JSON-RPC

January 18, 2023

Contents

JSON-RPC objects . . . 2
Request . . . 2
Response . . . 3
Notification . . . 4
Error . . . 4
Interfaces . . . 5
ContextManager . . . 7
ContextData . . . 16
ContextAction . . . 19
Registry . . . 20
ContextParticipant . . . 25
ContextAgent . . . 29
ContextSession . . . 30
ParticipantMonitor . . . 33
Transport protocols . . . 34
NamedPipes . . . 34
WebSockets . . . 35
Netstring . . . 36

JSON-RPC 2.0 is a simple and light-weight remote procedure call protocol. It is transport agnostic and Manatee has built-in support for three transports; WebSockets, NamedPipes and Netstring/Tcp.

JSON-RPC messages are divided into four types. Each type places specific requirements on content and behavioural semantics, e.g. a `Request` should always be followed by a `Response`.

JSON-RPC objects

Request

The `Request` object *must* contain:

- `jsonrpc` indicating the version of JSON-RPC supported – always `"2.0"`
- `id` is an identifier (string) used to correlate the request with the following response
- `method` is a string identifying which invocation the requests represents

Furthermore it *can* contain:

- `params` which is an object with named parameters for the method to be invoked

The following is an example of a `Request` object to invoke the `Plus` method with two arguments; `A` and `B`.

```
{
  "jsonrpc":  "2.0",
  "id":       "123",
  "method":   "Plus",
  "params":   { "A": 200, "B": 100 },
}
```

Response

The `Response` object *must* contain:

- `jsonrpc` indicating the version of JSON-RPC supported – always `"2.0"`
- `id` is the id of the `Request` object that this is a response to

Furthermore it *must* contain either:

- `result` to indicate a *successfu*l invocation, containing an object with the result, or
- `error` containing an `Error` object to indicate a *failed* invocation.

A `Response` for the `Plus` invocation above could be:

```
{
  "jsonrpc":  "2.0",
  "id":       "123",
  "result":   { "A+B": 300  },
}
```

or in case of failure;

```
{
  "jsonrpc":  "2.0",
  "id":       "123",
  "error":    { "code": 9000, "message": "A is not a number" },
}
```

::: tip Each request MUST be followed by a response

When receiving a `Request` message the client or server must respond with a `Response` meessage. Failure to do so will be considered a protocol violation.

:::

Notification

A `Notification` is similar to a `Response` except it *must not* contain an `id`.

An example could be;

```
{
  "jsonrpc":  "2.0",
  "method":   "UserLoggedIn",
  "params":   { "User": "John Doe" }
}
```

Error

An `Error` object *must* contain the following properties:

- `code` an integer denoting the type of the error
- `message` a string describing the error

Furthermore the following are optional:

- `data` an object with further details about the error

An example error could be:

```
{
  "code":     -32601, // Method not found
  "message":  "The method Minus is not found"
}
```

The following error codes are supported.

Error	Code
ParseError	`-32700`
InvalidRequest	`-32600`
NotFound	`-32601`
InvalidParams	`-32602`
InternalServerError	`-32000`
TimeOut	`-32001`
ProtocolViolation	`-32002`

Error	Code
ParticipantAlreadyJoined	`-32100`
LaunchFailure	`-32101`
InvalidContextCoupon	`-32102`
NotInTransaction	`-32103`
UnknownParticipant	`-32104`
TransactionInProgress	`-32105`

::: warning No batch support

We do not support batch operations as defined in the JSON-RPC specs.

:::

Interfaces

The interfaces available over JSON-RPC are the `ContextManager`, `ContextData` `ContextAction` as defined in the HL7 CCOW specification v1.6.2. Any connecting clients are expected to implement the `ContextParticipant` interface (and possibly `ParticipantMonitor` as well) to enroll in surveys etc when the common context changes.

This section assumes a knowledge of the CCOW standard and the interfaces defined therein.

A common sequence of interactions between a client (in CCOW terms this is called a “participant”) runs as illustrated below.

