Orb Leaf
Redefining Standards

Orb-Weaver
Programming Manual

Write Seamlessly, Works Securely

1.2.8

Table of Contents

1. Orb-Weaver Introduction

Orb-Weaver is a high level programming platform for Orb-Weaver based smart card device, the programming language itself is subset of ECMA-262 (Javascript) specification for ease of development. The basic of Orb-Weaver is act as bridge between human high level language and low level machine interpretation through the use of native APIs and virtual machine, the previous smart card development platform require understanding of low level interface between smart card device and host application, this problem could be eliminated in Orb-Weaver through the use of native Apis.

Orb-Weaver designed with several features such as

- Seamless operation, integration between native APIs and interpreted codes, development to deployment and installation all done within a single system.
- Small memory footprints, Orb-Weaver based on stack machine therefore it require less memory in order to run
- String based operation, all operation on Orb-Weaver are treated as string therefore weak programming type applied
- Safe, all operation in Orb-Weaver are done through virtual machine (sandbox), an exception might be generated but would not affecting real hardware
- Scalable, to extend the possibility between each Orb-Weaver application, each application is scalable in it's own way
- Secure, each application installed on Orb-Weaver based device is a secure application, each with their own private data and APIs, accessing Orb-Weaver application directly from card is nearly impossible without breaking the card
- Simplicity, Orb-Weaver limited to card based device, a simple programming language derived from ECMA-262 employed for this purpose to increase the programming curve. their own toolkit application.

2. Orb-Weaver System Architecture

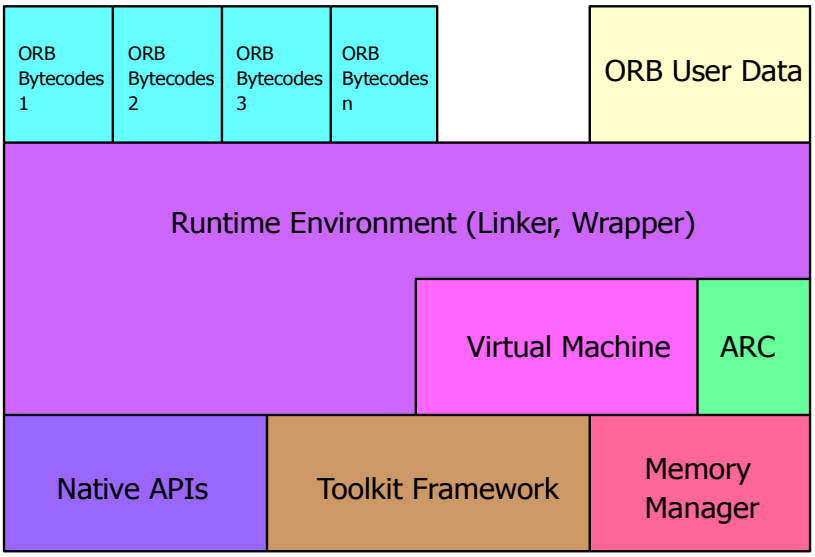


Fig 1.0 Orb-Weaver Architecture Design

Orb-Weaver main architecture consists of runtime environment, virtual machine, ARC (Automatic Reference Counting) garbage collector and toolkit framework.

3. Orb-Weaver Script Language

Orb-Weaver support standard generic programming language features such as mathematical operation, conditional branch, unconditional branch, loop operation, lazy language, structured (2nd generation language) with virtual "object-oriented" support for extending APIs. Unlike programming flow of native application where user must create it's own program loop to handle event, Orb-Weaver treat all functions as callback (menus and events), which means it must return it's runtime to the caller.

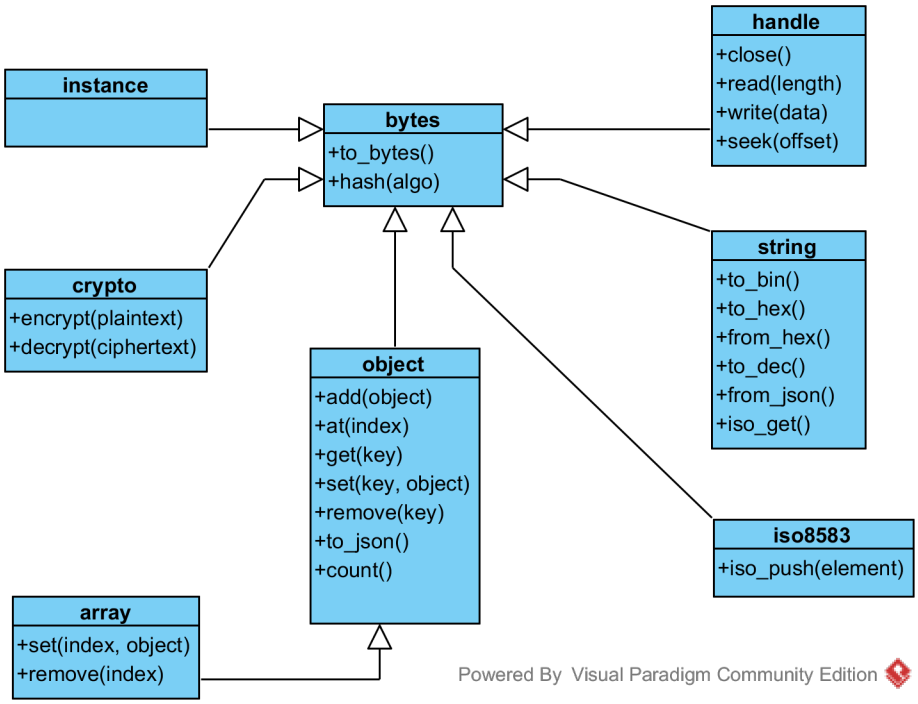


Fig 1.1 Variable types and their corresponding APIs

Each variable can be treated as sequence of bytes, different type of variables are inherited from bytes type, Fig 1.1 represent different type of variable scope and their corresponding APIs, invoking APIs from an undesignated variable might result in an exception or invalid result.

Class Declaration

Class declaration needed by Orb-Weaver compiler to extend package methods for exporting use, Orb-Weaver by default forces developer to declare class for each package. Class ideally used as package reference during install command to prevent same class exist simultaneously on same card.

Example :

```
public class MyClass { /* empty class */ }
```

Method Declaration

Package method must be declared inside class, for exported method function must be marked as public, otherwise it can only be accessed internally.

Example :

```
public class MyClass {
      public function MyMethod() {
            return "This is Method";
      }
}
```

A method could contain method arguments, total number of maximum arguments within a method is 15 arguments. Any method arguments that will be accessed inside method must be declared first, any method calling passing arguments more than the declared ones will be marked as variadic arguments (user defined method cannot access variadic arguments which doesn't defined previously, variadic arguments could only be accessed by native APIs).

