



JSON-RPC C Developer's Guide

Version 1.0
January 15, 2008

Bridge Interactive Group
6305 Brickworks Circle
Atlanta, GA 30307
www.big-llc.com

Table of Contents

1.0 – Building The Library

2.0 – JSON RPC C Reference

3.0 – Adding Handlers and Processing Requests

4.0 – Adding Events



1.0 – Building The Library

The library was originally built and tested on Windows XP Pro under Cygwin. At the time of writing it has not been built or tested on any other systems. If you have any problems and/or solutions building under other systems we would most appreciate of your feedback.

You will need to get the latest download from <http://www.big-llc.com/downloads/jsonrpc-c> and unzip it.

Inside the build structure there are two directories:

json	Supplied for your convenience is Metaparadigm's JSON C implementation.
jsonrpc	The JSON-RPC C library.

You will need to build the JSON C implementation first.

```
> cd json  
> make
```

If this builds correctly you can then proceed to build the jsonrpc library

```
> cd ../jsonrpc  
> make
```

If this builds correctly you will have successfully built the JSON-RPC C library. The libraries can be found in the lib directories of each module:

```
json/libjson.a  
jsonrpc/libjsonrpc.a
```

2.0 – JSON-RPC C Reference

Minimal documentation for [Metaparadigm's JSON C implementation](#) can be found [here](#).

Below is a short reference to the JSON-RPC library API.



```

/**
 * Prototype for user defined method handlers.
 * the request object should be returned with
 * only the "result" key and the user object.
 */
typedef void (*jsonrpc_method)(
    struct json_object *request,
    struct json_object *response);
/**
 * Register a method handler.
 * @param name the lookup name of the method such as myobj.get
 * @param method pointer to method handling requests to this name.
 */
extern void jsonrpc_add_method(
    char* name,
    jsonrpc_method method);
/**
 * Set the number of events the bus should hold before old
 * events start getting discarded.
 * @param size the size of the event list.
 */
extern void jsonrpc_set_event_list_size(int size);
/**
 * Add an event to the event bus. The event type is an arbitrary
 * name that clients can switch against.
 * @param eventType the event type name.
 * @param eventData the data related to the event to be
 * passed to client.
 *
 */
extern void jsonrpc_add_event(
    char* eventType,
    struct json_object *eventData);
/*
 * Service a request. This places the onus on the caller to create
 * the request and response objects, convert the response to a string
 * and free up the JSON objects after use.
 * @param request a JSON object containing the request

```



```

*           parameters and method
* @param response the JSON object to contain the response to be
*           used by the actual method called.
*/
extern void jsonrpc_service(
    struct json_object *request,
    struct json_object *response);
/**
* Process a JSON format request and returns a JSON format response.
* Typically the web server query is passed in and the response
* submitted to the client is the response from this method. The
* response must be freed using free() after it has been used.
* @param request a JSON request string.
* @return a JSON formatted response. This must be freed after use.
*/
extern char* jsonrpc_process(
    char* request);

```

3.0 – Adding Handlers and Processing Requests

First we need to register handlers with the json-rpc system. You only need to do this once on initialising your application. The methods are expected to have the following prototype:

```

void (*jsonrpc_method)(
    struct json_object *request,
    struct json_object *response);

```

We add the handlers as follows:

```

jsonrpc_add_method("phonebook.all", pb_get_all_entries);
jsonrpc_add_method("phonebook.add", pb_add_entry);
jsonrpc_add_method("phonebook.find", pb_get_entry);

```

This assumes that we have three methods called `pb_get_all_entries`, `pb_add_entry` and `pb_get_entry` which will be respectively called from your javascript using `jsonrpc.phonebook.all()`, `jsonrpc.phonebook.add()` and `jsonrpc.phonebook.find()`. Where `jsonrpc` is the JSONRPC javascript client and

the javascript methods may take parameters.

The handlers must be able to process the requests and format the responses. For example the pb_get_entry method may look something like this:

```
void pb_get_entry(
    struct json_object *request,
    struct json_object *response)
{
    /* get the parameters in this call */
    struct json_object *params =
        json_object_object_get(request, "params");

    /* assume the first and only parameter is the name */
    char* name = json_object_get_string(
        json_object_array_get_idx(params, 0));

    /* some methods to get the phone and address */
    /* for the name */
    char* phone = pb_get_phonenumber(name);
    char* address = pb_get_address(name);

    /* create a new object to send the results in */
    struct json_object* entry = json_object_new_object();

    /* add the name to it */
    json_object_object_add(
        entry, "name", json_object_new_string(name));

    /* add the phone number to it */
    json_object_object_add(
        entry, "phone", json_object_new_string(phone));

    /* add the address to it */
    json_object_object_add(
        entry, "address", json_object_new_string(address));

    /* add the entry to the response object as the result */
    /* of the method call */
}
```



```
    json_object_object_add(response, "result", entry);
}
```

Obviously this example is trivial. But if a call is made to `jsonrpc.phonebook.find('Smith')`, the server that serves this request would pass it to the `jsonrpc` lib and it would process the request and pass it to this method. This method then finds the entry for Smith and constructs a JSON response with the name, phone number and address of the recipient.

The final bit of the puzzle is how would you handle requests from your webserver. Well it should look something like this:

```
/* create the response for a query */
/* the query is that produced by */
/* the javascript JSONRPC client */
char* response = jsonrpc_process(query);

/* Send the response back to the client */
/* However you web server handles it */

/* Free up the response */
free(response);
```

For an example of how to hook this all up in the GoAhead WebServer please read this document - [GoAhead Integration](#).

4.0 – Adding Events

We've also added an event bus which can be accessed through the system methods.

To add an event, simply create a JSON C object and add it to the event queue with the type of event. The type is simply a string that identifies the type of event it is.

For example we could have an event when new entries are added to the phone book

```
void pb_get_entry(
    struct json_object *request,
    struct json_object *response)
```



```

{
  /* Do the stuff to add the new entry to the
  * to the phone book directory
  */
  char* name = ...
  char* phone = ...
  char* address = ...

  /* Create the event object */
  struct json_object* eventData = json_object_new_object();

  /* add the name to it */
  json_object_object_add(
    eventData , "name", json_object_new_string(name));

  /* add the phone number to it */
  json_object_object_add(
    eventData , "phone", json_object_new_string(phone));

  /* add the address to it */
  json_object_object_add(
    eventData , "address", json_object_new_string(address));

  jsonrpc_add_event("NewEntry", eventData);
}

```

The client can query all the events that have occurred since it was last called by calling:

```

/* lastEventId should be the last event id received by
* this client or 0 to fetch the complete event list.
*/

```

```

var events = jsonrpc.system.events(lastEventId);

```

This will return an array of events. Each event has four properties:

id	The id of the event. The last of these should be retained for subsequent calls to fetch latest events, otherwise all events will be resent.
time	The time the event was added to the servers event queue.
type	The event type. An arbitrary string set by the event bus injector.
data	The proprietary data of the event.



We already have a javascript file that can handle the polling of events for you. This can be found in our downloads section - [jsonrpc_event.js](#).

Please let us know if we have missed anything out. Or if you need more help.