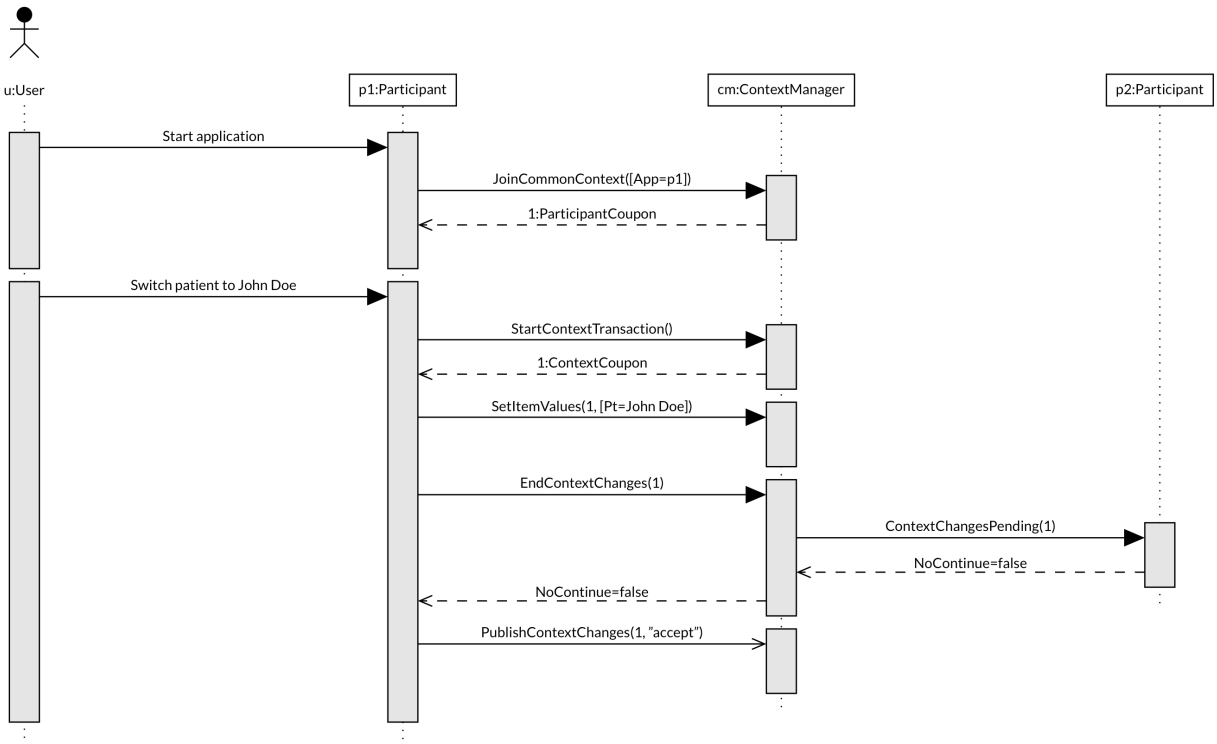

Figure 1: CCOW sequence diagram

The overall theme is that a particpant first *joins* a common context and once joined the partipant may change items in the common context while it must similarly respond to context changes originating with other participants in that same context.

The following describes the methods in each interface which the server (Manatee) exposes and the interfaces that the client is expected to implement.

::: details Conventions and abbreviations used in the following sections

The common context is a map of strings to strings. The keys in the map are called `subjects`.

In the coming sections the symbols >>> means that the following JSON-RPC message is sent from the participant to the context manager while the reverse <<< means the context manager is sending a message to the participant.

The names of the methods used in the JSON-RPC requests and notifications are of the form `<interface-name>.<method-name>` e.g. `ContextManager.JoinCommonContext`.

Abbreviations

- CM = ContextManager

- CP = ContextParticant
- CD = ContextData
- CA = ContextAction
- tx = transaction

:::

ContextManager

The `ContextManager` interface contains methods for creating and controlling context change transactions. Together with the `ContextData` interface it provides the complete functionality to change the content of the common context.

The following methods can be found in this interface:

- JoinCommonContext
- LeaveCommonContext
- SuspendParticipation
- ResumeParticipation
- StartContextChanges
- UndoContextChanges
- EndContextChanges
- PublishChangesDecision
- MostRecentContextCoupon
- ImplementationInformation

JoinCommonContext

This method is used to enroll the CP in the common context. It must precede most of the other invocations - exceptions to this rule includes ContextSession.Create and .Activate.

Request

The value for `method` is `"ContextManager.JoinCommonContext"` for JoinCommonContext invocations. This naming rule of interface-name followed by a dot then the method name is consistent in all requests and notifications.

The following arguments are applicable to this invocation:

- `ApplicationName` *[string, required]* is used by the CM to determine which parts (which subjects) that the CP is allowed to read/write. The CM must know about the application joining beforehand. This is accomplished via the Cuesta configuration interface.

- `ComponentId` *[string, optional]* determines which session is joined (see Registry). If no value is given then the current active session is joined.
- `SendContextInTxMethods` *[bool, optional]* if **`true`** then the ContextChangesPending request and the ContextChangesAccepted notification includes the current common context to save a the participant the trouble of using the ContextData methods to get this information.

Response

The response contains the following properties:

- `ParticipantCoupon` *[int]* represents the participant in this context. CP should hold on to this to reuse in later interactions with the CM.
- `Tag` *[string, optional]* is an optional identifier for the context.
- `ComponentId` *[string]* the id of the context.
- `Color` *[string]* an hex encoded color for the context. The CP can display this in its UI when joined.