Example :

```
public class MyClass {
	public function Method1(arg1) {
		return arg1;
	}
	public function Method2(arg1, arg2) {
		return arg2;
	}
	public function Method3(arg1) {
		var a = Method1("a", "c");		//"c" will be ignored by Method1
		var b = Method2("b");
		//arg2 automatically set to null
		return a + b;				//only return "a"
	}
}
```

Orb-Weaver compiler purpose was to encapsulate low level operation on toolkit application, toolkit APIs such as SETUP_MENU and SETUP_EVENT automatically handled by Orb-Weaver runtime during package loading automatically. In order to install method as menu or event, developer should use keyword **alias** and **event** during method declaration.

1. Installing Method as Menu

   when a method installed as menu it will automatically executed during terminal proactive command MENU_SELECTION as callback, addition or removal of menus can be done by adding or removing a package during card lifecycle.

   Example :

   ```
   public function MyMenu() alias "Menu 1" { }
   ```

   any string constant following **alias** declare the text that showed up during SETUP_MENU proactive command. Order of the menu showed up on SETUP_MENU list depends on which package installed first, the first installed package automatically made up to the top menu and vice-versa.

Variable Declaration

Variable declaration only can be done inside method, class didn't support variable declaration like others object oriented programming. Use keyword **var** to declare a new variable, by default virtual machine will initialize it's default value to **null**.

Example :

```
public class MyClass {
	public function MyMethod() {
		var r = "This is variable";
		return r;
	}
}
```

Mathematical Operation

Mathematical operation support operation between variable and constant value, the resulting operation must be stored on variable. In case one of the variable didn't contain numerical value operation will generate an exception to the resulting value, see table for detail of operation.

Operand 1	Operand 2	Opcode	Result
hello	world	Addition (+) Substraction (-) Division (/) Multiplication (*) Modulus(%)	helloworld NaN NaN NaN NaN
hello	1234	Addition (+) Substraction (-) Division (/) Multiplication (*) Modulus(%)	hello1234 NaN NaN NaN NaN
5678	world	Addition (+)	5678world

		Substraction (-) Division (/) Multiplication (*) Modulus(%)	NaN NaN NaN NaN
1234	5678	Addition (+) Substraction (-) Division (/) Multiplication (*) Modulus(%)	6912 -4444 0 (automatically rounded) 7006652 (in-case the resultant operand bigger than 2147483647 or lower than -2147483648, it will automatically modulated) 1234

Lazy Operation

For developer easeness Orb-Weaver compiler also support lazy operation such as increment after, increment before, decrement after or decrement before. Lazy operation only worked on variable with numerical value, any value other than numerical would cause the resulting operation generate an NaN (Not a Number) exception.

Example :

```
var a = 0;
var b = 4;
var c = b + a++;         //c = 4, increment after
var d = a;         //d = 5
var e = b + --a;         //e = 4, decrement before
```

Unconditional Branch

Unconditional branch in Orb-Weaver can be done via **goto-label** statement within a method, any goto operation referencing label outside a method will cause compiler error.

Example :

```
goto MyLabel;
a = a + 1;
MyLabel:
return a;
```

Conditional Branch

Orb-Weaver support several combination of conditional branch such as **if-else** statement and **switch-case** statement. For branch operation with more than one expression evaluated use if-else combination, switch-case can only be used on single variable expression.

Example :

```
switch(a) {
      case "yes":
            break;
      case 1:
            break;
      default: break;
}

if(a == "yes" && a != 1) { }
else if(a == "no") { }
else { }
```

Loop Operation

Orb-Weaver support **for** statement and **while-do** statement for loop operation, by using **goto-label** combination developer also could create loop operation.

Example :

```
var j = 1;
while(j!=0) { j--; }
for(var i=0; i<5; i++) {  }
```

Substring Operation

By default Orb-Weaver compiler didn't support Array within variable declaration, but developer could treat any variable as array later during operation or by using `array()` API. To concat several decimal value to be used for later processing during runtime use `array()` API, it can be used to receive any dynamic value which are assigned on runtime only, user input, binary data for example.

Example :

```
var MyArray = "123456789";
var LengthMyArray = sizeof(MyArray);//LengthMyArray  = 9
var MySubArray = MyArray.substr(0,5);//MySubArray = 12345
var SplittedString = "Hello World".split(" "); //split string to array
```

Object/Array Operation

Orb-Weaver supporting object/array operation with Orb-Weaver object (OWB) which is based on ASN.1 (for use with constrainted environment), access to variable could directly handled on compiler level. Later each object/array can serialized/deserialized to/from it's JSON string through the use of native API, to facilitate easier object/array access (including nested object/array access) one could use (dot) or [index] operator to access each object or array directly from compiler.

Example :

```
//define a key-value pair
var KVPair = { key : "value" };
//encapsulate within object
var MyObj = { my : KVPair };
//push an object within array
var MyArray = [1, 2, MyObj ];
//return number of object listed within array
var count = MyArray.count();
//retrieve object from array at index 2
var SelObj = MyArray[2];
//retrieve object from object with key (nested access)
var SelKV = SelObj.my.key;
```

JSON Support

Orb-Weaver object supporting encoding/decoding to/from JSON format to be used with later system. Processing any JSON string directly will be treated as normal string data, the string need to be converted to Orb-Weaver Object before it can be processed directly.

Example :

```
//define a json string
var jsonStr = "{ key: \"value\" }";
//convert to owb object
var owbObj =  jsonStr.from_json();
//retrieve specified key from object
var valObj = owbObj.key;
//convert back to json string
var jsonStr2 = owbObj.to_json();
```

Global Variable APIs

Global variable APIs provide mechanism for storing/loading variable for current instance, this similar with global variable on object oriented language. Accesing current instance can be done by calling `this()` API directly, you cannot assign current

instance returned by `this()` API to variable or as return value as it can be used to access current instance from another instance, the problem might arise when returned instance is released from memory as it might cause exception, system cannot determined whether the instance has been released or not, so you must access `this()` API directly from active class.

Example :

```
public class authenticator {
    public function init() {
        this()->ctx = cr_create("DES", "CBC", cr_random(24));    //3DES with random keys
    }

    public function auth(pass) {
        return this()->ctx.encrypt(pass);
    }

    public function my_ctx() {
        return this()->ctx;
    }
}
```

All assigned global variables are automatically released when their class instance collected by garbage collector, do not attempt to access assigned variable from caller instance, `this()` APIs cannot be assigned to any variable, though you can return the value of variable upon function return.

Wrong Example :

```
public class i_want_to_authenticate {
    public function main() {
        var auth = new authenticator();
        auth->init();
        var val = auth->ctx.encrypt("12345678");     //DO NOT DO THIS
    }
}
```

Right Example :

```
public class i_want_to_authenticate {
    public function main() {
        var auth = new authenticator();
        auth->init();
        var val = auth->my_ctx().encrypt("12345678");      //use method to get the var
    }
}
```

Persistent Storage APIs

Persistent storage APIs provide mechanism for storing/loading variable from persistent storage, Orb-Weaver didn't support global variable declaration therefore to create shared variables between callback persistent storage APIs shall be used. Persistent storage APIs are based on key-value pairs mechanism and shareable throughout the entire Orb-Weaver applications.

