LittleSmalltalk we have to look at both sides: The Smalltalk side and the C side within the virtual machine.

How to call a primitive within Smalltalk code?

Every primitive is identified by a unique number.
To call a primitive simply write this number between "<" and ">".
For example to call primitive 85 you could write:

```
<85>
```

If the primitive returns a useful result you can use it as you would use any other Smalltalk object like this:

```
| tmp |
tmp <- <85>
Transcript print: ('The result is: ' + tmp asString).
^tmp
```

Most primitives require one or more arguments. Simply add these arguments to the primitive call like so:

```
<85  'Argument1'  2  #ArgumentThree>
```

In this example the primitive 85 would consume three arguments, one String, one SmallInt and one Symbol object.

Attention: Be careful to always provide arguments of the correct type! The primitive implementations within the virtual machine usually don't convert any objects to other classes and may cause problems for example when a String is expected and a Symbol is given!

How does the mechanism work in the VM?

Look into the file interp.c. This file represents the main part of the execution engine of the virtual machine. You will find a huge case statement checking for the bytecode "Opcode_DoPrimitive".

Within that case block you'll find another larger switch statement that checks which exact primitive shall be executed.

Most elementary system primitives are for performance reasons directly implemented within their respective case blocks and have rather small ids.
Some blocks of primitives are implemented in other files, namely gui_primitives.c, ffi_primitives.c and sqlite_primitives.c. Some io related primitives are implemented in main.c.

If a primitive id is not directly implemented in interp.c, the VM calls the function primitive() in main.c. If the primitive id belongs to the more specific primitive blocks the primitive call is delegated to the special implementation file.

The LittleSmalltalk list of primitives

These primitives are currently available in the LittleSmalltalk system.
In some cases the primitive id is not specialized enough to identify a single action (like "open dialog") but to identify a logical group (like "gui actions"). In those cases the first argument to the primitive is interpreted as an action id that specifies the detailed action within the action group.

Id	Action Id	Name	Arguments	Description
1	-	Object Identity	2 objects	Returns true if the arguments refer to the same (!) object (identity test – not equivalence test!)
2	-	Object Class	1 object	Returns the class the argument
3	-	Print Character	1 Char	Prints a single character to stdout. Please note that with a normal LittleSmalltalk distribution stdout is not any longer connected to a terminal when you are using the GUI only version
4	-	Object Size	1 object	Returns the size the object in bytes. This is the size that is required to store the object in memory
5	-	Object #at:put:	3 object index object2	Stores object2 into member variable index of object object. Note that index must be a SmallInt (0 <= index < 1000).
6	-	New Process Execute	1 Process	Executes the specified process (note that this doesn't mean OS level processes, but Process objects within LST)
7	-	Create New Object	2 size class	Creates an object of class class with size size and allocates memory for it. The object is returned. Please note that size has to be a SmallInt (0 <= size < 1000).
8	-	Block Invocation		Executes a block
9	-	Read Characater		Reads a single character from stdin and returns its ASCII value or nil if stdin reached EOF. Please note that GUI only versions of LST are not connected to a terminal and don't have an stdin channel open.
10	-	SmallInt Addition	2 SmallInt	Adds the arguments and returns the result
11	-	SmallInt Division	2 SmallInt	Division on SmallInt arguments
12	-	SmallInt Remainder	2 SmallInt	Remainder operation on SmallInt arguments
13	-	SmallInt <	2 SmallInt	< operator on SmallInt arguments