Example

```
>>>
{
  "jsonrpc":  "2.0",
  "id":       "1",
  "method":   "ContextManager.JoinCommonContext",
  "params":   {
                "ApplicationName":        "Foo",
                "ComponentId":            "100",
                "SendContextInTxMethods": true
              }
}

<<<
{
  "jsonrpc":  "2.0",
  "id":       "1",
  "result":   {
                "ParticipantCoupon":  1001,
                "Tag":                "Bar",
                "ComponentId":        "1234",
                "Color":              "#000000"
               }
}
```

LeaveCommonContext

This method is invoked to leave the common context.

Request

- `ParticipantCoupon` *[long]* the identifier for the CP to leave the common context

Response

The response will always be an empty response in case of success. Otherwise an error will be set.

Example

```
>>>
{
  "jsonrpc":  "2.0",
  "id":       "2",
  "method":   "ContextManager.LeaveCommonContext",
  "params":   { "ParticipantCoupon": 1001 }
}

<<<
{
  "jsonrpc":  "2.0",
  "id":       "2",
}
```

StartContextChanges

This method is invoked to initiate a context change transaction in which one or more items in the context changes values.

Once the transaction has been started it **must** be completed (committed or aborted) within a configured deadline (default is 30 seconds). If this is not done, the transaction will be aborted.

Request

It accepts a single argument:

- `ParticipantCoupon` *[long]* which is the value given to the participant when it joined the common context.

Response

The response will also contain a single value if the invocation was successful.

- `ContextCoupon` *[long]* which represent the context change transaction and must be used in e.g. the `SetItemValues` method when specifying which parts of the context to change.

Example

```
>>>
{
  "jsonrpc":  "2.0",
  "id":       "3",
  "method":   "ContextManager.StartContextChanges",
  "params":   { "ParticipantCoupon": 1001 }
}

<<<
{
  "jsonrpc":  "2.0",
  "id":       "3",
  "result":   { "ContextCoupon":  2001 }
}
```

The client can now start changing the desired parts of the context using the context coupon and finally commit the changes.

UndoContextChanges

Aborts a transaction *without* notifying other participants.

Request

Takes a single argument:

- `ContextCoupon` *[long]* designating the context change transaction to abort.

Response

The response will always be an empty response in case of success. Otherwise an error will be set.

Example

```
>>>
{
  "jsonrpc":  "2.0",
  "id":       "4",
  "method":   "ContextManager.UndoContextChanges",
  "params":   { "ContextCoupon": 2001 }
}

<<<
{
  "jsonrpc":  "2.0",
  "id":       "4"
}
```

EndContextChanges

The `EndContextChanges` method is invoked after the desired changes have been made to the common context in a context change transaction. The CM will now query all participants in the same common context to determine whether there are any objections to committing the tx and return the results of this query to the caller.

Request

Takes a single argument:

- `ContextCoupon` *[long]* designating the context change transaction to end.

Response

The response contains:

- `NoContinue` *[bool]* which is **`true`** if one or more participants rejected the change.
- `Responses` *[string[]]* a list of human-readable responses from those participants that rejected the change.

Example

The following is an example where a participant has rejected the change because it needs the user to do a task prior to switching. It is up to each individual participant to determine when a transaction can proceeed according to the their internal business logic.

```
>>>
{
  "jsonrpc":  "2.0",
  "id":       "4",
  "method":   "ContextManager.EndContextChanges",
  "params":   { "ContextCoupon": 2001 }
}

<<<
{
  "jsonrpc":  "2.0",
  "id":       "4",
  "result":   {
                "NoContinue":   true,
                "Responses":    ["User must save document in NNN prior
                   to change of patient"]
              }
}
```

Following an `EndContextChanges` the initiator of the transaction must now decide whether to proceed and commit the transaction or abort.

PublishChangesDecision

Once `EndContextChanges` has been invoked it is up to the tx initiator to decide if the tx should be committed. That is what this method is for.

If the result `EndContextChanges` has a `NoContinue` value of **`true`** the initiator can still decide to commit the transaction. The rejecting participant may be removed from the common context in that case.

Request

The invocation takes two parameters:

- `ContextCoupon` *[long]* to identify the tx to commit/abort.
- `Decision` *[string]* is the decision to commit when the value `"accept"` is given. To abort use `"cancel"`.

Response

The response will always be an empty response in case of success. Otherwise an error will be set.

Example

```
>>>
{
  "jsonrpc":  "2.0",
  "id":       "5",
  "method":   "ContextManager.PublishChangesDecision",
  "params":   {
                "ContextCoupon":  2001,
                "Decision":       "accept"
              }
}

<<<
{
  "jsonrpc":  "2.0",
  "id":       "5"
}
```

SuspendParticipation

Use this to temporarily remove the participant from the common context.

Request

Takes the participant coupon of the participant to suspend.