Example :

```
public class sample {
    public function start_up_handler() event 15 {
        if(data()->reg == null) {
            start_registration();
            data()->reg = "yes";
        }
    }
```

```
    function start_registration() {
        if(data()->lang != null) {
            data().delete("lang");
        }
    }
}
```

But unlike global variable do not attempt to assign an instance or context to persistent storage, any instance or context are automatically released during machine termination therefore accessing their previously allocated memory area would result in system violation by operating system.

Wrong Example :

```
public class authenticator {
    public function init() {
         data()->ctx = cr_create("DES", "CBC", cr_random(24));     //DO NOT DO THIS
    }
}
```

Lambda Expression

Orb-Script also supporting lambda expression for use within function, lambda expression allowing function declaration to be declared as variable, therefore enabling the function to be passed as argument. There are several limitation when declaring lambda function within classes such as.

1. Function assignment to variable can only be done by another function or parent function within the same class where the lambda function is being declared, passing a lambda function of a method of another instance would result in system exception, but passing a variable that has been assigned with specific lambda function within the same class is allowed, enabling the function to be used as callback method.

2. Assigning pre-defined variable of parent function to lambda function during function declaration (as in JavaScript) is not allowed, it would trigger a compiler error. Unlike JavaScript where program are being interpreted, Orb-Weaver is a virtual machine compiler (hybrid compiler), therefore any pre-defined variable of parent function didn't assigned with it's value until it is being run on virtual machine, therefore the compiler cannot assign any value during lambda declaration (or make assumption of it's pre-defined value) as it would create an ambiguity between parent function and lambda function, but assigning a pre-defined value on lambda function variable would still allowed.

There are two type of lambda operation that can be done within Orb-Weaver,

1. Declaration, function declaration could occur within another function, typically declared lambda function are assigned to variable, passed as argument or returned as return value.

2. Execution, execution are done by passing argument to specific variable that had been assigned with lambda function, a mismatch in function argument or executing an invalid variable (not assigned with lambda function) would result in system exception.

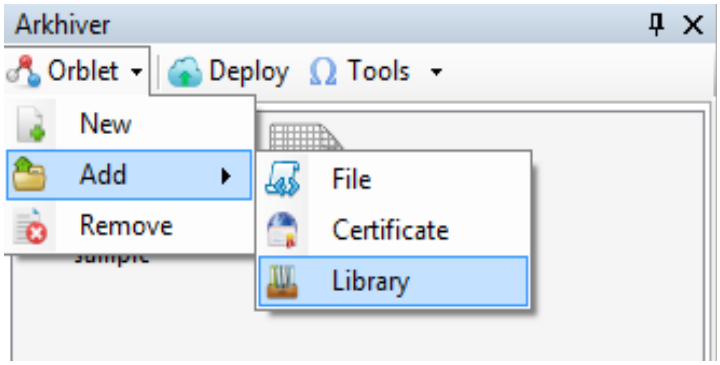
Example :

```
public class lambda_test {

    function callee(x, y) {
        return x + y(x);
    }

    public function caller() {
        var v = callee;
        var result = v(1, function(x) {
            var t = 1;
            return x + t;
        });        //result = 3
    }
```

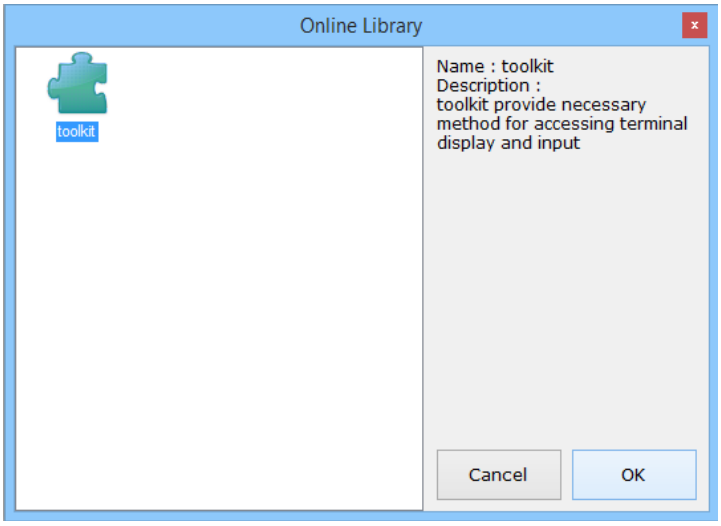
```
    function returning_lambda() {
        return function(x, z) { return x * z; };            //returning function
    }
}

public class lambda_invalid {

    function callee(x, y) {
        return x + y(x);
    }

    public function caller() {
        var t = "text";
        var v = callee;
        var result = v(1, function(x) {
            return text + x;                //invalid variable declaration within lambda
        });
    }
}
```

Invoking External Class

Like high level programming language, invoking external class as instance is possible within Orb-Weaver, this is needed to provide programmer with external framework possibility, later could decrease development time. Before a class can be invoked, programming need to add the class to current project, the class can either be sourcecode or an Orblet file (precompiled classs), the project addition of external class can be done through online library on Organ Studio by clicking Orblet > Add > Library from Project Explorer module



select toolkit library and click OK, new library will automatically added to project list



Example :

```
var t = new toolkit();
//case class toolkit has method name method1 with one argument
t->method1("external invoke");
```

Through use of online library, it enable collaboration with another vendor for accessing 3rd party services or devices, it eliminates the needs to locally distribute each library independently.

HTTP/HTTPS Request

Accessing HTTP/HTTPS on Orb-Weaver could be done through external library, use square bracket to create an array of parameters for request payload, the square bracket automatically create a new Orb-Weaver object to be passed on method argument.

Example :

```
public function register() alias "Register" {
	var t = new toolkit();
	t->http_get("Register", "http://www.mysite.com", [
		"name":"me"
	]);
}
public function register() alias "Register" {
	var t = new toolkit();
	t->http_post("Register", "http://www.mysite.com", [
		"name":"me"
	]);
}
```

in case a secure transaction is needed you can add SSL certificate to use SSL mechanism (SSLv3, TLSv11, TLSv12) to secure transaction, the detail are explained in Organ Studio chapter 4 (Adding Certificate for SSL Operation). To invoke SSL request for HTTPS use HTTPS prefix as request URL.

```
public function register() alias "Register" {
	var t = new toolkit();
	//secure transaction
	t->http_post("Register", "https://www.mysite.com", [
		"name":"me"
	]);
}
```

Detail of each method would refer to OrbLeaf Toolkit Framework documentation (or other 3rd party Framework), this documentation only provide some OrbLeaf Toolkit example and should be dismissed in-case there are some differences between the sample code in this document and actual framework documentation. OrbLeaf would maintain to keep all deprecated method intact according to Pattern Of Least Astonishment (POLA), should there's a change in some methods/APIs it would not affect old application.

Cryptography

Cryptography on Orb-Weaver can be done through combination of native and object APIs, Orb-Weaver currently supporting Data Encryption Standard (DES56, DES112, DES168) and Advanced Encryption Standard (AES128), any processed data or key would automatically padded with zero (Padding Method 1) in-case there are some differences between argument length and expected length. Advanced algorithms might be added on future release, see Orb-Weaver Native APIs table for detailed reference.

