Narrative App UI Specification

This document describes how the UI components of Narrative Apps are specified via fields in `spec.json` and `display.yaml` files. These spec files include input and output parameters for the app.

Structure of Folders and Files

To introduce UI specifications of Apps in an SDK repo you have to keep files and folders in a particular structure. Currently all UI method specifications (method-specs) live in ui/narrative/methods/ path. Each method has its own subfolder with the name meaning the app or method ID. Inside this folder a typical method has a “spec.json” file defining structural input/output and parameter type components, a “display.yaml” file defining display text and “img” subfolder containing icon and screenshot files.

Structural file (“spec.json”)

This is a file in JSON format containing these top-level properties:

- "ver" - version of each UI method-spec
- "authors" - list of GlobusOnline accounts of contributers
- "contact" - e-mail of contact person
- "categories" - list of tags (from fixed set of supported tags) categorizing this method into one or more groups shown in navigation interface as set of filters; in case particular method should be made invisible then "inactive" category should be added to this list (in place of "active" category if it’s present)
- "widgets" - structure defining "input" and "output" widgets (typically "input" widget is set to null meaning that standard one is used); “output” widget points to name of JavaScript widget supported by Narrative
- "parameters" - list structures defining each parameter type and other features except name (which is defined in “display.yaml” file)
- "behavior" - block of settings for mapping UI parameters to service function input parameters and separately for mapping from output results returned from service function to what should be passed to visualizing widget to show these results
- "job_id_output_field" - field for support of legacy code; for SDK method it should always be set into "docker" value

Parameters in “spec.json”

Each structure in array defined in "parameters" field defines one parameter shown in UI input panel of this method. Here is the list of main properties of parameter structure:

- "id" - ID of parameter for referring to it from “display.yaml” file and from mapping settings of "behavior" block
- "optional" - flag defining whether UI interface allows empty value for this parameter or it requires it to be set to something non-empty
- "advanced" - flag defining whether UI interface moves this parameter into “Advanced parameters” group collapsed by default or this parameter is listed in normal order
- "allow_multiple" - flag defining whether UI interface allows to define more than one value for this parameter and then treat these values as an array of values rather than one textual/numeric/boolean value
- "default_values" - if set to any non-empty string this value will appear as pre-defined one
- "field_type" - type of parameter, could be one of "text", "dropdown", "checkbox", "textarea", "textsubdata"
- "text_options" - optional block defining details of "text" type
- "dropdown_options" - optional block defining details of "dropdown" type
- "checkbox_options" - optional block defining details of "checkbox" type
- "textarea_options" - optional block defining details of "textarea" type
- "textsubdata_options" - optional block defining details of "textsubdata" type

Options for text parameter type in “spec.json”

Options in "text_options" block allow to specify many different ways of validation and behaviour. Field named "valid_ws_types" connects a parameter with one or more types of objects stored in Workspace KBase storage. In this mode Narrative interface will show available objects of listed types as drop-down for this parameter. For instance here is an example of "text_options" allowing to choose one of Genome objects stored in workspace:

```
"text_options" : {
    "valid_ws_types" : [ "KBaseGenomes.Genome" ]
}
```

If you would like to mark this parameter as output which means UI interface shouldn’t require chosen object to be present in Workspace storage you can set "is_output_name" sub-option to true like:

```
"text_options" : {
    "valid_ws_types" : [ "KBaseGenomes.Genome" ],
    "is_output_name" : true
}
```

Another sub-option is "validate_as" allowing to validate value entered in UI as "int" or "float". If you what some parameter to be an integer with minimum and/or maximum limits you can use properties like in this example:

```
"text_options" : {
    "valid_ws_types" : [ ],
```

```
        "validate_as": "int",
        "min_int" : 1,
        "max_int" : 200
    }
```

And similarly for float type:

```
    "text_options" : {
        "valid_ws_types" : [ ],
        "validate_as": "float",
        "min_float" : 1,
        "max_float" : 200
    }
```

Options for drop-down parameter type in “spec.json”

There is only one sub-option available inside "dropdown_options" block currently. It has “options” name and value is list of objects defining drop-down items. Each item object should have two properties: "value" defining internal item ID (it’s sent to back-end function when given item is selected) and "display" defining text shown for this item in UI. Here is an example of the whole "dropdown_options" block:

```
    "dropdown_options":{
        "options": [{
            "value": "lloyd",
            "display": "Lloyd"
        }, {
            "value": "hartigan_wong",
            "display": "Hartigan-Wong"
        }, {
            "value": "forgy",
            "display": "Forgy"
        }, {
            "value": "mac_queen",
            "display": "MacQueen"
        }]
    }
```

Options for checkbox parameter type in “spec.json”

Here is the list of sub-options available inside "checkbox_options" block:

- "checked_value" - defines value to be sent to service function when checkbox is selected

- "unchecked_value" - defines value to be sent to service function when checkbox is not selected

Options for textarea parameter type in “spec.json”

There is only one sub-option available inside "textarea_options" block:

- "n_rows" - defines number of lines shown for this textarea in UI.