- `ParticipantCoupon` *[long]* identifies the participant to suspend.

Response

The response will always be an empty response in case of success. Otherwise an error will be set.

Example

```
>>>
{
  "jsonrpc":  "2.0",
  "id":       "6",
  "method":   "ContextManager.SuspendParticipation",
  "params":   { "ParticipantCoupon":  1001 }
}
```

```
<<<
{
  "jsonrpc":  "2.0",
  "id":       "6"
}
```

ResumeParticipation

Undo a participant suspension.

Request

Takes the participant coupon of the participant to resume.

- `ParticipantCoupon` *[long]* identifies the participant to resume.

Response

The response will always be an empty response in case of success. Otherwise an error will be set.

Example

```
>>>
{
  "jsonrpc":  "2.0",
  "id":       "7",
  "method":   "ContextManager.ResumeParticipation",
  "params":   { "ParticipantCoupon":  1001 }
}

<<<
{
  "jsonrpc":  "2.0",
  "id":       "7"
}
```

MostRecentContextCoupon

This method will return the context coupon of the most recent successfull context tx (or the initial coupon generated for the initial context if no transactions have been completed).

::: tip Shorthand is -1

Use `-1` as `ContextCoupon` in the methods which has this argument to automatically use the most recent context coupon. This saves a request/response occasionally.

:::

Request

Takes no arguments.

Response

The response will contain the `ContextCoupon`.

- `ContextCoupon` *[long]* the most recent committed tx.

Example

```
>>>
{
  "jsonrpc":  "2.0",
  "id":       "8",
  "method":   "ContextManager.MostRecentContextCoupon"
}

<<<
{
  "jsonrpc":  "2.0",
  "id":       "8",
  "result":   { "MostRecentContextCoupon": 123 }
}
```

ImplementationInformation

Extract details about the implementation of the CM.

Request

Takes no arguments.

Response

Returns the following information:

- `ComponentName` *[string]* the name of the component.
- `Manufacturer` *[string]* is always “Sirenia”.
- `PartNumber` *[string]* further details about the CM.
- `RevMajorNum`, `RevMinorNum`, `RevMaintenanceNum` *[string]* are the components of the version (of the CM).
- `TargetOS` *[string]* the os for which the CM was built.
- `TargetOSRev` *[string]* the OS version compatibility.

ContextData

The `ContextData` interface contains the methods needed to read and write from and to the common context.

The following methods are available here:

- GetItemNames
- GetItemValues
- SetItemValues

GetItemNames

Get a list of the key/subject of all items in the designated context.

Request

The request contains an identifier for the context to query for names.

- `ContextCoupon` *[long]* the context to return names for. `-1` can be used for the latest committed names.

Response

The response is a string array of all the names/subjects in the context.

- `Names` *[string[]]* the names (filtered to match the permissions of the CP) in the context

Example

```
>>>
{
  "jsonrpc":  "2.0",
  "id":       "9",
  "method":   "ContextData.GetItemNames",
  "params":   { "ContextCoupon": 2001 }
}

<<<
{
  "jsonrpc":  "2.0",
  "id":       "9",
  "result":   { "Names": ["foo", "bar"] }
}
```

GetItemValues

Use this method to retrieve the values of a selected portion of the context.

Request

- `ItemNames` *[string[]]* the list of names to return values for
- `ContextCoupon` *[long]* the identifier of the context to query
- `OnlyChanges` *[bool, optional]* if **`true`** and the `ContextCoupon` denotes an uncompleted transaction then only the values changed will be returned

Response

- `ItemValues` *[string[]]* is a list of the names and values (alternating)

Example

```
>>>
{
  "jsonrpc":  "2.0",
  "id":       "10",
  "method":   "ContextData.GetItemValues",
  "params":   {
                "ItemNames":      ["foo", "bar"],
```

```
                  "ContextCoupon":  2001
               }
}

<<<
{
  "jsonrpc":  "2.0",
  "id":       "10",
  "result":   { "ItemValues": ["foo", 100, "bar", 200] }
}
```

Note that the returned values contain both the names and the values such that index `N` contains the name and `N+1` the value where `{ N | N % 2 == 0 }`.

SetItemValues

This method can only be called after `StartContextChanges` and will set the values to be changed in the transaction to the given values.

Request

The request contains the names and values to change.

- `ItemNames` *[string[]]* the names of the items to change. The index of this property must match up with the value to change in the `ItemValues` property.
- `ItemValues` *[string[]]* the values to change matching the names (on index) given in `ItemNames` .
- `ContextCoupon` *[long]* the identifier for the context in which the change should take place.
- `ParticipantCoupon` is the identifier for the CP doing the change.

Response

The response will always be an empty response in case of success. Otherwise an error will be set.