Example :

```
var crypto = cr_create("DES", "CBC", "1234567812345678");
```

```
var cipher = crypto.encrypt("my sample text");
var plain = crypto.decrypt(cipher);
```

For DES operation the key length determined the algorithm being used with 8 bytes for DES 56 bits, 16 bytes for DES 112 bits and 24 bytes for DES 168 bits.

Invoking External Application

Orb-Weaver provide mechanism to invoke class of another application, therefore it possible to access functionality of another application. The mechanism depending on application which will be invoked, each application issuer has their own public APIs and different kind of parameters, contact your client application for method to access their APIs.

Example :

```
var t = ci_load("issuer", "appname", "class");
t->method1("from application 1");
```

Invoking Terminal Framework

Orb-Weaver provide method to invoke method from framework installed on terminal device, these kind of invocation useful when terminal framework want to verified card application or accessing another 3rd party device/network. Once a terminal framework invoked by card application the runtime execution will be passed onto terminal until the invoked method execution finished or an exception occurred, the latter course will terminate both execution. There are two types of terminal invocation, either by alias or thorugh an instance similar with external application invocation. External invocation enabling card application to scale-up their capability while providing card application application provider with collaboration mechanism with terminal framework/application provider.

Invoke By Alias

Invoke by alias require toolkit framework to included as library, invoke by alias make use of underlying CAT-API manually by provide terminal interpreter with specified alias of function to be invoked, the total arguments of invoked function should are limited to only one argument, therefore you should use array/object to pass in-case you need more arguments.

Example :

```
var t = new toolkit();
t->dev_invoke("alias", [arg1, arg2]);                    //case for 2 arguments
```

Invoke By Instance

Invoke by instance doesn't require toolkit framework to be included, the invocation process similar with invoking external card application, by issuing vendor name, framework name and class name to be invoked, the returning function will return an instance of the specified class on terminal. Use normal invocation should work at this time, in-case the function not found on the terminal it will trigger an exception. Unlike “Invoke by Alias”, maximum total arguments of invoked function could up to 16 arguments, but all arguments are still subject to limitation of 240 bytes (which means total bytes of all arguments should no bigger than 240 bytes, as it is the maximum amount of bytes transferred during one CAT-transfer)

Example :

```
var t = ti_load("issuer", "framework", "class");
t->method1(arg1, arg2);                    //case for 2 arguments
```

Both ci_load and ti_load are only applicable for card application, invoking these APIs from terminal application would generate an exception.

Comments and Documentation

Organ Studio support several mnemonics for source code documentation, all can be embed within code comment for function documentation.

Example :

```
/*
@fn:print(text)
@desc:print text to display
@param: text, text to print
*/
```

```
	public function print(text) {
		tk_start(0x21, 0x01, STK_DEV_DISPLAY);
		tk_push(STK_TAG_TEXT_STRING, text);
		return tk_dispatch(STK_TAG_RESULT);
	}
```

Graphical User Interface APIs

Accessing direct Graphical User Interface (GUI) APIs only supported by device/terminal type hardware platform through specialized native APIs, accessing GUI APIs from card type hardware platform only can be done through standard toolkit APIs which is defined by ETSI102.223 or 3GPP11.14 standard, customization would be very limited to the specification defined.

Toolkit APIs

Toolkit APIs enabling card type hardware platform to send command to terminal type platform using ETSI102.223 (Card Application Toolkit) or 3GPP11.14 (SIM Application Toolkit) protocol, interpretation of each command are depend on Card Acceptance Device (CAD) command interpreter. It is not recommended to access Toolkit APIs directly from user application, instead use Toolkit library provided by online repository, accessing Toolkit APIs directly from user code would result in user code being less portable.

Example

```
public function print(text) {
	tk_start(0x21, 0x01, STK_DEV_DISPLAY);	//initialize proactive command buffer
	tk_push(STK_TAG_TEXT_STRING, text);	//set the text string
	return tk_dispatch(STK_TAG_RESULT);	//dispatch buffer, wait for specified tag
}
```

Direct GUI APIs

Direct GUI APIs are accessible as specialized native APIs on device/terminal type hardware platform, it enable advanced UI customization on the device such as custom GUI (either through user code by calling each APIs manually, or based on config data generated by GUI designer).

Example

```
public function print(text) {
	var win = ui_create_window();		//create new empty window
	win.create_label(text);			//create label on newly created window
	win.create_button(0, "OK");		//create button with text OK, returned ID=0
	win.create_button(1, "Cancel");		//create button with text cancel, returned ID=1
	var res = win.wait(0);			//wait until user action
	win.destroy();				//release window from memory
	return res;				//return the ID of clicked button
}
```

Network APIs

Network APIs only available on device/terminal type hardware platform, accessing Network APIs from card type hardware platform only can be done through Toolkit APIs. There are two type of mode for socket operation through Network APIs, Transmit Mode enable user application to send a payload to specified host on remote address and wait till host response, Listen Mode enable user application to wait until it receive a payload from specified client, which later can be processed on user defined callback function.

Transmit Mode

Opening a socket in transmit mode doesn't require any callback being defined (remote host location should be defined), program flow automatically blocked during receive sequence, if no response received until a given time, a timeout will occurred and null response will be returned.

Example

```
var conn = net_open("TCP", "host.com", 130);
if(conn != null) {
```

```
	conn.write("my payload");
	var response = conn.read();
	conn.close();
}
```

Listen Mode

Opening socket in listen mode require a processing callback being defined during socket creation, host parameter will be ignored, this callback will automatically executed each time a request being made to the device/terminal. You can use lambda expression to simplified the declaration process or assign the function name.

Example

```
var sock = net_open("TCP", null, 130, function(conn, payload) {
	var lines = payload.split("\n", 1);
	var res = "";
	for(var i=lines.count();i>0;i--) {
		res += lines[i-1] + "\n";
	}
	conn.write(res);		//use the socket provided on the argument to send response
	conn.close();
});
```

4. Orb-Weaver Virtual Machine

Orb-Weaver virtual machine based on stack machine, several features of virtual machine includes :

- String based virtual machine, all object instances allocated on transient memory and treated as string (LV format).
- Automatic garbage collector, for each instruction execution, virtual machine automatically collect any de-referenced instances, therefore the amount of memory usage can be as low as possible.
- Total 23 opcodes available, each instruction capable to process instance (LV format) operand.
- Integrated exception handler, when operation generate an exception, virtual machine automatically handle exception.

ARC (Automatic Reference Counting)

ARC used as mechanism to detect de-referenced object on heap by employing counter mechanism, for each object referencing the counter increased by one, and for each object de-referencing the counter decreased by one, when counter reach zero object will be released from heap.

User Data - Stack Data Layout

Orb-Weaver virtual machine based on stack machine which perform every operation and store it's result on stack, the depth of stack pointer depending on the depth of the operation executed. User data area also stored on heap which grows toward stack area for every function call, using pointer known as base pointer to maintain base area, when base pointer overlapped with stack pointer Orb-Weaver automatically generate stack overflow exception.

During function return user data area released by recalculating base pointer value back to the caller function, any returning operand would be pushed onto stack to be used for next operation on caller function, when function treated as procedure compiler should release the value on stack by popping it.

