Hard Mode: Roll Your Own Client

Discord's Rich Presence feature is designed as an obfuscated addition to our existing RPC infrastructure. The standalone library and header files make it easy for any dev to drop it into their game.

Our library communicates with Discord over the local Discord RPC socket. We've already done the work in connecting properly, handling disconnects and reconnects, and other RPC intracacies, but those who have done this implementation for our private alpha Voice and Chat SDK can simply make use of the new RPC commands and events to implement Rich Presence.

Hark! A warning!

By committing to an RPC-only integration, you decide to forego the work our library and header file have done for you in the way of error handling, state storage, disconnecting and reconnecting, and other quality of life abstractions. While simply implementing the new RPC command and events will enable Rich Presence for your game, we highly suggest that you do your best to mimic the functionality of the SDK the most that you can. It ensure not only code quality on your part, but also an excellent experience on the part of your players.

Application Protocol Registration

One thing that cannot be explicitly done over RPC is registering an application protocol for your game. If you choose to do an RPC-only implementation, you will have to register your application protocol yourself in the format of `discord-[your_app_id]://`. You can use `Discord_Register()` as a good(?) example of how to properly register an application protocol for use with Discord. For OSX and Linux it is probably simpler to handle the protocol registration as part of your install/packaging.

New RPC Command

The new RPC command for Rich Presence is `SET_ACTIVITY`. The fields are similar to what is outlined in the SDK; we've combined similar fields into objects for the sake of less data on the wire.

The one major difference is the `party.size` field. It is an array with a size of two. The first element is the current party size, `partySize` from the main documentation. The second element is the maximum party size, `partyMax` from the main documentation.

Below is a full example of a `SET_ACTIVITY` command. Field restrictions like size are the same as outlined in the main documentation.

```
{
  "cmd": "SET_ACTIVITY",
  "args": {
    "pid": 9999,  // Your application's process id - required field
    "activity": {
      "state": "In a Group",
      "details": "Competitive | In a Match",
      "timestamps": {
```

```
        "start": time(nullptr),
        "end": time(nullptr) + ((60 * 5) + 23)
      },
      "assets": {
        "large_image": "numbani_map",
        "large_text": "Numbani",
        "small_image": "pharah_profile",
        "small_text": "Pharah"
      },
      "party": {
        "id": GameEngine.GetPartyId(),
        "size": [3, 6]
      },
      "secrets": {
        "join": "025ed05c71f639de8bfaa0d679d7c94b2fdce12f",
        "spectate": "e7eb30d2ee025ed05c71ea495f770b76454ee4e0",
        "match": "4b2fdce12f639de8bfa7e3591b71a0d679d7c93f"
      },
      "instance": true
    }
  },
  "nonce": "647d814a-4cf8-4fbb-948f-898abd24f55b"
}
```

New RPC Events

The three new RPC events for Rich Presence power the ability to join and spectate your friends' games.

First is the `ACTIVITY_JOIN` event:

```
{
  "cmd": "DISPATCH",
  "data": {
    "secret": "025ed05c71f639de8bfaa0d679d7c94b2fdce12f"
  },
  "evt": "ACTIVITY_JOIN"
}
```

Second is the `ACTIVITY_SPECTATE` event:

```
{
  "cmd": "DISPATCH",
  "data": {
    "secret": "e7eb30d2ee025ed05c71ea495f770b76454ee4e0"
  },
  "evt": "ACTIVITY_SPECTATE"
}
```

And third is the `ACTIVITY_JOIN_REQUEST` event:

```
{
  "cmd": "DISPATCH",
```

```
  "data": {
    "user": {
      "id": "53908232506183680",
      "username": "Mason",
      "discriminator": "1337",
      "avatar": "a_bab14f271d565501444b2ca3be944b25"
    }
  },
  "evt": "ACTIVITY_JOIN_REQUEST"
}
```

In order to receive these events, you need to subscribe to them like so:

```
{
    "nonce": "be9a6de3-31d0-4767-a8e9-4818c5690015",
    "evt": "ACTIVITY_JOIN",
    "cmd": "SUBSCRIBE"
}
```

```
{
    "nonce": "ae9qdde3-31d0-8989-a8e9-dnakwy174he",
    "evt": "ACTIVITY_SPECTATE",
    "cmd": "SUBSCRIBE"
}
```

```
{
    "nonce": "5dc0c062-98c6-47a0-8922-bbb52e9d6afa",
    "evt": "ACTIVITY_JOIN_REQUEST",
    "cmd": "SUBSCRIBE"
}
```

To unsubscribe from these events, resend with the command `UNSUBSCRIBE`

Responding

A discord user will request access to the game. If the ACTIVITY_JOIN_REQUEST has been subscribed too, the ACTIVITY_JOIN_REQUEST event will be sent to the host's game. Accept it with following model:

```
{
    "nonce": "5dc0c062-98c6-47a0-8922-15aerg126",
    "cmd": "SEND_ACTIVITY_JOIN_INVITE",
    "args":
    {
        "user_id": "53908232506183680"
    }
}
```

To reject the request, use `CLOSE_ACTIVITY_REQUEST`:

```
{
    "nonce": "5dc0c062-98c6-47a0-8922-dasg256eafg",
```

```
    "cmd": "CLOSE_ACTIVITY_REQUEST",
    "args":
    {
        "user_id": "53908232506183680"
    }
}
```

Notes

Here are just some quick notes to help with some common troubleshooting problems.

- IPC will echo back every command you send as a response. Use this as a lock-step feature to avoid flooding messages. Can be used to validate messages such as the Presence or Subscribes.
- The pipe expects for frames to be written in a single byte array. You cannot do multiple `stream.Write(opcode);` `stream.Write(length);` as it will break the pipe. Instead create a buffer, write the data to the buffer, then send the entire buffer to the stream.
- Discord can be on any pipe ranging from `discord-ipc-0` to `discord-ipc-9`. It is a good idea to try and connect to each one and keeping the first one you connect too. For multiple clients (eg Discord and Canary), you might want to add a feature to manually select the pipe so you can more easily debug the application.
- All enums are `lower_snake_case`.
- The opcode and length in the header are `Little Endian Unsigned Integers (32bits)`. In some languages, you must convert them as they can be architecture specific.
- Discord Rich Presence How-To contains a lot of the information this document doesn't. For example, it will tell you about the response payload.
- In the documentation, DISCORD_REPLY_IGNORE is just implemented the same as DISCORD_REPLY_NO.
- You can test the Join / Spectate feature by enabling them in your profile and whitelisting a test account. Use Canary to run 2 accounts on the same machine.