RoyalJSON

Last Update: June 24, 2019

Table of Contents

Introduction

RoyalJSON, or rJSON for short, is an unidirectional, human-read-and-writable data format for importing data from external sources into Royal TS/X.
Its primary objective is to provide users an easy, yet powerful way to get data stored outside of Royal TS/X into the application.

Prime examples where RoyalJSON shines are:

- You have a central database where you store credentials and want to use those credentials in Royal TS/X without having to manually add/update them.
- You have one or more VM hosts where you regularly (un-)provision VMs and want to keep them in sync with Royal TS/X.

At the moment, RoyalJSON is an import-only format and there's no way to export existing objects stored in Royal TS/X documents as RoyalJSON. This might change in the future though.

Once you have some RoyalJSON content, there are multiple ways to get the data into our applications:

- Via "Dynamic Folder" objects stored in your documents:
  Dynamic Folders allow you to specify a script that is executed and expected to return rJSON which is then parsed by the application. After successfully parsing the data, objects are created as children of the Dynamic Folder. All objects created by the Dynamic Folder are immutable.
- By creating and opening an rJSON document:
  rJSON documents are simple plain text files that contain rJSON data and are postfixed with the ".rjson" file extension. rJSON documents are immutable when opened with Royal TS/X.

In the future, additional means of working with rJSON in Royal TS/X may be provided.

The data format

As you might've guessed, RoyalJSON uses JSON to store its data. We use regular JSON and don't plan to add any RoyalJSON specific extensions to keep in line with the JSON spec and so you can put your existing JSON skills to good use. If you're not familiar with JSON, please consult one of the many references out there.

We think the best way to introduce you to the data format is by just showing it to you. So here we go:

```
1       {
2               "Objects": [
3                       {
4                               "Type": "Credential",
5                               "Name": "Root",
6                               "Username": "root",
7                               "Password": "!ehrfew9fe9gew7rgew@",
8                               "Description": "The root account",
9                               "ID": "000001"
10                      },
11                      {
12                              "Type": "TerminalConnection",
13                              "TerminalConnectionType": "SSH",
14                              "Name": "VM01",
15                              "ComputerName": "vm01",
16                              "CredentialID": "000001",
17                              "Description": "The first VM"
18                      }
19              ]
20      }
```

If you're familiar with JSON, this should be easy to parse for you and you'll probably guess most of the things we're going to explain now.

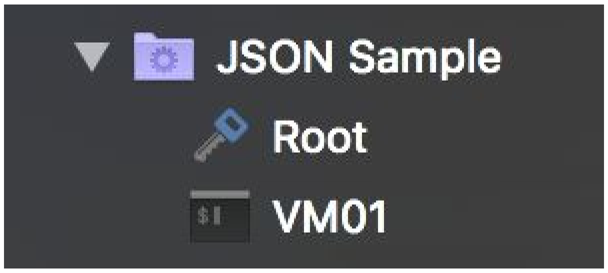
Let's walk over this sample line by line:

1. Start of the JSON document.
2. Start of an array (a collection) of objects.
3. Start of the first object.
4. Specify the type of the object. In this case, a credential.
5. Specify the (display) name of the credential. In this case, "Root".
6. Specify the username of the credential.
7. Specify the password of the credential.
8. Specify the description of the credential.

9. Specify the ID of the credential.
10. End of the credential object.
11. Start of the second object.
12. Specify the type of the object. In this case, a terminal connection.
13. Specify the sub-type of the terminal connection. In this case, an SSH connection.
14. Specify the (display) name of the connection.
15. Specify the computer name (FQDN/IP address/etc.) of the connection.
16. Specify the credential ID that this connection will reference.
17. Specify the description of the connection.
18. End of the connection object.
19. End of the "Objects" array.
20. End of the JSON document

The end result is that we'll have one credential and one SSH connection in Royal TS/X. The SSH connection will have the "Root" credential assigned to it.

Here's what the end result looks like in Royal TSX:



Note: In this case we created a Dynamic Folder in Royal TSX, directly specified the JSON content and named it "JSON Sample".

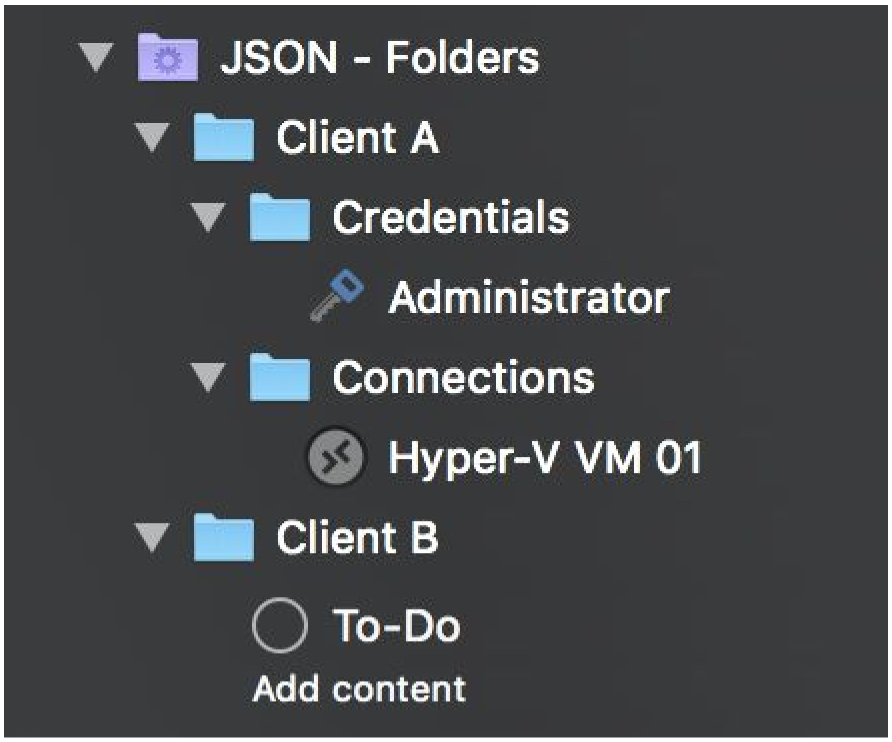
Here are a few things worth pointing out about this sample:

- The root object must always be a JSON object, NOT an array.
  We call this the "RoyalJSONDocument" which currently only has two properties:
  - "Objects": An array of objects (We call those "RoyalJSONObjects").
  - "Name": An optional name which is used as the document name when opening ".rjson" documents.
- The "Type" property is one of only two properties that is required for every RoyalJSONObject. Without it, we wouldn't know which kind of object you're describing.
- The "Name" property is the second required property of RoyalJSONObjects. Just like in Royal TS/X, every object must have a display name.
- If you omit either the type or name property, Royal TS/X will throw an error.
- All other properties are optional.
- Which properties are available depends on the specified object type. For instance, credentials don't have a "Port" property while most connections do.
- If you set a property that isn't available for the specified object type it is ignored.