Example

```
>>>
{
  "jsonrpc":  "2.0",
  "id":       "11",
  "method":   "ContextData.GetItemValues",
```

```
  "params":   {
                "ItemNames":          ["foo", "bar"],
                "ItemValues":         [1000, 2000],
                "ContextCoupon":      2001,
                "ParticipantCoupon":  1001
              }
}

<<<
{
  "jsonrpc":  "2.0",
  "id":       "11",
}
```

ContextAction

The `ContextAction` interface is used to invoke named actions in action agents (which may be other context participants).

Similarly to how information about which parts of the common context is supported by which CP, we use the Registry to store information about which actions are supported by which agents.

Perform

This is the only method available and it is used to invoke a particular action in a context agent. The agent will (if using this interface) have its ContextAgent.ContextChangesPending method called.

Request

- `ActionIdentifier` *[string]* an identifier for the action to invoke.
- `ParticipantCoupon` *[long]* the identifier of the caller.
- `InputNames` *[string[], optional]* a list of names for arguments given in `InputValues`
- `InputValues` *[string[], optional]* a list of arguments matching the names given in `InputNames` on index.

Response

- `Decision` *[string]* a string designating wether the CM managed to locate an agent and execute the action.

- `Reason` *[string]* a human-readable explanation of why the method failed (if it failed, empty otherwise).
- `OutputNames` *[string[]]* a list of names for return values.
- `OutputValues` *[string[]]* a list of return values.

Example

```
>>>
{
  "jsonrpc":  "2.0",
  "id":       "12",
  "method":   "ContextAction.Perform",
  "params":   {
                "ActionIdentifier":   "Plus",
                "InputNames":         ["A", "B"],
                "InputValues":        [1, 2],
                "ParticipantCoupon":  1001
              }
}

<<<
{
  "jsonrpc":  "2.0",
  "id":       "12",
  "result":   {
                 "OutputNames":   ["A+B"],
                 "OutputValues":  [3],
                 "Decision":      "accept"
              }
}
```

Registry

The `Registry` interface exposes an `OfType` and `ById` methods which can be used to query for various configuration- and instance-level items. It is primarily used to list instances of context managers and the result can be used in the `JoinCommonContext` method. However since the argument here is optional and the default behaviour is to join the currently active CM (which is most likely what you want) the methods here are rarely used.

Note that this interface is a poor mans convenience proxy for using the Kwanza registry directly which is a proper context management registry (CMR).

OfType

Request

- `Typename` *[string, optional]* the type of the item to query, default is `"ContextManager"`.

Response

- `Identifiers` *[string[]]* a list of identifiers for items of the given type.

Example

```
>>>
{
  "jsonrpc":  "2.0",
  "id":       "13",
  "method":   "Registry.OfType",
  "params":   { "Typename": "ContextManager" }
}

<<<
{
  "jsonrpc":  "2.0",
  "id":       "13",
  "result":   { "Identifiers": ["foo", "bar", "123"] }
}
```

ById

Return a single item given its identifier.

Request

- `Typename` *[string, optional]* the type of the item to query, default is `"ContextManager"`.
- `Identifier` *[string]* the identifier for the item to return.

Response

The response contains a single item

- `Item` *[object]* the item with the given identifier - the structure of the object is dependent of its type. All types of items have the properties;
  - `Identifier` *[string]* the identifier of the item.
  - `Typename` *[string]* the type of the item.

For the `"ContextManager"` type additional properties of the item are;

- `Tag` *[string]* a human-readable optional tag given to the CM when created
- `Color` *[string]* an auto-generated color associated with the CM

Example

```
>>>
{
  "jsonrpc":  "2.0",
  "id":       "14",
  "method":   "Registry.ById",
  "params":   {
                "Typename":     "ContextManager",
                "Identifier":   "foo"
              }
}

<<<
{
  "jsonrpc":  "2.0",
  "id":       "14",
  "result":   {
                "Item": {
                  "Identifier":   "foo",
                  "Typename":     "ContextManager",
                  "Tag":          "bar",
                  "Color":        "#4E4E4E"
                }
              }
}
```

LocalActions

The `LocalActions` endpoint returns all available *actions* from the local CM.

Request

No arguments needed.

Response

A list of *actions* each containing;

- `Identifier` *[string]* for the *action*.
- `Name` *[string]* a human readable name.
- `Subject` *[string]* a human readable *identifier* although no guarantees of uniqueness is offered.
- `ApplicationName` *[string]* contains the name of the application/agent to which the *action* is attached.
- `ApplicationDescription` *[string]* is a human readable description of the application.
- `Groups` *[group[]]* contains a list of attached groups (can be used to do further filtering filtering) where each group contains the following properties;
  - `Identifier` *[string]* for the group.
  - `Name` *[string]* the name of group.
  - `Description` *[string]* a human readable description of the group.
  - `Key` *[string]* an unspecified configuration property (use at your own discretion when configuring).