Supported Exception

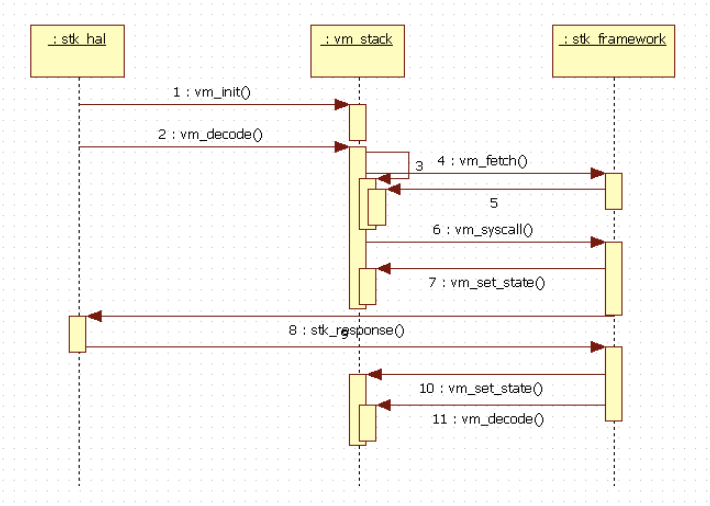
Orb-Weaver support several exception, each with it's own severity level, depending on severity level an exception can be handled or unhandled. For handled exception virtual machine automatically terminate it's execution and aborting transaction, while unhandled exception could cause the system run indefinitely. The likeliness of unhandled exception cause by the use to unstable Native APIs are very high in these case, therefore user code didn't have to worry about the exception.

Code	Definition	Description	Severity	Common cause
0	VX_SYSTEM_EXCEPTION	Unknown error on runtime invoked by unexpected toolkit result/operation	High	These type of error occurred when system cannot determined the cause, if this error occurred try contacting our team at support@orbleaf.com
1	VX_UNIMPLEMENTED_API	Native APIs not implemented on current environment	Medium	These type of error occurred when a program executed on older virtual machine, if this error occurred try contacting our team at support@orbleaf.com
2	VX_UNKNOWN_INSTRUCTION	Unknown instruction detected, caused by different machine-compiler version	Medium	These type of error occurred when a program executed on older virtual machine, if this error occurred try contacting our team at support@orbleaf.com
3	VX_STACK_OVERFLOW	Stack area overlapped with user data area	Medium	These type of error occurred when use defined function allocating too many local variable or during recursion, try reduce the amount of variable declaration on specified function.
4	VX_INSUFFICIENT_HEAP	Current memory insufficient for operation, try null-ing some unused variable to force garbage collector to run	High	These type of error occurred when system doesn't have anymore heap to do the processing, try freeing some variable by assigning them with null value
5	VX_STACK_UNDERFLOW	Stack underflow possibly caused by buffer overrun on another application/framework	Fatal	These type of error rarely occurred on clean compiler generation, if this error occurred try contacting our team at support@orbleaf.com
6	VX_OUT_OF_BOUNDS	An operation failure caused by index out of ranged	Medium	These type of error mostly caused from concatenating several variables altogether generating a new resultant variable longer than maximum bounds allocated for each variable
7	VX_UNRESOLVED_CLASS	No class with specified name existed	Medium	Make sure the all project classes has been correctly installed and compiled successfully.
8	VX_UNRESOLVED_METHOD	No method with specified name existed	Medium	this might be caused by overloading, try checking number of arguments passed during invocation
9	VX_ARGUMENT_MISMATCH	Total number of argument during function call mismatch	High	These type of error occurred when invoking a function through lambda expression, make sure that the argument passed matched with invoked function
10	VX_INVALID_CONTEXT	An API invoked from invalid context	High	Make sure that the context var is suitable with invoked API, example encrypt and decrypt API only worked on crypto context

11	VX_DIVIDE_BY_ZERO	A mathematical operation divide by zero exception	High	Make sure that zero is not used as divisor

5. Orb-Weaver Features And Limitations

Orb-Weaver provide integration between native and bytecodes, card-terminal mechanism seamlessly, safe and scalable. Using high level language scope on compiler level any toolkit rookie could code toolkit application script at the flip of one hand, which makes Orb-Weaver highly recommended for toolkit application development.



Example of Activity Diagram during Request-Response communication between card and terminal

Orb-Weaver doesn't support global variable declaration, as it might increase the possibility of memory leakage when the variables didn't released even when the system is in inactive state. In order to perform shared variable between function, user might consider using persistent storage APIs as alternative.

Programming Best Practice

The workspace stack for Orb-Weaver is quite small, because the program is being run on small machine with constrained resource and processing power, therefore local variable allocation should be taken into account when programming with Orb-Weaver, one might better reuse variable than allocating a new one. The maximum variable length must less than 255 bytes, keep in mind that processing large size variable, would exhaust memory and might cause an exception, cause the limitation of internal memory space which is only around 1K bytes.

Here are several tips on Orb-Weaver programming:

1) Avoid processing large data, keep your data at max 100 bytes, or less than 100 character, processing large data would exhaust heap usage.
2) Keep the bare maximum size of variable within 240 bytes, lower bytes would better for heap process, Orb-Weaver cannot process any variable longer than 255 bytes.
3) Size of response from server during network request should be taken into account when processing with Orb-Weaver, larger response than 240 bytes (payload only) would not be processed by Orb-Weaver.
4) Try to avoid declaring too many local variable within one function, and keep your function depth at bare minimum, recursion or deep function call would overflowed the stack
5) When processing remote request such as HTTP, Keep It Small and Simple (KISS), do more process on the back-end server instead of the Orb-Weaver, constrained resources is not a good platform for advanced calculation.
6) Do not include too many libraries on your project, too many libraries would increase code size and loading time.

Example (Bad Practice):

```
public function my_func() {
        var a = 1. b = 2, c = 3, d, e;
        d = b + c;
        e = a + b + d;
}
```

Example (Best Practice):

reuse variable as possible when necessary.

```
public function my_func() {
        var a = 1. b = 2, c = 3;
        b = b + c;
        a = a + b;
}
```

6. Card APIs

Card Native APIs

Orb-Weaver support several native APIs, it's one of key feature of Orb-Weaver that support integration between native APIs and interpreted bytecodes seamlessly. If an API didn't exist during API call virtual machine automatically generate an exception and terminal application automatically terminate application.