- The “ID” property is quite magical. ;-) In more serious terminology:
  Royal TS/X internally uses GUIDs to reference objects by ID. Since not all external data sources use these kinds of IDs, we allow RoyalJSON to deal with any type of ID that can be expressed as String. Basically there are two code-paths:
  - When an “ID” is specified as GUID: In this case we directly use the GUID to reference the object. No magic here.
  - When a generic String is used as “ID”: This is the “magic path”. In this case we generate a new internal GUID and take the generic string and store it with the object as “External ID”. If other objects in your rJSON reference this external ID we look up our internal ID and make sure the reference is stored correctly in the resulting object. Also, if you assign objects that you imported through rJSON to other objects outside of your rJSON we will use the internal ID to make the references. Now, if you decide to refresh your rJSON imported objects, we cache the external to internal ID mapping and try to restore it after the import is finished. This ensures that references to objects with external IDs aren’t broken on refresh.
- All of this internal/external ID magic is applied to all properties that take an ID as value.
- RoyalJSON property names are NOT case-sensitive. In other words, `"computerName": "vm01"` is the same as `"ComputerName": "vm01"`.

Building folder hierarchies

By now you should have a good overview of how the rJSON format is structured and probably already toyed around with it a bit. One important topic we haven't touched yet is how folder hierarchies can be expressed in rJSON. Consider the following example:



So, what do we have here? Multiple folders, some of them nested in other folders, each having one child object.

Let's break the individual objects in this sample down into path representation:

- Client A/Credentials/Administrator
- Client A/Connections/Hyper-V VM 01
- Client B/To-Do

So how do we express this in rJSON? Turns out there are two ways:

- The easy but less flexible way
- The harder but more flexible way

The easy (less flexible) method

We’ll start with the easy but less flexible method:

```
{
	"Objects": [
		{
			"Type": "Credential",
			"Name": "Administrator",
			"Username": "administrator",
			"Password": "@he38qewhq82r!",
			"ID": "00001",
			"Path": "Client A/Credentials"
		},
		{
			"Type": "RemoteDesktopConnection",
			"Name": "Hyper-V VM 01",
			"ComputerName": "hyper-v-vm-01",
			"CredentialID": "00001",
			"Path": "Client A/Connections"
		},
		{
			"Type": "ToDo",
			"Name": "To-Do",
			"Description": "Add content",
			"Path": "Client B"
		}
	]
}
```

We’re going to skip most of the content in this sample and focus only on the “Path” properties. Recall the path representation we previously mentioned? Basically we apply this to each of the objects we create in this sample.
For instance, the first object (the “Administrator” credential) has its “Path” set to "Client A/Credentials". This tells our parser to put the credential into a folder named “Credentials” which itself is nested inside another folder named “Client A”.
The remote desktop connection “Hyper-V VM 01” uses the same approach to ensure it lands in a folder named “Connections” which is nested in a folder named “Client A”.
Which leaves us at the last object, the creatively named To-Do item “To-Do”. ;-) This one is not nested in subfolders but instead will end up right in the “Client B” folder.

A few things worth pointing out here:

- The paths are always relative to the dynamic folder or rJSON document. You cannot relocate objects outside of those boundaries using rJSON.

- You can add a leading (root) slash if you want. It will be ignored by the parser. Or in other words: `"Path": "/Client A"` is the same as `"Path": "Client A"`.
- If you're a Windows guy, go ahead and use backslashes instead of those beautiful forward slashes everyone else is using. In other words: `"Path": "Client A\Credentials"` is the same as `"Path": "Client A/Credentials"`.

The hard (more flexible) method

You might be wondering what's wrong with the easy method. It's perfectly sound for most scenarios but is missing one key feature: There's no way to configure the properties of the folders themselves. So that's why we're happy to introduce you to the dark side… I mean, the hard method:

```
{
    "Objects": [
        {
            "Type": "Folder",
            "Name": "Client A",
            "Objects": [
                {
                    "Type": "Folder",
                    "Name": "Credentials",
                    "Objects": [
                        {
                            "Type": "Credential",
                            "Name": "Administrator",
                            "Username": "administrator",
                            "Password": "@he38qewhq82r!",
                            "ID": "00001"
                        }
                    ]
                },
                {
                    "Type": "Folder",
                    "Name": "Connections",
                    "CredentialID": "00001",
                    "Objects": [
                        {
                            "Type": "RemoteDesktopConnection",
                            "Name": "Hyper-V VM 01",
                            "ComputerName": "hyper-v-vm-01",
                            "CredentialsFromParent": true
                        }
                    ]
                }
            ]
        },
        {
            "Type": "Folder",
            "Name": "Client B",
            "Objects": [
                {
                    "Type": "ToDo",
                    "Name": "To-Do",
                    "Description": "Add content"
                }
            ]
        }
    ]
}
```

Now that looks complicated! And we're talking 47 lines vs. 25 for the previous method. Doesn't sound very efficient but there's a very good reason we'd want to use this instead of the easy method.

Consider having more than one connection in the "Client A/Connections" folder. Now if you wanted to assign the same credential to each of those connections, wouldn't it be easier to just assign it to the parent folder and let the connections inherit their parent folder's credentials? Or maybe you want to save some additional notes on the Client folders for documentation purposes? Or assign a Secure Gateway to the folder and let all connections use it… All of these cases make it necessary to be able to specify additional properties for the folder objects. So that's why folder objects are first-class citizens in rJSON and can be created just like other objects.

To create a folder object, just set its "Type" to "Folder" and specify the child objects using the "Objects" property, just like the "Objects" property of the root JSON element (RoyalJSONDocument).
To let objects inherit their credential configuration from their parent, set their "CredentialsFromParent" property to "true".

Note that the "Path" property takes precedence over folder nesting, so if both are used, the object will be placed in the folder pointed to by the "Path" property.