Example

```
>>>
{
  "jsonrpc":  "2.0",
  "id":       "140",
  "method":   "Registry.LocalActions"
}

<<<
{
  "jsonrpc":  "2.0",
  "id":       "140",
  "result":   {
    "Actions": [
      {
        "Identifier": "1234",
        "Name": "Action #1",
        "Subject": "[eu.sirenia]Action.One",
```

```
18          "ApplicationName": "App #1",
19          "ApplicationDescription": "Application One",
20          "Groups": [
21            {
22              "Identifier": "1",
23              "Name": "Group #1",
24              "Description": "",
25              "Key": ""
26            },
27            {
28              "Identifier": "2",
29              "Name": "Group #2",
30              "Description": "bar",
31              "Key": ""
32            }
33          ]
34        },
35        ...
36      ]
37    }
38  }
```

LocalApplications

The `LocalApplications` endpoint returns all available *applications* from the local CM.

Request

No arguments needed.

Response

A list of *actions* each containing;

- `Identifier` *[string]* for the *action*.
- `ApplicationName` *[string]* same as `Identifier`.
- `FriendlyName` *[string]* a human readable name.
- `ApplicationType` *[string]* the type of the application.
- `Version` *[string]*

Example

```
>>>
{
  "jsonrpc":  "2.0",
  "id":       "180",
  "method":   "Registry.LocalApplications"
}

<<<
{
  "jsonrpc":  "2.0",
  "id":       "180",
  "result":   {
    "Actions": [
      {
        "Identifier": "1234",
        "ApplicationName": "1234",
        "FriendlyName": "Application Foo",
        "ApplicationType": "CONTEXTPARTICIPANT",
        "Version": "v1.2.3",
      },
      ...
    ]
  }
}
```

ContextParticipant

The `ContextParticipant` interface is expected to be implemented by the CP joining the common context. It contains methods that will be **invoked by the CM on the CP** and the flow of requests and responses are therefore inverted with

- *requests/notifications/errors* coming from the server/CM and targeting the client/CP
- *responses* generated by the client/CP and returned to the server/CM.

The methods contained in this interfaces:

- ContextChangesPending
- ContextChangesAccepted
- ContextChangesCancelled
- CommonContextTerminated
- Ping

ContextChangesPending

This method is invoked by the CM when another CP issues an `EndContextChanges` invocation and is the query to this CP to determine whether it can *accept* a change in the common context.

Request

The arguments supplied are:

- `ContextCoupon` *[long]* the identifier for the context transaction in which the changes occur
- `Changes` *[map<string,string>, optional]* if the `SendContextInTxMethods` argument for `JoinCommonContext` is **`true`** then this argument will contain the changes to the context directly. If not then the participant may query for values using the `GetItemValues` method.

Response

The response should contain:

- `Decision` *[string]* `"accept"` to accept the change or `"cancel"` to not.
- `Reason` *[string, optional]* should contain a human-readable message as why `Decision` is not `"accept"` if that is the case.

Example

```
<<<
{
  "jsonrpc":  "2.0",
  "id":       "100",
  "method":   "ContextParticipant.ContextChangesPending",
  "params":   {
                "ContextCoupon":     2003,
                "Changes":          { "foo": "bar", "qux": "zop" }
              }
}

>>>
{
  "jsonrpc":  "2.0",
  "id":       "100",
  "result":   {
                  "Decision":   "cancel",
                  "Reason":     "'zop' is not a valid value for 'qux'"
```

```
                }
}
```

ContextChangesAccepted

This notification is sent when the context change transaction commits.

::: tip The CP must update its internal state in response

The CP is required to update its internal state to match that of the common context when it receives a `ContextChangesAccepted` notification. If it cannot do this it must leave the common context.

:::

Request

The arguments are the same as for `ContextChangesPending`.

Response

No response is expected. It is a protocol violation to supply a response.

```
<<<
{
  "jsonrpc":  "2.0",
  "method":   "ContextParticipant.ContextChangesAccepted",
  "params":   {
                "ContextCoupon":     2003,
                "Changes":          { "foo": "bar", "qux": "zop" }
              }
}
```

ContextChangesCancelled

This notification is sent when the context change transaction aborts.

Request

- `ContextCoupon` *[long]* the identifier for the transaction rolling back.

Response

No response is expected. It is a protocol violation to supply a response.

```
<<<
{
  "jsonrpc":  "2.0",
  "method":   "ContextParticipant.ContextChangesCancelled",
  "params":   {
                "ContextCoupon":     2003,
              }
}
```

CommonContextTerminated

This notification is sent when the context manager terminates.

Request

Contains no additional arguments.

Response

No response is expected. It is a protocol violation to supply a response.