ID	Function Name	Return Value	Arguments	Description
Global Variable APIs				
0	**this**	context	none	Get current instance context
Orb-Weaver Object APIs				
14	**arg_create**	object	key Key name in string value Either object, string or binary	Create a new key-value pair for object operation
15	**arg_object**	object	x1, x2,...x[n] Variadic arguments containing any Orb-Weaver objects	Create an Orb-Weaver object, use curly bracket ({}) to invoke this API directly from compiler
16	**arg_array**	object	x1, x2,...x[n] Variadic arguments containing any Orb-Weaver objects	Create an Orb-Weaver array, use square bracket ([]) to invoke this API directly from compiler
Persistent Storage APIs				
30	**data**	context	none	Get current persistent storage context
Bit operation APIs				
128	**check_bit**	0/1	binary data Binary data to be checked index Index of bit to be checked start from 0	Check bit at specified index of variable for binary value
129	**set_bit**	none	binary data Binary data to be set index Index of bit to be set start from 0	Set bit at specified index of variable
130	**clear_bit**	none	binary data Binary data to be cleared index Index of bit to be cleared start from 0	Clear bit at specified index of variable
ISO8583 APIs				
37	**iso_create**	ISO8583 context	MTI Message Type Indicator, 4 digit string	Create new ISO8583 message
38	**iso_push**	none	ISO8583 context Message context from iso_create element Element to push, in string format	Push an element to ISO8583 message string
39	**iso_get**	string	ISO8583 string ISO8583 string tag Tag of element to get	Get an element from ISO8583 message string
Cryptography APIs				
136	**cr_create**	Crypto context	algo “DES” DES “DES2” 3 DES with 2 keys “DES3” 3 DES with 3 keys “AES” AES mode “CBC”/”ECB” cipher mode key Key for operation, length depending on algorithm, automatically padded with zero	Create new crypto context for enciphering/deciphering operation, case of DES, key length will determined the algorithm used
139	**cr_random**	Random bytes	length Length of random bytes	Generate random value with specified length
Proactive Command APIs				
160	**tk_start**	none	cmd Proactive command tag qualifier Command qualifier target Target device	Start proactive command builder session, use arguments to create proactive command header
161	**tk_dispatch**	variable	tag Proactive command tag to be read	Dispatch current proactive command session to device, and wait for specific tag as return value
162	**tk_push**	none	tag Tag to be pushed to proactive current command session data Data to be pushed, length automatically adjusted	Push a TLV to current proactive command session before being dispatched
164	**tk_result**	numeric	none	Get last toolkit result from terminal
External Invocation				
32	**ci_load**	instance	issuer Issuer name in string name Name of application to invoke in string class Class to invoke in string for instance creation	Invoke an instance from external class of another application
33	**ti_load**	instance	issuer Issuer name in string name Name of application	Invoke an instance from terminal framework

			class	to invoke in string Class to invoke in string for instance creation	

Card Object APIs

Another way to interact with Orb-Weaver native APIs, is using Object methods, each object can is represented as binary data, either string, instance or numerical which has a combined methods that can be accessed using dot (.) symbolic operation. If an API didn't exist during API call virtual machine automatically generate an exception and terminal application automatically terminate application.

ID	Function Name	Applied To	Return Value	Arguments	Description
Variable APIs					
1	**store**	context	context	key Key to find value New value	Store a value to context variable either persistent through data() API or global through this() API
2	**load**	context	any	key Key to find	Get a value from context variable either persistent through data() API or global through this() API
3	**delete**	context	context	key Key to find	Delete a value from context variable either persistent through data() API or global through this() API
String Manipulation					
4	**index_of**	string	numeric	pattern Pattern string to be searched offset Start offset to search	Search any occurrence of specific pattern from an offset, when offset didn't specified 0 will be used, returning position of matched pattern
5	**replace**	string	string	pattern Pattern string to be replaced value New value which will replace pattern	Replace specific pattern of string with new value string
6	**substr**	string	string	offset offset length length	
7	**to_bytes**	string	bytes	none	Convert current object to bytes array
Handle operation					
9	**close**	handle	none	none	Close current handle
10	**read**	handle	bytes	length Length of bytes to read	Read some bytes at specified length from current handle
11	**write**	handle	none	bytes Bytes to write	Write a sequence of bytes to current handle
12	**seek**	handle	none	offset Offset to seek	Set current handle read/write offset
Orb-Weaver Object APIs					
13	**count**	array	number	none	Return number of object/array within array
17	**at**	array	object	index Index of specified object at array	Get an object from specified array at specified index
18	**get**	object	object	key Key name of specified key-value pair at object	Get an object from with specified key
19	**to_json**	object	string	none	Convert to json string
20	**from_json**	string	object	none	Convert from json string
21	**add**	array/object	object	Any array/object to be added	Add new array/object to collection
22	**set**	array/object	none	Key Key to find value New value	Change specified key value
23	**remove**	array/object	none	Key Key to find value New value	Remove specified key value from collection, change key value to number in-case of array
25	**split**	string	array	pattern String pattern to match mode 0 Normal mode 1 Remove empty entries	Split a string based on matching pattern
Crypto Operation APIs					
137	**encrypt**	Crypto context	Cipher text	value Plain text to encrypt	Encrypt a plain text to cipher text, crypto context which the API invoked will determined the algorithm being used
138	**decrypt**	Crypto context	Plain text	value Cipher text to decode	Decrypt a cipher text to plain text, crypto context which the API invoked will determined the algorithm being used
140	**hash**	Binary	Binary	string Any of string value below determine the algorithm being used for hash generation md5 md5 hash sha1 Sha128 hash sha256 Sha256 hash crc32 crc32 lrc lrc	Generate a hash value for specified variable (bytes), it will return the hash in binary, use to_hex() to convert it to hexstring value for string representation
Conversion APIs					
131	**to_hex**	binary	hexstring	binary data Convert binary data to corresponding hexstring value	Convert a binary value to it's hexstring value
132	**from_hex**	hexstring	binary	hexstring Convert hexstring value to correponding binary data	Convert a hexstring value to it's binary value
133	**to_dec**	binary	decimal	binary data Convert binary value to corresponding decimal value, only 2 bytes perconversion maximum	Convert a binary value to decimal digit
134	**to_b64**	any	base64	none	Encode to base64 string

135	**from_b64**	base64	any	none	Decode base64 string

7. Terminal APIs

Terminal Framework or Device Entity are later added to increase scalability of Orb-Weaver application to access external devices, it can also act to provide two factor authentication between terminal and card application. Unlike normal terminal application, terminal framework fully working as extension of card application therefore it's execution are triggered by card application. Integration between terminal framework and card application are done in seamless way so that programmer didn't need to understand of low-level process involved between card and terminal.

There are several benefits of processing data on terminal, unlike card application where each variable are limited to 255 bytes at max, the maximum bytes for variable on terminal framework could be sized up to 64K (depending on heap available by operating system, though most system actually didn't posses heap that much during runtime), therefore terminal framework could process larger data than card application does.

In-order to maintain conformity between terminal framework and card application, Orb-Weaver scripting language employs for framework development, but with several differences on some APIs.