Object references and inheritance

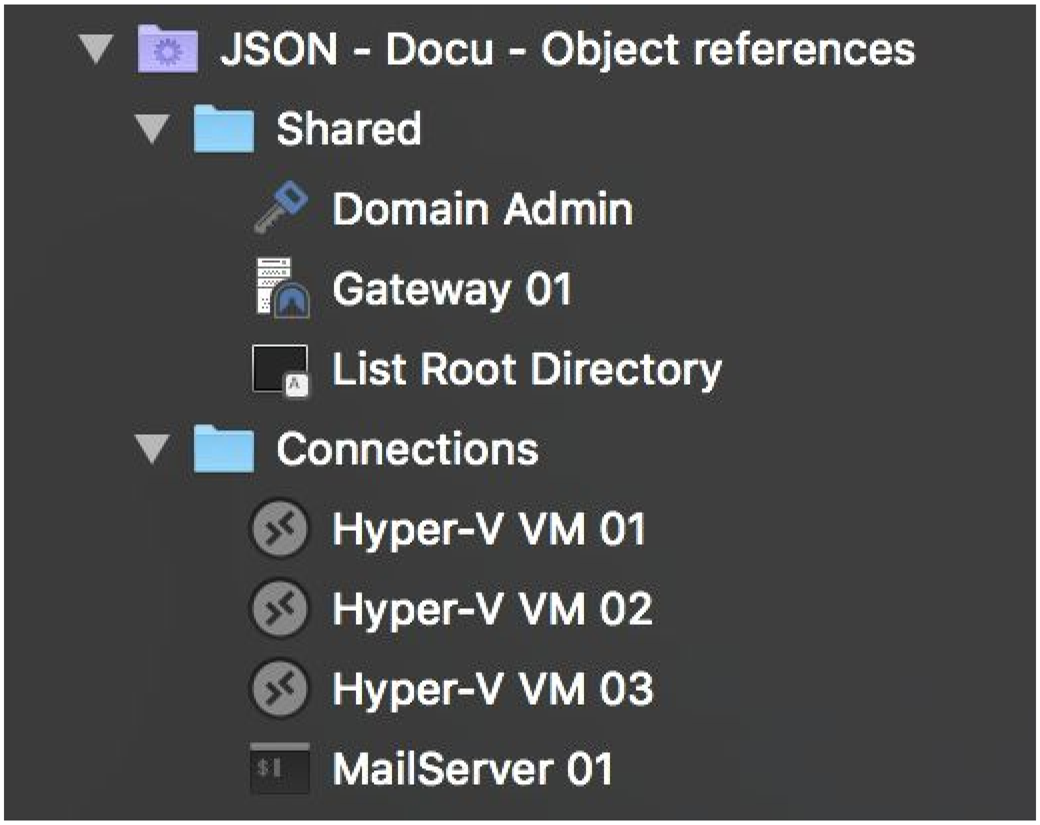
One of the core features of Royal TS/X is to be able to assign objects to other objects, essentially creating object references. This, for instance, makes it possible to use a single credential object for hundreds or thousands of connections. Now, when the password of the credential expires it must only be changed in a single, central place instead of for each connection individually.
While credential objects are the most common example for object references, other objects, like Secure Gateways and command- and key-sequence tasks can be referenced too.

Depending on the object type, multiple different ways to assign the object may be available. Credentials, for instance can be assigned using their ID (In the UI this is called “Use an existing credential”), their name (“Specify a credential name”) or even inherited from their parent folder (“Use credentials from parent folder”). Of course, you can also directly enter your username and password by using the “Specify username and password” mode.
Secure Gateways on the other hand can only be referenced by their ID or inherited from the object’s parent folder. So there are slight differences that you need to be aware of when dealing with object references. The best way to learn those is to study which options are available in the UI for the various object types.

This should give you a quick overview over how object references and inheritance works in Royal TS/X. Now let’s jump right into how this can be expressed in rJSON:

```
{
	"Objects": [
		{
			"Type": "Folder",
			"Name": "Shared",
			"Objects": [
				{
					"Type": "Credential",
					"Name": "Domain Admin",
					"Username": "administrator",
					"Password": "@he38qewhq82r!",
					"ID": "00001"
				},
				{
					"Type": "SecureGateway",
					"Name": "Gateway 01",
					"ComputerName": "gateway01",
					"CredentialID": "00001",
					"ID": "00002"
				},
                {
                    "Type": "KeySequenceTask",
                    "Name": "List Root Directory",
                    "KeySequence": "ls -lh /"
                }
            ],
        },
        {
            "Type": "Folder",
            "Name": "Connections",
            "CredentialID": "00001",
            "SecureGatewayID": "00002",
            "Objects": [
                {
                    "Type": "RemoteDesktopConnection",
                    "Name": "Hyper-V VM 01",
                    "ComputerName": "hyper-v-vm-01",
                    "CredentialsFromParent": true,
                    "SecureGatewayFromParent": true
                },
                {
                    "Type": "RemoteDesktopConnection",
                    "Name": "Hyper-V VM 02",
                    "ComputerName": "hyper-v-vm-02",
                    "CredentialsFromParent": true,
                    "SecureGatewayFromParent": true
                },
                {
                    "Type": "RemoteDesktopConnection",
                    "Name": "Hyper-V VM 03",
                    "ComputerName": "hyper-v-vm-03",
                    "CredentialsFromParent": true,
                    "SecureGatewayFromParent": true
                },
                {
                    "Type": "TerminalConnection",
                    "TerminalConnectionType": "SSH",
                    "Name": "MailServer 01",
                    "ComputerName": "mail-server-01",
                    "Username": "root",
                    "Password": "@(h8e3r12!",
                    "SecureGatewayFromParent": true,
                    "KeySequenceName": "List Root Directory"
                }
            ]
        }
    ]
}
```

Alright, so let’s first look at what this gives us in Royal TSX:



Here’s what we have:

- Two folders (“Shared” and “Connections”)
- One credential (“Domain Admin”)
- One Secure Gateway (“Gateway 01”)
- One key sequence task (“List Root Directory”)
- Three Remote Desktop connections
- One SSH connection

What you don’t get to see in the screenshot is how the various objects are connected to each other. Let’s start with the “Domain Admin” credential.