Example

```
<<<
{
  "jsonrpc":  "2.0",
  "method":   "ContextParticipant.CommonContextTerminated"
}
```

Ping

The `ping` method is invoked periodically to check the liveliness of the CP. If the CP does not respond within a configurable timeout then the CP is ejected from the common context.

Request

Contains no additional arguments.

Response

Should contain no additional result.

Example

```
<<<
{
  "jsonrpc":  "2.0",
  "id":       "101",
  "method":   "ContextParticipant.Ping"
}

>>>
{
  "jsonrpc":  "2.0",
  "id":       "101",
}
```

ContextAgent

The `ContextAgent` interface is used by the CM to perform actions when its Perform method is invoked. A single application may implement both the ContextParticipant as well as this interface, thus both taking the participant and agent role simultaneuosly.

ContextAgent.ContextChangesPending

Invoked when this agent must perform an action. The arguments are;

Request

- `ActionIdentifier` *[string]* contains the name/identifier of action to run.
- `Subject` *[string]* contains the subject of the action if available.
- `ItemNames` *[string[]]* the names of the input arguments for the action.
- `ItemValues` *[string[]]* the values of the input arguments corresponding to the `ItemNames` on index.
- `AgentCoupon` *[long]* an identifier for this agent.
- `ContextCoupon` *[long]* an identifier for this “transaction”.
- `Principal` *[string]* the identifier of the CM asking for the action to be run.

Response

The response should contain:

- `Decision` *[string]* `"accept"` if the action was run successfully, `"cancel"` otherwise.
- `Reason` *[string]* a human-readable message if the action failed.
- `ItemNames` *[string[]]* a list of names for the output generated by the action.
- `ItemValues` *[string[]]* a list of values for the names in `ItemNames`.

Example

```
<<<
{
  "jsonrpc":  "2.0",
  "id":       "102",
  "method":   "ContextAgent.ContextChangesPending",
  "params":   {
                "ActionIdentifier": "Plus",
                "ItemNames":        ["A", "B"],
                "ItemValues":       [100, 200],
                "AgentCoupon":      3001,
                "ContextCoupon":    2001,
                "Principal":        "foobar"
              }
}
>>>
{
  "jsonrpc":  "2.0",
  "id":       "102",
  "result":   {
                "Decision":         "Accept",
                "ItemNames":        ["A+B"],
                "ItemValues":       [300]
              }
}
```

ContextSession

The `ContextSession` interface contains methods for creating and activating context sessions. Each session is run by its own CM and has the following properties:

- `color` is a color which participants can use to indicate session participation (in their own UI),

- `tag` is a unique string which can be assigned when the session is created.

The `tag` is unique in the sense that if the `Create` method is invoked with an already existing tag then the existing session is returned and no new session is created.

The following methods can be found in this interface:

- Create
- Activate

Create

This method is used to create a new session. If a `tag` is given and an existing session exists which has the same tag then the existing session is returned. A newly created session starts by setting itself as the active session.

Request

The value for `method` is `"ContextSession.Create"`.

The following arguments are applicable to this invocation:

- `Tag` *[string, optional]* is the "unique" tag assigned to the newly created session. If a session with an identical tag already exists then no new session is created and the existing session is returned.
- `Color` *[string, optional]* which color to apply to the session. You can supply a hex value or a named color. If no color is specified the CM will choose from its own palette.
- `NoFullAuto` *[bool, optional]* if **`true`** then the session will not attach automatically to discovered applications but let the user do this. Default is **`false`**.

Response

The response contains the following properties:

- `ComponentId` *[string]* the id of the session/context.

Example

```
>>>
{
  "jsonrpc":  "2.0",
  "id":       "200",
  "method":   "ContextSession.Create",
```

```
  "params":   {
                "Tag":    "Foo",
                "Color":  "green",
              }
}

<<<
{
  "jsonrpc":  "2.0",
  "id":       "200",
  "result":   { "ComponentId": "1234" }
}
```

Activate

This method is used to activate a session. There can only be one “active” session at a time. The active session is the session that will automatically attach to new applications if it runs in `FullAuto` mode. Non-active sessions will still be synchronising context and for external applications the difference between having joined an active vs inactive session is not noticeable.

Request

The value for `method` is `"ContextSession.Activate"`.

The following arguments are applicable to this invocation:

- `ComponentId` *[string, required]* is the identifier of the session/context to activate.

Response

The response contains the following properties:

- `ComponentId` *[string]* the id of the session/context.

Example

```
>>>
{
  "jsonrpc":  "2.0",
  "id":       "201",
  "method":   "ContextSession.Activate",
  "params":   { "ComponentId": "1234" }
```

```
}

<<<
{
  "jsonrpc":  "2.0",
  "id":       "201"
}
```

ParticipantMonitor

This interface is *not* part of the CCOW standard and only contains *notifications*. It is optional to implement but the intent is to provide the participant with more information about the other participants in the same context.