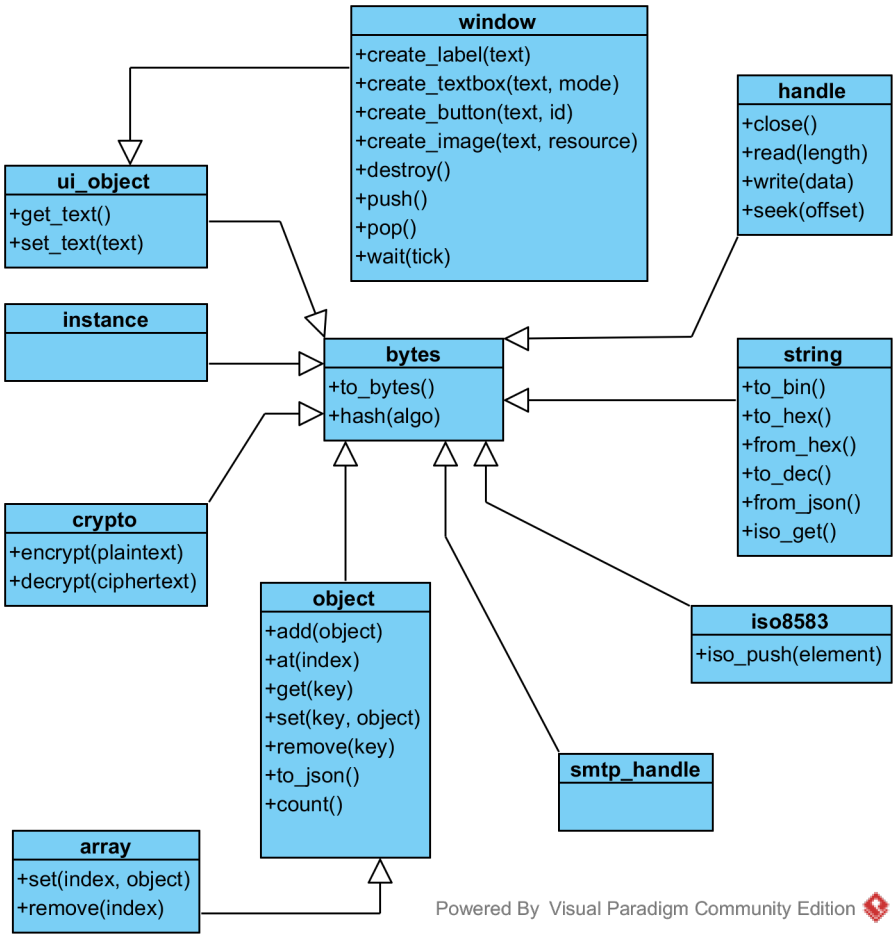


Fig 2.1 Terminal Framework Object

Since version 1.5, Terminal Orb-Weaver runtime added with program.main() capability to run terminal application as standalone application without any external trigger from event or card application invocation (like on previous design). This program.main() are actually a type of event that are trigger upon user action, by clicking application icon on main menu. Organ Studio IDE handle all the application configuration including application icon and supported capabilities.

Terminal Native APIs

Orb-Weaver support several native APIs, it's one of key feature of Orb-Weaver that support integration between native APIs and interpreted bytecodes seamlessly. If an API didn't exist during API call virtual machine automatically generate an exception and terminal application automatically terminate application.

ID	Function Name	Return Value	Arguments	Description
Global Variable APIs				
0	**this**	context	none	Get current instance context
Orb-Weaver Object APIs				
14	**arg_create**	object	key Key name in string	Create a new key-value pair for object operation

			value	Either object, string or binary	
15	**arg_object**	object	x1, x2,...x[n]	Variadic arguments containing any Orb-Weaver objects	Create an Orb-Weaver object, use alternative curly bracket
16	**arg_array**	object	x1, x2,...x[n]	Variadic arguments containing any Orb-Weaver objects	Create an Orb-Weaver array, use alternative square bracket
Persistent Storage APIs					
30	**data**	context	none		Get current persistent storage context
Bit operation APIs					
128	**check_bit**	0/1	binary data index	Binary data to be checked Index of bit to be checked start from 0	Check bit at specified index of variable for binary value
129	**set_bit**	none	binary data index	Binary data to be set Index of bit to be set start from 0	Set bit at specified index of variable
130	**clear_bit**	none	binary data index	Binary data to be cleared Index of bit to be cleared start from 0	Clear bit at specified index of variable
ISO8583 APIs					
37	**iso_create**	ISO8583 context	MTI	Message Type Indicator, 4 digit string	Create new ISO8583 message
38	**iso_push**	none	ISO8583 context element	Message context from iso_create Element to push, in string format	Push an element to ISO8583 message string
39	**iso_get**	string	ISO8583 string tag	ISO8583 string Tag of element to get	Get an element from ISO8583 message string
Cryptography APIs					
136	**cr_create**	Crypto context	algo mode key	“DES” DES “DES2” 3 DES with 2 keys “DES3” 3 DES with 3 keys “AES” AES “CBC”/”ECB” cipher mode Key for operation, length depending on algorithm, automatically padded with zero	Create new crypto context for enciphering/deciphering operation, case of DES, key length will determined the algorithm used
139	**cr_random**	Random bytes	length	Length of random bytes	Generate random value with specified length
Network APIs					
144	**net_open**	handle	type address port callback	Either “TCP” or “UDP” string IP address in string format Port number from 0-65535 Callback function with at max two parameters that will be called upon data received, setting callback function automatically set socket in listen mode	Open a network socket for socket operation, currently only support TCP or UDP protocol
145	**net_transmit**	bytes	URI parameters method type headers payload	Requested URI, please specify including the protocol http://, https:// or coap:// Additional parameters in array of key-value pairs {key1:value1, key2:value2, keyN:valueN}, set to null in-case additional payload used Either “GET” or “POST” for underlying request method, null for default GET Either “TCP” or “UDP” for underlying transport protocol, null for default TCP Either string sequence with “\r\n” delimiters or array of string [“header1”, “header2”], null if not used Additional payload for POST request, null if not used	Transmit a supported request, either in form of HTTP(S) or COAP, header, payload, and underlying protocol can be set from parameters
146	**net_list**	array	none		List all available NetBIOS name on cache, useful for discovering local P2P nodes on the same network
148	**net_mail_create**	smtp_handle	server username password port	Server DNS name or IP address Username for mail account Password for mail account Port for SMTP server, default 25	Create a new mail context for net_mail_send
149	**net_mail_send**	none	smtp_handle to subject message from	Handle created by net_mail_create Destination mail address Message subject Message body Source mail address	Send mail to destination recipient with specified subject, message body
GUI APIs					