In line 12 we express that we would like the credential to have the ID “00001”. We can then use this ID to refer to the credential from other objects. In line 31 we set the “CredentialID” property of the “Connections” folder to the ID of our “Domain Admin” credential (ID “00001”).
This basically means: Assign “Domain Admin” to “Connections”.
Now that we have our credential assigned to the “Connections” folder, we want the connections themselves to actually use that credential. To do so, we specify `"CredentialsFromParent": true` in lines 38, 45 and 52. Now all of the three Remote Desktop connections inherit the credentials from their parent folder, “Connections”.

For the “MailServer 01” SSH connection we want to use a different set of credentials and since no other object uses the same credentials, we directly specify the username and password of this connection in lines 60 and 61 using the “Username” and “Password” properties.

That’s it for credential assignment. Because we need a Secure Gateway to connect to each of our remote machines, we also assign the Secure Gateway object (ID “00002”) to the “Connections” folder and let the connections inherit it from their parent folder. In line 32 we specify `"SecureGatewayID": "00002"` on our “Connections” folder. Then, for each of our connections we set `"SecureGatewayFromParent": true` to let them inherit the Secure Gateway configuration from their parent folder.

That’s it for Secure Gateway assignment. Last but not least, we defined a key sequence task which lists the contents of the root directory. The definition of the object starts at line 21. For demo purposes we chose a different approach to assign this task to the SSH connection. Instead of referring to it by its ID, we use its name “List Root Directory”. So in line 63 we have this: `"KeySequenceName": "List Root Directory"`. This means, when the connection is established, launch the key sequence task with the name “List Root Directory”.

Dynamic Credentials

Dynamic Credentials can be useful in cases where it's unfeasible (or just not possible) to retrieve the full dataset of an external credential management system in one go. Instead, you can retrieve just the metadata (Name, ID, etc.) of your credentials and populate the other fields (Username, Password, etc.) on demand, when a connection referencing the credentials is established.

Advantages:

- Fast retrieval of credential objects with the bare minimum of information that can be assigned to connections just like regular credential objects.
- Confidential information can be kept on the remote system until required.
- Can be useful in scenarios where the administrator needs to provide users access to a set of credentials but the end users should not be able to "see" the actual usernames or passwords of those credentials.

Things to consider:

- Properties that aren't pre-populated (Username, Password, etc.) are only available when establishing a connection that references the credential.
- Two distinct calls are required to retrieve the full data set.
- If the dynamic credentials are stored on a remote database, a network connection to that database must be available when establishing connections that reference dynamic credentials.

Dynamic Credentials share the same properties as regular credential objects. That means, the process of creating dynamic credentials is exactly the same as for regular credentials. The difference is that the following properties can (but don't have to) be left empty and fetched on demand by the dedicated "Dynamic Credential Script" of your dynamic folder:

- Username
- Password
- KeyFilePath
- KeyFileContent
- KeyFilePassphrase

Here's an example of how to create a dynamic credential object and assign it to an SSH connection:

```
{
	"Objects": [
		{
			"Type": "DynamicCredential",
5                           "Name": "User 1",
6                           "ID": "Cred01"
7                   },
8                   {
9                           "Type": "TerminalConnection",
10                          "ComputerName": "localhost",
11                          "Name": "Localhost",
12                          "CredentialID": "Cred01"
13                  }
14      }
```

Like previously mentioned, creating dynamic credentials works exactly the same as regular credentials. You can however leave the username, password, etc. properties empty and fill them on demand with the dedicated “Dynamic Credential Script”.

This second script is expected to return a single RoyalJSONDynamicCredential object like in the following example:

```
1       {
2               "Username": "user",
3               "Password": "pass"
4       }
```

So let's see what happens here:

- From line 3 to 7 we define a dynamic credential object named “User 1”. We set its ID to “Cred01”. We don’t specify username or password values.
- From line 8 to 13 we create a new SSH connection and reference the dynamic credential by its ID.
- In the separate dynamic credential script we return a RoyalJSONDynamicCredential object which provides the values for “Username” and “Password”.
- Just before the SSH connection is established, the dedicated dynamic credential script is executed and its return value is used to populate the missing information, namely the username and password of the credential.

Dynamic Folders

Dynamic Folders are the main way to get rJSON into Royal TS and Royal TSX. They allow you to specify scripts that are expected to return rJSON which is then parsed, converted into objects and populated underneath the dynamic folder.

Various script interpreters are supported but availability differs between platforms. For instance, AppleScript is supported on the Mac but unavailable on Windows. VBScript on the other hand is not available on the Mac but well supported under Windows. For testing purposes you can also directly specify JSON. This allows you to “play” with the data format and easily verify if you’re writing valid rJSON.

Currently, two separate scripts can be configured:

- **The “Dynamic Folder Script”:**
  This script retrieves the content that should appear underneath the dynamic folder. Its return value must be a valid RoyalJSONDocument object.
- **The “Dynamic Credential Script”:**
  This script retrieves detailed information (username, password, etc.) of dynamic credentials when a connection referencing a dynamic credential is opened. Its return value must be a valid RoyalJSONDynamicCredential object.

All scripts are expected to return their data in stdout and the exit code must be 0 for a script run to be considered successful. If either of those conditions is not met, an error will be displayed and the script output is discarded. For error handling purposes you may use stderr. If the exit code is not 0 and content is present on stderr, Royal TS/X will display the output of stderr to the user.

To actually execute a dynamic folder script the “Reload” command must be used from the context menu of a dynamic folder in the navigation panel. In the future, additional means of reloading a dynamic folder’s content may be provided.

Tokens

You may already be familiar with tokens in Royal TS/X from other object types, like Command- and Key Sequence Tasks. In the context of dynamic folder scripts, tokens allow you to inject values of the dynamic folder into your scripts. You can, for instance use the `$EffectiveUsername$` and `$EffectivePassword$` tokens to inject credentials into your scripts.

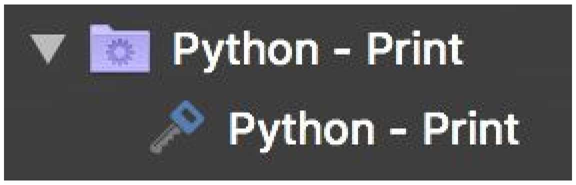
This is useful to ensure no confidential data is stored in plain-text inside your script's content. It also allows you to store parameters of your scripts in a central location instead of somewhere in the scripts.

Here's a simple Python example:

```
1      import json
2
3      jsonStr = json.dumps(
4              {
5                      "Objects": [
6                              {
7                                      "Type": "Credential",
8                                      "Name": "$Name$"
9                              }
10                     ]
11             }
12     )
13
14     print(jsonStr)
```

In line 8 we're injecting the `$Name$` token into the credential's `"Name"` property value.