ParticipantsChanged

This notification is sent whenever a participant change occurs in the common context. When a new CP joins it will also get a number of these notifications to notify about current members.

Request

- `ApplicationName` *[string]* is the identifier for the CP (not its context coupon)
- `Event` *[string]* is the event which occurred, can be:
  - `"ParticipantJoined"`
  - `"ParticipantLeft"`
  - `"ParticipantAlreadyHere"`
  - `"ParticipantSuspended"`
  - `"ParticipantResumed"`

Example

```
<<<
{
  "jsonrpc":  "2.0",
  "method":   "ParticipantMonitor.ParticipantsChanged",
  "params":   {
                "ApplicationName":    "Foo",
                "Event":              "ParticipantJoined"
              }
}
```

ApplicationLaunched

This notification is sent whenever the CM launches an application as e.g. part of a `Perform` invocation.

Request

- `ApplicationName` *[string]* is the identifier for the CP (not its context coupon)
- `Hwnd` *[long]* is the window handle for the launched application
- `Pid` *[int]* is the id of the process launched

Example

```
<<<
{
  "jsonrpc":  "2.0",
  "method":   "ParticipantMonitor.ApplicationLaunched",
  "params":   {
                "ApplicationName":    "Foo",
                "Hwnd":               1234,
                "Pid":                4321
              }
}
```

Transport protocols

We support a few different transport protocols for the JSON-RPC message protocol. The following describes how to connect and interact using them.

NamedPipes

The *NamedPipes* transport uses two bi-directional pipes.

The first pipe is used for the interfaces in which the client/CP sends requests to the server/CM. Those are

- ContextManager
- ContextData
- ContextAction
- Registry

The pipe is named `JSON-RPC v1/Context{Manager,Data,Action}/<Username>` and a server endpoint is created by the CM. The `<Username>` part should be replaced by the windows username of the current user. The client must connect to it with

- `PipeDirection = InOut`
- `TransmissionMode = Message`

Once the CP issues a `JoinCommonContext` and receives a `ParticipantCoupon` the CP must now connect to the second pipe in order for the CM to use the `ContextParticipant` and `ParticipantMonitor` interfaces.

This second pipe must be created as a client pipe with the same properties as the first and the CM will initiate the server end after the `JoinCommonContext` invocation successfully completes.

This pipe is named `JSON-RPC v1/ContextParticipant/<Username>/<ParticipantCoupon>`.

The reason why two named pipes are needed is such that the strict request/response sequence can be maintained and that the client and server may agree on this sequence based on the which requests/notifications/errors are issued by either.

The encoding for both pipes is UTF-8.

WebSockets

The *WebSockets* transport is conceptually simple. Initiating a connection and joining a context is done in one step by connecting to the url:

- `ws(s)://localhost:<port>/v2/ContextManager/JoinCommonContext`

The arguments for `JoinCommonContext` should be given as query parameters in the url e.g. `...?ApplicationName=<applicationname>&ComponentId=<componentid`. Furthermore the arguments for `ContextSession.Create` can be provided as well. This will correspond to invoking the `.Create` method before joining the context and the resulting session (created or matched on tag) will be used. In this case you will not need to provide `componentId`.

The `port` for the WebSocket server can be read from the registry at `HKCU\Software\Sirenia\Manatee\Ports\websocketserver` and `HKCU\Software\Sirenia\Manatee\Ports\websocketserversecureselfsigned` depending on wether a secure connection is needed.

Once the connection is established the fully duplexed interaction may begin with all interfaces using the same WebSocket.

Netstring

The *Netstring* transport is a raw TCP stream over which JSON-RPC messages are encoded using the Netstring scheme. Netstring is a simple encoding scheme in which the format is:

```
<byte-count>:<data>,
```

Where <data> are the bytes of an UTF-8 encoded string.

So for instance the JSON-RPC request:

```
{
  "jsonrpc":  "2.0",
  "id":       "1",
  "method":   "ContextManager.JoinCommonContext",
  "params":   {
                "ApplicationName":        "Foo",
                "ComponentId":            "100",
                "SendContextInTxMethods": true
              }
}
```

would be encoded as:

```
156:{"jsonrpc":"2.0","id":"1","method":"ContextManager.
    JoinCommonContext","params":{"ApplicationName":"Foo","ComponentId":"
    100","SendContextInTxMethods": true}},
```

Connection

The client should connect to the locahost:<port> where <port> can be found in the registry at HKCU\Software\Sirenia\Manatee\Ports\netstringtcp or HKCU\Software\Sirenia\Manatee\Ports\netstringtcpsecure. The TCP stream is encrypted using SSL for the secure connection and the server certificate is written to disk at %appdata%/Sirenia/Manatee/certs and the registry at HKCU\Software\Sirenia\Manatee\Certs\netstringtcpsecure if the client wishes to use it for validation.