14	-	SmallInt =	2 SmallInt	Equality (equivalence) test on SmallInt arguments
15	-	SmallInt *	2 SmallInt	Multiplication of two SmallInt arguments
16	-	SmallInt -	2 SmallInt	Subtraction on SmallInt arguments
17	-	-	-	Not specified
18	-	Debugging ON	-	Activates some debugging messages (for example regarding the GC). Has effect only if LST has an open stdout channel
19	-	Error Trap – HALT		Stops execution of the current LST process (not on OS process level)
20	-	Byte Array Allocation	2 size class	Creates a byte array object of class class and with size size. Please note that size must be a SmallInt (0 <= size < 1000)
21	-	String at	2 string index	Returns the character at index index in the string
22	-	String at:put:	3 string index char	Sets the character at index index of the string to char
23	-	String clone		Clones a string object
24	-	Array at	2 array index	Returns the item at index index of the array. Please note that index must be a SmallInt (0 <= index < 1000)
25	-	New Object Allocation	2 class size	Creates and returns a new object of class class and with size size. Please note that size is a large integer passed as a string.
26	-	Byte Array Allocation	2 class size	Creates and returns a byte array object of size size. Please note that size is a large integer passed as a string.
27	-	String at	2 string index	Returns the character at index index. Please note that index is a large integer passed as a string.
28	-	String #at:put:	3 string index char	Sets the character at index index in the string to char. Please note that index is a large integer passed as a string.
29	-	Object #at:put:	3 object index object2	Sets the member variable at index index in the object object to object2. Please note that index is a large integer passed as a string.
30	-	File Open	2 string string2	Opens the binary file string with mode string2 ('r', 'w', 'a'). Returns a file id.
31	-	File Read Char	1 fileId	Returns the next character from the file.
32	-	File Write Char	2 fileId char	Writes a character to the file file
33	-	File Close	1 fileId	Closes the file
34	-	File Out Image	1 fileId	Write the LittleSmalltalk image to the file file
35	-	Edit String	1 string	Opens an external editor and allows the user to edit the argument string. The modified string is returned. Please note that this primitive is obsolete and will be removed in the future.

36	-	SmallInt To Int	1 SmallInt	Returns an Int object representing the same number as the SmallInt argument does
37	-	Int to SmallInt	1 Int	Converts an Int object into a SmallInt object if possible (ie if 0 <= value < 1000).
38	-	Memory Allocate	1 size	Allocates size bytes of memory via malloc()
39	-	Memory Read Byte	1 address	Reads one byte of memory from the specified address
40	-	Memory Write Byte	2 address byte	Writes the byte byte to the memory at the specified address
41	-	Memory Allocate Int Array	1 size	Allocates memory for an array of integer values via malloc().
42	-	Memory Read Int	1 address	Reads an integer value from the specified memory address
43	-	Memory Write Int	2 address integer	Write an integer value to the specified address
44	-	Memory Free	1 address	Deallocated the specified memory via free()
45	-	File Open	2 string string2	Opens the file string with mode string2 ('r', 'rb', 'w', 'wb' etc.). Returns a fileId.
46	-	File Write String	2 fileId string	Writes the string to the file
47	-	File Delete	1 string	Deletes file string.
70	-	Array at	2 array index	Returns the object at index index from the array. Please note that index is a large integer passed as a string
80	-	Object Set Class	2 object class	Changes the class of an object
81	-	Flush Method Cache	-	Flushes the method cache. This is done after compiling a Smalltalk method.
160	-	Show Message	2 string string	Displays a message box containing the string with the specified title.
161	0	Create Canvas	1 string	Creates a canvas control
161	1	Create Button	2 string string	Creates a button control
161	2	Create Toggle	1 string	Creates a toggle control
161	3	Create Label	1 string	Create a label control
161	4	Create Frame		
161	5	Create List		
161	6	Create EditBox		
161	7	Create Texfield		
161	8	Create Image		Create an image from memory
161	9	Create Color		
161	10	Create Dialog		
161	11	Create File Dialog		
161	12	Create Gauge		

161	13	Create Timer		
161	14	Create SplitBox		
161	15	Create Image		Creates an image from a file
162	0	Create Spacer		
162	1	Create HBox		
162	2	Create VBox		
162	3	Create ZBox		
162	4	Create Radio Box		
163	-	Append Widget		
164	-	Detach Widget		
165	-	Show Widget		
170	-	Register Event		
171	-	Get Next Event		
172	-	Event Loop		Starts a blocking event loop. For testing purposes only. Don't use this primitive within your code.
173	-	Sleep		
174	-	Destroy Widget		
175	-	Set Attribute		
176	-	Get Attribute		
177	-	Register List Event		
178	-	Delete Attribute		
179	-	Show Modal Widget		
180	-	Save Image		
200	-	Open Database		
201	-	Close Database		
202	-	Execute SQL		
203	-	Get Last Row Id		
204	-	Execute Query		
205	-	Release Table		
206	-	Get Error Message		
207	-	Get number of rows in Table		
208	-	Get number of columns in table		
209	-	Get Value from table		

230	-	Open DLL		
231	-	Close DLL		
232	-	Get Last DLL Error		
233	-	Resolve Function		
234	-	Execute Function		Attention: This is subject to change!