Options for textsubdata parameter type in “spec.json”

This parameter type allows to select items that are parts of workspace object (let’s call them sub-objects). Here is the list of sub-options available inside "textsubdata_options" block:

- "multiselection" - flag (boolean) allowing to have more than one selected object
- "show_src_obj" - flag (boolean) shows name of workspace object where we are selecting sub-objects as well
- "allow_custom" - flag (boolean)
- "subdata_selection" - main block with following sub-options:
    - "path_to_subdata" - JSON-path leading to the level of an array of sub-objects (instead of string type JSON-path here is treated as an array of elements)
    - "subdata_included" - list of string JSON-paths to be loaded (in case JSON-path leads to certain field inside sub-objects then level of array of sub-objects is denoted as [*])
    - "constant_ref" - static reference to some object in public workspace (alternative to "parameter_id")
    - "parameter_id" - points to ID of another UI parameter used for selection of workspace object where we are selecting sub-objects
    - "selection_id" - name of field of sub-object which will be sent as selected value
    - "selection_description" - list of fields of sub-object to be shown for each selectable item
    - "description_template" - optional template defining the way of representation of fields from "selection_description" (placeholders of fields are defined as {{field-name}})

Here is an example of "textsubdata_options" block for model reactions in KBaseFBA.FBAModel object:

```
    "textsubdata_options" : {
      "subdata_selection": {
        "parameter_id" : "input_model",
        "subdata_included" : ["modelcompounds/[*]/id",
"modelcompounds/[*]/name","modelcompounds/[*]/formula"],
        "path_to_subdata": ["modelcompounds"],
        "selection_id" : "id",
```

```
      "selection_description" : ["name","formula"],
      "description_template" :"- {{name}} ({{formula}})"
    },
    "multiselection":true,
    "show_src_obj":false,
    "allow_custom":false
  }
```

Behavior in “spec.json”

There are three alternative sub-blocks available inside "behaviour" block:

- "service-mapping" - defines mapping rules for input and output data for typical SDK method (this way will be described below)
- "none" - could be used in case UI method is not supposed to run any service function (for instance when input parameters should be passed into widget directly)
- "script-mapping" - support for legacy software not recommended to use in SDK repos

In most cases "service-mapping" sub-block should be used. Here is the list of sub-elements available inside "service-mapping":

- "url" - defines URL end-point of deployed service (in case of SDK repos the convention requires to keep this parameter empty)
- "name" - module name of SDK repo registered in catalog (see module name in KIDL specification)
- "method" - name of service function to be called (see funcdef in KIDL specification)
- "input_mapping" - defines rules for mapping UI parameters onto service function input arguments
- "output_mapping" - defines rules for mapping output results returned from service function onto input options of visualizing widget showing these results

Both "input_mapping" and "output_mapping" sub-blocks are arrays of mapping items. Each mapping array is an object with following optional properties:

- "input_parameter" - ID of UI input parameter to be used as a source of mapping
- "constant_value" - constant value to be used as a source of mapping
- "narrative_system_variable" - system variable in narrative back-end to be used as a source of mapping (only "workspace" variable is currently officially supported)
- "target_property" - name of structure field to be set as a target of mapping
- "target_argument_position" (allowed for input mapping items only, default value is 0) - position of input argument of service function to be set as a target of mapping
- "target_type_transform" - optional rule allowing to modify passing value; here is the list of allowed transformations:
    - "none" (default value in case it not defined) - no modification
    - "ref" - changes object name into workspace reference by adding prefix with workspace name followed by “/”
    - "int" - treats text value as an integer

    - "list<inner-transformation>" - tries to prepare list of items or just iterate over items if it's a list already applying inner-transformation to each element
- "service_method_output_path" (allowed for output mapping items only) - defines JSON-path into output prepared for widget as a place for target value; if this path is empty array it corresponds to root point and all the data returned from service function will be captured

In group of source properties ("input_parameter", "constant_value", "narrative_system_variable") only one can be used. For target properties both "target_property" and "target_argument_position" can be used at the same time meaning that service function will get as argument with position from "target_argument_position" an object with property having name from "target_property" with target value.

Example for mappings in "spec.json"

Let's consider some example of mappings defined in "service-mapping" sub-block of "behaviour". Suppose we have function "func1" in module "module1" where we expect to get as input two arguments: a string and an object with internal field "input_prop" (in JSON this argument looks like {"input_prop": "..."}). And we have two UI parameters of type "text" with IDs "param1" and "param2". Output returned from the function is an array of objects containing only one object which has internal field "output_prop". Value of this field should be mapped to "option1" option in UI widget. In this case we will have following mappings:

```
  "behavior" : {
    "service-mapping" : {
      "url" : "",
      "name" : "module1",
      "method" : "func1",
      "input_mapping" : [
        {
          "input_parameter": "param1"
          "target_argument_position": 0
        }, {
          "input_parameter": "param2",
          "target_argument_position": 1,
          "target_property": "input_prop"
        }
      ],
      "output_mapping" : [
        {
          "service_method_output_path": [0, "output_prop"],
          "target_property": "option1"
        }
      ]
```

```
    }
  }
```

Display Text file (“display.yaml”)

This file has Yaml format. Here is the list of top-level block names:

- name - name of method listed in UI
- tooltip - more detailed explanation about the method shown on mouse-over event
- screenshots - list of names of screenshot files from “img” sub-folder
- icon (optional) - name of icon file from “img” sub-folder
- method-suggestions - list of objects defining the set of other methods that could be suggested to the user as related ones; there are two sub-elements “related” and “next” pointing to arrays of method IDs
- parameters - a map from parameter IDs defined in “spec.json” to objects designed to add textual information to these parameters (see details below)
- description - very detailed explanation about what and how the method does; it appears on separate web page describing this method
- publications (optional) - list of objects describing publications; each object includes two fields: “display-text” containing reference to scientific journal and “link” with URL to online resource

Each parameter is a pair in parameter map linking ID of this parameter (the only key of) and textual object having following fields:

- ui-name - name of parameter used to show given parameter in UI
- short-hint - short description shown in front of each parameter on right side of method input panel in Narrative
- long-hint - more detailed explanation available by mouse-over on question-mark sign (in case it’s the same as short-hint question-mark is not shown)
- placeholder (optional) - in case of parameter type is textual (one of "text", "textarea", "textsubdata") it defines placeholder text shown in gray color explaining the meaning of value user is going to set.