Here's what this gets you in Royal TSX:



The name of the credential created from rJSON is the same as the dynamic folder's name. Obviously, this isn't quite a real world example but it should get the point over.
Basically it's up to you how/if you want to use tokens in your scripts. If you're for instance calling a web service that requires authentication in your script, it makes sense to not store the credentials directly in your script. Instead, you should save them in the dynamic folder's credential settings. You may even store the URL of your web service endpoint in a custom property instead directly in the script and refer to it using the `$CustomProperty.AlphanumericTitleWithoutSpaces$` token.

Script Interpreters

Royal TS/X ships with support for the most common script interpreters. Among others, we support Bash, Python, Perl and PowerShell. Like previously mentioned, the list of supported script interpreters differs between platforms and will likely expand in the future.

Royal TS/X doesn’t ship with binaries of the supported script interpreters. So in some cases, if the script interpreter of your choice doesn’t ship with the operating system, you may have to install it before using it from Royal TS/X. For example, while PowerShell Core is supported on macOS, it doesn’t ship with the OS and thus is required to be installed before use.

When a dynamic folder script is executed, any tokens contained within the script are replaced by actual values and a temporary file with the full, expanded content is written to disk. On macOS, we prepend the script’s content with the shebang of the selected script interpreter. For example, this is Python’s shebang: `#!/usr/bin/env python`. This ensures that the OS can find the correct script interpreter to execute the script with. After the script finishes execution, the temporary file is deleted.

Advanced scenarios

While rJSON was designed to be easy to use for regular users as well as developers, we also wanted it to be as powerful as you need it to be. For that reason we decided to expose only a small subset of functionality as first-class properties but also provide a way to drop down one level lower, to our RoyalDocument data model.

Because dropping down to the RoyalDocument level requires knowledge of our internal data model, it's generally not recommended to go that route unless you really need to access properties that aren't exposed through rJSON.

The RoyalDocument data model is vastly more complicated to use than rJSON. A good example for demonstrating the complexity is setting up credentials for a connection. While in rJSON, the only thing required to assign a username and password to a connection is to set the two respective properties (`Username` and `Password`). In the RoyalDocument data model however five properties are required: First you need to disable inheritance (`CredentialFromParent`), then you need to set the correct credential mode (`CredentialMode`), you also need to ensure that autologon is enabled (`CredentialAutologon`) and finally you assign the username (`CredentialUsername`) and password (`CredentialPassword`).

Of course not everything is that complicated in the RoyalDocument data model but complexity is often hidden and will only become apparent when something doesn't work as expected. Another reason we do not recommend using the RoyalDocument data model is that it can change at any time and we cannot guarantee a stable API.

So with the disclaimer out of the way, let's look at how you can access the RoyalDocument data model. Here's an example:

```
{
	"Objects": [
		{
			"Type": "TerminalConnection",
			"TerminalConnectionType": "SSH",
			"Name": "VM01",
			"ComputerName": "vm01",
			"Properties": {
				"EnableLoggingOSX": true,
				"LogDirectoryOSX": "~/vm01-logs",
				"RecorderOutputPath": "%USERPROFILE%\\vm01-logs",
				"RecordStartOnConnect": true,
				"PuttyLogging": 1
			}
15                          }
16                  ]
17          }
```

In this example we configure an SSH connection using rJSON and then start dropping down into the RoyalDocument layer starting at line 8. The “Properties” key is used to specify that we intend to set some RoyalDocument properties. Line 9 and 10 are Royal TSX (for macOS) specific and enable logging for the SSH connection and configure the logging directory respectively. Line 11-13 basically do the same for Royal TS (for Windows).
The result of this example is a single SSH connection which is properly configured for logging on both macOS and Windows.

Available properties

RoyalJSONDocument (root element)

Name

Data Type: String
Description: The (display) name of the document if the rJSON is opened via a “.rjson” file. If not specified, the filename is used as the document name instead.
Example: `"Name": "My rJSON Document"`

Objects

Data Type: Array of RoyalJSONObject
Description: An array of objects that will be direct descendants of the dynamic folder or rJSON document.
Example: `"Objects": [ { "Type": "Credential", "Name": "A Credential" } ]`

RoyalJSONObject

Type

Required: Yes
Data Type: Enumeration
Applies To: All object types
Description: The type of the object.
Supported Values:

- Folder
- Credential
- DynamicCredential
- ToDo
- CommandTask
- KeySequenceTask

- SecureGateway
- RoyalServer
- RemoteDesktopGateway
- RemoteDesktopConnection
- TerminalConnection
- WebConnection
- VNCConnection
- FileTransferConnection
- TeamViewerConnection
- ExternalApplicationConnection
- PerformanceConnection
- VMwareConnection
- HyperVConnection
- WindowsEventsConnection
- WindowsServicesConnection
- WindowsProcessesConnection
- TerminalServicesConnection
- PowerShellConnection

Example: `"Type": "Folder"`

Name

Required: Yes
Data Type: String
Applies To: All object types
Description: The (display) name of the object.
Example: `"Name": "My Folder"`

ID

Data Type: String
Applies To: All object types
Description: The ID of the object. This can either be a GUID (formatted as string) or any other generic string. If you pass in a GUID it will be directly used to set the ID of the resulting object. If you pass in any other string, a GUID will be generated for the object and the string you pass in will be stored as external ID.
Supported Values:

- GUID formatted as string
- String

Example: `"ID": "041471cd-1202-40a6-84e3-83f3ceb57a99"`
Example 2: `"ID": "0001"`

ComputerName

Data Type: String
Applies To: All connection types, Credential, Secure Gateway, Royal Server, Remote Desktop Gateway
Description: The computer name (IP address/FQDN) of this object. Can also be used to set the URL of web connections and credentials.
Example: `"ComputerName": "vm01.local"`
Example 2: `"ComputerName": "10.0.0.1"`
Example 3: `"ComputerName": "https://www.royalapplications.com/"`

URL

Data Type: String
Applies To: See ComputerName
Description: Alias for ComputerName.
Example: `"URL": "vm01.local"`
Example 2: `"URL": "10.0.0.1"`
Example 3: `"URL": "https://www.royalapplications.com/"`

Port

Data Type: Integer
Applies To: Most connection types
Description: The port of this object.
Example: `"Port": 5900`