160	**ui_alert**	numeric	title Title string text Text message on alert box mode Button mode 1 OK 2 Cancel 3 OK + Cancel icon Icon to display 0 info 1 warning 2 error 3 busy 4 network 5 secure	Display an alert box for user confirmation
161	**ui_create_window**	window	config_data UI configuration data generated by GUI designer	Create a new custom window, see Terminal Object APIs for more detail of supported APIs
169	**ui_display_text**	numeric	title Title string text Text message on alert box	Display a text confirmation, returning 0 on user confirmation
170	**ui_get_input**	string	title Title string text Text message on alert box mode textbox mode 0 alphanumeric 1 numeric 4 Masked alphanumeric (Password) 5 Masked numeric (PIN) Special mode 0x84 IP address 0x85 Email address	Display a user input, mode determined type of keyboard and textbox features
171	**ui_select_item**	numeric	title Title string list Array of string, example [“menu1”, “menu2”]	Display a list-item to user, returning index of selected item on user confirmation
IO APIs				
192	**com_open**	handle	port Port number, default 0 baud Baud rate, 9600, 19200, etc parity Parity, 0=even, 1=odd stop Stop bit 0 1 stop bit 1 1.5 stop bit 2 2 stop bit	Open a com port for listening/transmiting, the communication protocol based on RS232
196	**port_open**	handle	pin Pin to open 0 Pin 0 1 Pin 1 2 Pin 2 frequency Frequency in Hertz for clock generation (in-case for PWM output), 0 for logic type pin output, value should be between 0-1000 Duty-cycle Duty cycle for PWM, value should be between 0-100 (percent)	Open a port either for listening or output, there are two type of output, either digital logic or PWM output (which frequency and duty cycle can be set from parameters)
PICC APIs				
200	**picc_open**	handle	none	Open active PICC connection, currently only support Mifare Classic 1K/4K and Ultra-Light
201	**picc_auth**	boolean	handle Picc handle block Block number to authenticate key_id Key identifier for Mifare authentication 0 Key A 1 Key B key 12 digit hexstring key for mifate authentication	Authenticate a Mifare sector for read/write operation
202	**picc_transmit**	bytes	handle Picc handle command APDU command to be send to current PICC connection, including header and payload	Transmit an APDU to current active PICC connection
Bluetooth APIs				
208	**ble_open**	handle	svc_uuid Service uuid in binary format chr_uud Characteristic uuid in binary format	Open a bluetooth connection, bluetooth device should already been connected or the API would try re-connecting
System APIs				
240	**os_wait**	none	tick Number of tick in miliseconds	Wait for specified amount of time

Terminal Object APIs

Another way to interact with Orb-Weaver native APIs, is using Object methods, each object can is represented as binary data, either string, instance or numerical which has a combined methods that can be accessed using dot (.) symbolic operation. If an API didn't exist during API call virtual machine automatically generate an exception and terminal application automatically terminate application.

ID	Function Name	Applied To	Return Value	Arguments	Description
Variable APIs					
1	**store**	context	context	key Key to find value New value	Store a value to context variable either persistent through data() API or global through this() API
2	**load**	context	any	key Key to find	Get a value from context variable either persistent through data() API or global through this() API
3	**delete**	context	context	key Key to find	Delete a value from context variable either persistent through data() API or global through this() API
String Manipulation					
4	**index_of**	string	numeric	pattern Pattern string to be searched offset Start offset to search	Search any occurrence of specific pattern from an offset, when offset didn't specified 0 will be used, returning position of matched pattern
5	**replace**	string	string	pattern Pattern string to be replaced value New value which will replace pattern	Replace specific pattern of string with new value string
6	**substr**	string	string	offset offset length length	
7	**to_bytes**	string	bytes	none	Convert current object to bytes array
Handle operation					
9	**close**	handle	none	none	Close current handle
10	**read**	handle	bytes	length Length of bytes to read	Read some bytes at specified length from current handle
11	**write**	handle	none	bytes Bytes to write	Write a sequence of bytes to current handle
12	**seek**	handle	none	offset Offset to seek	Set current handle read/write offset
Orb-Weaver Object APIs					
13	**count**	array	number	none	Return number of object/array within array
17	**at**	array	object	index Index of specified object at array	Get an object from specified array at specified index
18	**get**	object	object	key Key name of specified key-value pair at object	Get an object from with specified key
19	**to_json**	object	string	none	Convert to json string
20	**from_json**	string	object	none	Convert from json string
21	**add**	array/object	object	Any array/object to be added	Add new array/object to collection
22	**set**	array/object	none	Key Key to find value New value	Change specified key value
23	**remove**	array/object	none	Key Key to find value New value	Remove specified key value from collection, change key value to number in-case of array
25	**split**	string	array	pattern String pattern to match mode 0 Normal mode 1 Remove empty entries	Split a string based on matching pattern
Crypto Operation APIs					
137	**encrypt**	Crypto context	Cipher text	value Plain text to encrypt	Encrypt a plain text to cipher text crypto context which the API invoked will determined the algorithm being used
138	**decrypt**	Crypto context	Plain text	value Cipher text to decode	Decrypt a cipher text to plain text crypto context which the API invoked will determined the algorithm being used
140	**hash**	Binary	Binary	string Any of string value below determine the algorithm being used for hash generation md5 md5 hash sha1 Sha128 hash sha256 Sha256 hash crc32 crc32 lrc lrc	Generate a hash value for specified variable (bytes), it will return the hash in binary, use to_hex() to convert it to hexstring value for string representation
Conversion APIs					
131	**to_hex**	binary	hexstring	binary data Convert binary data to corresponding hexstring value	Convert a binary value to it's hexstring value
132	**from_hex**	hexstring	binary	hexstring Convert hexstring value to correponding binary data	Convert a hexstring value to it's binary value
133	**to_dec**	binary	decimal	binary data Convert binary value to corresponding decimal value, only 2 bytes perconversion maximum	Convert a binary value to decimal digit

134	**to_b64**	any	base64	none	Encode to base64 string
135	**from_b64**	base64	any	none	Decode base64 string
150	**to_html**	string	string	none	Convert a normal string to it's HTTP escape representation
151	**from_html**	string	string	none	Convert HTTP escaped string to normal string representation
Window APIs					
162	**destroy**	window	none		Destroy current window from memory, must be popped first from display stack
163	**create_label**	window	ui_object	text Text to display mode 0 Normal text (default) 1 Small text 2 QR code	Create a new label, mode is optional and can be used to display specific label representation
164	**create_button**	window	ui_object	text Text to display id Numeric Identifier associated with current object	Create a new button with associated identifer
165	**create_textbox**	window	ui_object	text Current textbox text mode textbox mode 0 alphanumeric 1 numeric 4 Masked alphanumeric (Password) 5 Masked numeric (PIN) Special mode 0x84 IP address 0x85 Email address	Create a new textbox with specified mode and default text
166	**create_image**	window	ui_object	text Text to display resource An image resource loaded etiher from file, network, card or other devices, currently only support PNG and JPG	Create an image loader at current window, maximum image that can be handled should at max 64px width and 64 px height with size no bigger than 4KB
168	**wait**	window	numeric	tick Wait for specified tick before window executing next instruction, use 0 to wait until user click a button, it will return a numeric identifier associated with clicked button	Display the current window from display stack and wait for specified amount of time or until user event, wait must be called in-order to system to render the generated window
172	**create_item**	window	ui_object	text Text to display id Numeric Identifier associated with current object	Create a list-item with associated identifier
175	**find**	window	ui_object	name Name of an ui_object to find	Find a child object with specific name
176	**get_text**	ui_object	string	none	Get an ui_object text, such as textbox
177	**set_text**	ui_object	none	text New text for current ui_object	Set an ui_object text
179	**set_image**	ui_object	none	image_data A resource data containing image binary	Set an image for current ui_object, only support JPG and PNG for displaying image, maximum size should no bigger than 64x64 pixel or maximum allocated object (4K)
190	**push**	window	none	none	Push window to display stack
191	**pop**	window	none	none	Pop window from display stack