Design: Paragon Messaging v1.0

Overview

Extend the synergy between apps on the GS desktop by providing a desktop messaging platform as a simple and easy solution for client side (desktop) inter-app communication:

- Paragon Apps to Paragon Apps
- Non- Paragon Apps to Paragon Apps
- Non- Paragon Apps to Non-Paragon Apps
- Browser to Native Apps and visaversa

Functional Specifications

Javascript Implementation for Paragon Apps

- Paragon Apps can discover other paragon apps and send messages to it point-to-point.
- Paragon Apps can consume messages intended for it by adding a message handler.
- Paragon Apps can launch other Paragon apps.
- Paragon Apps can also take advantage of the messaging platform as a generic message bus for other purposes by utilizing the topic based publish and subscribe features.

Messaging Extended Over Websockets (MEOW)

- Extend messaging to non-paragon apps by providing a simple json formatted messaging protocol over websockets.

JavaScript Language Binding for MEOW

- Provide a pure Javascript Language Binding for MEOW designed for web browsers that support websockets.

MEOW Protocol

Connecting to the platform

The Paragon Message Bus platform is process that runs on the GS Desktop. You connect to it via localhost port 8000. Use the typical garden variety websocket to connect the platform with a url similar to following:

ws://localhost:8000/paragon/messagebus?appid=*{provisioned app id}*

The platform only allows apps with a valid app id to connect to it.

Message types

All message types follow a JSON format. The message types include:

send

```
{type:"send", address:"topic", message:{...}, rid:"client_provided_id"}
```

type

Indicates the type of message which implies the expected behavior.

address

The topic is a string in any format. However it's highly suggested you follow a naming convention to maintain readability and avoid name collisions. Dot notation is always good choice to create a manageable name space. For example: *Division.Department.Team.Application*

message

The content to be sent. It must be a JSON object which can contain all valid JSON types.

rid

Response Id (rid) is a client side application provided string value to coordinate the receipt of a response if the other application listening for messages on the topic decides to respond. The rid is optional. If there is no expected response or sender doesn't want a response, then don't include the rid in the send message.

Note:

More than one subscriber can subscribe to a topic. *send* only sends it one of them in a round-robin fashion. If the intent is to deliver the message to all subscribers, then use *publish* instead.

publish

```
{type:"publish", address:"topic", message:{...}}
```

type

Indicates the type of message which implies the expected behavior.

address

The topic is a string in any format. However it's highly suggested you follow a naming convention to maintain readability and avoid name collisions. Dot notation is always good choice to create a manageable name space. For example: *Division.Department.Team.Application*

message

The content to be sent. It must be a JSON object which can contain all valid JSON types.

Note:

More than one subscriber can subscribe to a topic. *publish* delivers the message to all subscribers. If you intend for only one of any of the subscribers receive the message then use *send* instead.

subscribe

```
{type:"subscribe", address:"topic", rid:"client_provided_id"}
```

type

Indicates the type of message which implies the expected behavior.

address

The topic is a string in any format. However it's highly suggested you follow a naming convention to maintain readability and avoid name collisions. Dot notation is always good choice to create a manageable name space. For example: *Division.Department.Team.Application*

rid

Response Id (rid) is a client side application provided string value to coordinate the receipt of a response of whether or not the attempt to subscribe to topic specified in the address field was successful. rid is optional. Leave it out if confirmation is not required.

Note:

Once subscribed to a topic, all messages *publish*ed to the topic will be sent across the socket to the connected subscribing end point app. Messages delivered to a topic using *send* are only guaranteed to reach the subscribing end point app if it is the only subscriber. Otherwise another subscribing end point can receive it instead (refer to send api).

unsubscribe

```
{type:"unsubscribe", address:"topic", rid:"client_provided_id"}
```

type

Indicates the type of message which implies the expected behavior.

address

The topic is a string in any format. However it's highly suggested you follow a naming convention to maintain readability and avoid name collisions. Dot notation is always good choice to create a manageable name space. For example: *Division.Department.Team.Application*

rid

Response Id (rid) is a client side application provided string value to coordinate the receipt of a response of whether or not the attempt to subscribe to topic specified in the address field was successful. rid is optional. Leave it out if confirmation is not required.

Note:

Unsubscribe from the topic referenced in the address field will stop any further messages sent to that topic from being delivered to the end point application.

message

```
{type:"message", address:"topic", replyaddress:"reply_topic", message:{...}}
```

type

Indicates the type of message.

address

The topic from which the message was delivered.

replyaddress:

The topic to send a reply back if necessary. This field will only appear in the message if the sender use the send message containing a rid indicating an interest in a response.

message

The content delivered from the address.

Note:

End point applications receive this message as a result of subscribing to the topic referred to in the address field.