Description

Data Type: String
Applies To: All object types
Description: The description of the object.
Example: `"Description": "Interesting object"`

Favorite

Data Type: Boolean
Applies To: All object types
Description: Specifies whether or not the object is a favorite.
Example: `"Favorite": true`

ColorFromParent

Data Type: Boolean
Applies To: All object types
Description: Specifies whether or not the object gets its color from its parent.
Example: `"ColorFromParent": false`

Color

Data Type: String
Applies To: All object types
Description: The color of the object, specified as Hex (HTML) color (i.e. #FFFFFF for white).
Supported Values:

- Hex Color String

Example: `"Color": "#FF0000"`

IconName

Data Type: String
Applies To: All object types except To-Do
Description: The icon library icon name of the object.
Example: `"IconName": "Flat/Objects/User ID"`

Notes

Data Type: String
Applies To: All object types
Description: The notes of the object.
Example: `"Notes": "Notes support <strong>HTML</strong>"`

Path

Data Type: String
Applies To: All object types
Description: The folder path of the object. This may be used as an alternative to nesting objects in Folder objects. The “Path” property takes precedence over folder nesting, so if both are used, the object will be placed in the folder pointed to by the “Path” property.
Supported Values:

- A folder path either in *NIX (This/Is/A/Test) or in Windows (This\Is\A\Test) format

Example: `"Path": "Path/To/Object"`
Example 2: `"Path": "Path\To\Object"`

CustomField1

Data Type: String
Applies To: All object types except To-Do
Description: Custom Field 1 (non-protected).
Supported since: Royal TS 5.1, Royal TSX 4.1.2
Example: `"CustomField1": "Text in this field won’t be encrypted"`

CustomField2

Data Type: String
Applies To: All object types except To-Do
Description: Custom Field 2 (non-protected).
Supported since: Royal TS 5.1, Royal TSX 4.1.2
Example: `"CustomField2": "Text in this field won’t be encrypted"`

CustomField3

Data Type: String
Applies To: All object types except To-Do
Description: Custom Field 3 (non-protected).
Supported since: Royal TS 5.1, Royal TSX 4.1.2
Example: `"CustomField3": "Text in this field won’t be encrypted"`

CustomField4

Data Type: String
Applies To: All object types except To-Do
Description: Custom Field 4 (non-protected).
Supported since: Royal TS 5.1, Royal TSX 4.1.2
Example: `"CustomField4": "Text in this field won’t be encrypted"`

CustomField5

Data Type: String
Applies To: All object types except To-Do
Description: Custom Field 5 (non-protected).

Supported since: Royal TS 5.1, Royal TSX 4.1.2
Example: `"CustomField5": "Text in this field won’t be encrypted"`

CustomField6

Data Type: String
Applies To: All object types except To-Do
Description: Custom Field 6 (non-protected).
Supported since: Royal TS 5.1, Royal TSX 4.1.2
Example: `"CustomField6": "Text in this field won’t be encrypted"`

CustomField7

Data Type: String
Applies To: All object types except To-Do
Description: Custom Field 7 (non-protected).
Supported since: Royal TS 5.1, Royal TSX 4.1.2
Example: `"CustomField7": "Text in this field won’t be encrypted"`

CustomField8

Data Type: String
Applies To: All object types except To-Do
Description: Custom Field 8 (non-protected).
Supported since: Royal TS 5.1, Royal TSX 4.1.2
Example: `"CustomField8": "Text in this field won’t be encrypted"`

CustomField9

Data Type: String
Applies To: All object types except To-Do
Description: Custom Field 9 (non-protected).
Supported since: Royal TS 5.1, Royal TSX 4.1.2
Example: `"CustomField9": "Text in this field won’t be encrypted"`

CustomField10

Data Type: String
Applies To: All object types except To-Do
Description: Custom Field 10 (non-protected).

Supported since: Royal TS 5.1, Royal TSX 4.1.2
Example: `"CustomField10": "Text in this field won’t be encrypted"`

CustomField11

Data Type: String
Applies To: All object types except To-Do
Description: Custom Field 11 (protected).
Supported since: Royal TS 5.1, Royal TSX 4.1.2
Example: `"CustomField11": "Text in this field will be encrypted"`

CustomField12

Data Type: String
Applies To: All object types except To-Do
Description: Custom Field 12 (protected).
Supported since: Royal TS 5.1, Royal TSX 4.1.2
Example: `"CustomField12": "Text in this field will be encrypted"`

CustomField13

Data Type: String
Applies To: All object types except To-Do
Description: Custom Field 13 (protected).
Supported since: Royal TS 5.1, Royal TSX 4.1.2
Example: `"CustomField13": "Text in this field will be encrypted"`

CustomField14

Data Type: String
Applies To: All object types except To-Do
Description: Custom Field 14 (protected).
Supported since: Royal TS 5.1, Royal TSX 4.1.2
Example: `"CustomField14": "Text in this field will be encrypted"`

CustomField15

Data Type: String
Applies To: All object types except To-Do
Description: Custom Field 15 (protected).

Supported since: Royal TS 5.1, Royal TSX 4.1.2
Example: `"CustomField15": "Text in this field will be encrypted"`

CustomField16

Data Type: String
Applies To: All object types except To-Do
Description: Custom Field 16 (protected).
Supported since: Royal TS 5.1, Royal TSX 4.1.2
Example: `"CustomField16": "Text in this field will be encrypted"`

CustomField17

Data Type: String
Applies To: All object types except To-Do
Description: Custom Field 17 (protected).
Supported since: Royal TS 5.1, Royal TSX 4.1.2
Example: `"CustomField17": "Text in this field will be encrypted"`

CustomField18

Data Type: String
Applies To: All object types except To-Do
Description: Custom Field 18 (protected).
Supported since: Royal TS 5.1, Royal TSX 4.1.2
Example: `"CustomField18": "Text in this field will be encrypted"`

CustomField19

Data Type: String
Applies To: All object types except To-Do
Description: Custom Field 19 (protected).
Supported since: Royal TS 5.1, Royal TSX 4.1.2
Example: `"CustomField19": "Text in this field will be encrypted"`

CustomField20

Data Type: String
Applies To: All object types except To-Do
Description: Custom Field 20 (protected).

Supported since: Royal TS 5.1, Royal TSX 4.1.2
Example: `"CustomField20": "Text in this field will be encrypted"`

Properties

Data Type: Key/Value pairs
Applies To: All object types
Description: Can be used to specify properties of the object that aren't exposed in the RoyalJSON object model. To be able to use this, knowledge of the RoyalDocument data model is required. Documentation is available here.
Example: `"Properties": { "IsExpanded": "True" }`

CustomProperties

Data Type: Key/Value pairs
Alternative Data Type: Array of RoyalJSONCustomProperty
Applies To: All object types
Description: The custom properties of the object. You can either pass in key/value pairs or an array of objects of type RoyalJSONCustomProperty. If using key/value pairs, the key will be the name of the custom property and the value will be its value. All custom properties specified that way will be of type “Text”. It’s not possible to specify a different type when using key/value pairs. If using an array of objects instead, specify the name of the custom property using "Name", the value using "Value" and the type using "Type" (Supported types are: Text, Protected, URL, Email, Address, Phone, Date, YesNo, Header)
Example: `"CustomProperties": { "PIN Code": "1234" }`
Alternate Example: `"CustomProperties": [ { "Name": "PIN Code", "Value": "1234", "Type": "Protected" } ]`

CredentialsFromParent

Data Type: Boolean
Applies To: Most object types that support credentials
Description: Specifies whether or not to use the credentials of the parent folder.
Example: `"CredentialsFromParent": true`

CredentialsFromContextConnection

Data Type: Boolean
Applies To: Remote Desktop Gateway, Command Task
Description: Specifies whether or not to use the credentials of the context connection.

Example: `"CredentialsFromContextConnection": false`

CredentialID

Data Type: String
Applies To: All object types that support credentials
Description: The ID of the credential that is assigned to this object. This can either be a GUID (formatted as string) or any other generic string. If you pass in a GUID it will be directly used to set the Credential ID of the resulting object. No validation is performed in this case. If you pass in any other string, the external to internal ID mapping table will be searched for a match. If no match is found, the property will be ignored. Otherwise the internal ID will be used to reference the credential.
Supported Values:

- GUID formatted as string
- String

Example: `"CredentialID": "041471cd-1202-40a6-84e3-83f3ceb57a99"`
Example 2: `"CredentialID": "0001"`

CredentialName

Data Type: String
Applies To: Most object types that support credentials
Description: The name of the credential that is assigned to this object.
Example: `"CredentialName": "Administrator"`

Username

Data Type: String
Applies To: All object types that support credentials, Credential
Description: The username of this object.
Example: `"Username": "administrator"`

Password

Data Type: String
Applies To: All object types that support credentials, Credential
Description: The password of this object.
Example: `"Password": "!dfhe8yr498hh@"`

KeyFilePath

Data Type: String
Applies To: All object types that support private key files, Credential
Description: The private key file path of this object.
Example: `"KeyFilePath": "/Path/To/Key/File"`
Example 2: `"KeyFilePath": "c:\Path\To\Key\File"`

KeyFileContent

Data Type: String
Applies To: All object types that support private key files, Credential
Description: The content of the private key file of this object.
Example: `"KeyFileContent": "-----BEGIN RSA PRIVATE KEY----- ..."`

KeyFilePassphrase

Data Type: String
Applies To: All object types that support private key files, Credential
Description: The passphrase of this object's private key file.
Example: `"KeyFilePassphrase": "^234hsd9344hf@"`

AutoFillElements

Data Type: Array of RoyalJSONAutoFillElement
Applies To: Credential, Web connection
Description: The auto fill elements for the credential or web connection.
Supported since: Royal TS 5.1, Royal TSX 4.1.2
Example: `"AutoFillElements": [ { "Element": "#username", "Action": "Fill", "Value": "$$EffectiveUsername$$" } ]`

AutoFillDelay

Data Type: Integer
Applies To: Credential, Web connection
Description: The delay in milliseconds before the auto fill script is executed.
Supported since: Royal TS 5.1, Royal TSX 4.1.2
Example: `"AutoFillDelay": 200`

RetryAutoFillUntilSuccess

Data Type: Boolean
Applies To: Credential, Web connection
Description: Whether or not auto fill is retried until all fields have been found.
Supported since: Royal TS 5.1, Royal TSX 4.1.2
Example: `"RetryAutoFillUntilSuccess": true`

Objects

Data Type: Array of RoyalJSONObject
Applies To: Folder
Description: An array of objects that will be direct descendants of the current folder.
Example: `"Objects": [ { "Type": "Credential", "Name": "A Credential" } ]`

Status

Data Type: Enumeration
Applies To: To-Do
Description: The status of this To-Do item.
Supported Values:

- Active
- Completed

Example: `"Status": "Completed"`

NoConfirmationRequired

Data Type: Boolean
Applies To: Key Sequence Task, Command Task
Description: Specifies whether or not user confirmation is required to run this task.
Example: `"NoConfirmationRequired": true`

CommandMac

Data Type: String
Applies To: Command Task
Description: The command this task executes on macOS.
Example: `"CommandMac": "ping"`

ArgumentsMac

Data Type: String
Applies To: Command Task
Description: The arguments required to run this task on macOS.
Example: `"ArgumentsMac": "$URI$"`

WorkingDirectoryMac

Data Type: String
Applies To: Command Task
Description: The working directory of this task for macOS.
Example: `"WorkingDirectoryMac": "/Path/To/Working/Directory"`

CommandWindows

Data Type: String
Applies To: Command Task
Description: The command this task executes on Windows.
Example: `"CommandWindows": "ping"`

ArgumentsWindows

Data Type: String
Applies To: Command Task
Description: The arguments required to run this task on Windows.
Example: `"ArgumentsWindows": "$URI$"`

WorkingDirectoryWindows

Data Type: String
Applies To: Command Task
Description: The working directory of this task for Windows.
Example: `"WorkingDirectoryWindows": "c:\Path\To\Working/Directory"`

KeySequence

Data Type: String
Applies To: Key Sequence Task, All connection types, Folder

Description: The key sequence that will be executed before the connection is established if set on a connection or a folder. The key sequence that the key sequence task executes if set on a key sequence task object.
Example: `"KeySequence": "{WAIT:5000}abc{ENTER}"`

UseSSL

Data Type: Boolean
Applies To: VMware connection, Royal Server
Description: Specifies whether or not to use SSL for this object.
Example: `"UseSSL": "true"`

SecureGatewayPort

Data Type: Integer
Applies To: Secure Gateway, Royal Server
Description: The Secure Gateway port.
Example: `"Port": 22`

SecureGatewayCredentialID

Data Type: String
Applies To: Secure Gateway, Royal Server
Description: The ID of the credential that is assigned to the Secure Gateway. This can either be a GUID (formatted as string) or any other generic string. If you pass in a GUID it will be directly used to set the Secure Gateway Credential ID of the resulting object. No validation is performed in this case. If you pass in any other string, the external to internal ID mapping table will be searched for a match. If no match is found, the property will be ignored. Otherwise the internal ID will be used to reference the credential.
Supported Values:

- GUID formatted as string
- String

Example: `"SecureGatewayCredentialID": "041471cd-1202-40a6-84e3-83f3ceb57a99"`
Example 2: `"SecureGatewayCredentialID": "0001"`

SecureGatewayCredentialName

Data Type: String
Applies To: Secure Gateway, Royal Server

Description: The name of the credential that is assigned to the Secure Gateway.
Example: `"SecureGatewayCredentialName": "root"`

SecureGatewayUsername

Data Type: String
Applies To: Secure Gateway, Royal Server
Description: The username of the Secure Gateway.
Example: `"SecureGatewayUsername": "root"`

SecureGatewayPassword

Data Type: String
Applies To: Secure Gateway, Royal Server
Description: The password of the Secure Gateway.
Example: `"SecureGatewayPassword": "!dfhe8yr498hh@"`

RemoteDesktopGatewayComputerName

Data Type: String
Applies To: Remote Desktop connection, Remote Desktop Gateway
Description: The computer name (IP address/FQDN) of the Remote Desktop Gateway.
Example: `"RemoteDesktopGatewayComputerName": "rdg-01.local"`
Example 2: `"RemoteDesktopGatewayComputerName": "10.0.1.5"`

RemoteDesktopGatewayCredentialsFromConnection

Data Type: Boolean
Applies To: Remote Desktop connection, Remote Desktop Gateway
Description: Specifies whether or not to use the credentials of the connection for the Remote Desktop Gateway.
Example: `"RemoteDesktopGatewayCredentialsFromConnection": false`

RemoteDesktopGatewayCredentialID

Data Type: String
Applies To: Remote Desktop connection, Remote Desktop Gateway
Description: The ID of the credential that is assigned to the Remote Desktop Gateway. This can either be a GUID (formatted as string) or any other generic string. If you pass in a GUID it will be directly used to set the Remote Desktop Gateway Credential ID of the resulting object.

No validation is performed in this case. If you pass in any other string, the external to internal ID mapping table will be searched for a match. If no match is found, the property will be ignored. Otherwise the internal ID will be used to reference the credential.
Supported Values:
- GUID formatted as string
- String

Example: `"RemoteDesktopGatewayCredentialID": "041471cd-1202-40a6-84e3-83f3ceb57a99"`
Example 2: `"RemoteDesktopGatewayCredentialID": "0001"`

RemoteDesktopGatewayCredentialName

Data Type: String
Applies To: Remote Desktop connection, Remote Desktop Gateway
Description: The name of the credential that is assigned to the Remote Desktop Gateway.
Example: `"RemoteDesktopGatewayCredentialName": "Administrator"`

RemoteDesktopGatewayUsername

Data Type: String
Applies To: Remote Desktop connection, Remote Desktop Gateway
Description: The username of the Remote Desktop Gateway.
Example: `"RemoteDesktopGatewayUsername": "administrator"`

RemoteDesktopGatewayPassword

Data Type: String
Applies To: Remote Desktop connection, Remote Desktop Gateway
Description: The password of the Remote Desktop Gateway.
Example: `"RemoteDesktopGatewayPassword": "!dfhe8yr498hh@"`

ConnectTaskFromParent

Data Type: Boolean
Applies To: All connection types, Folder
Description: Specifies whether or not to use the connect task from the parent folder of this object.
Example: `"ConnectTaskFromParent": false`

ConnectTaskID

Data Type: String
Applies To: All connection types, Folder
Description: The ID of the command task that is to be assigned to this object as connect task. This can either be a GUID (formatted as string) or any other generic string. If you pass in a GUID it will be directly used to set the Connect Task ID of the resulting object. No validation is performed in this case. If you pass in any other string, the external to internal ID mapping table will be searched for a match. If no match is found, the property will be ignored. Otherwise the internal ID will be used to reference the credential.
Supported Values:

- GUID formatted as string
- String

Example: `"ConnectTaskID": "041471cd-1202-40a6-84e3-83f3ceb57a99"`
Example 2: `"ConnectTaskID": "0001"`

ConnectTaskName

Data Type: String
Applies To: All connection types, Folder
Description: The name of the command task that is to be assigned to this object as connect task.
Example: `"ConnectTaskName": "root"`

DisconnectTaskFromParent

Data Type: Boolean
Applies To: All connection types, Folder
Description: Specifies whether or not to use the disconnect task from the parent folder of this object.
Example: `"DisconnectTaskFromParent": false`

DisconnectTaskID

Data Type: String
Applies To: All connection types, Folder
Description: The ID of the command task that is to be assigned to this object as disconnect task. This can either be a GUID (formatted as string) or any other generic string. If you pass in a GUID it will be directly used to set the Disconnect Task ID of the resulting object. No validation is performed in this case. If you pass in any other string, the external to internal ID mapping

table will be searched for a match. If no match is found, the property will be ignored. Otherwise the internal ID will be used to reference the credential.
Supported Values:

- GUID formatted as string
- String

Example: `"DisconnectTaskID": "041471cd-1202-40a6-84e3-83f3ceb57a99"`
Example 2: `"DisconnectTaskID": "0001"`

DisconnectTaskName

Data Type: String
Applies To: All connection types, Folder
Description: The name of the command task that is to be assigned to this object as disconnect task.
Example: `"DisconnectTaskName": "root"`

KeySequenceFromParent

Data Type: Boolean
Applies To: All connection types, Folder
Description: Specifies whether or not to use the connect key sequence from the parent folder of this object.
Example: `"KeySequenceFromParent": false`

KeySequenceID

Data Type: String
Applies To: All connection types, Folder
Description: The ID of the key sequence task that is to be assigned to this object as connect key sequence. This can either be a GUID (formatted as string) or any other generic string. If you pass in a GUID it will be directly used to set the Key Sequence ID of the resulting object. No validation is performed in this case. If you pass in any other string, the external to internal ID mapping table will be searched for a match. If no match is found, the property will be ignored. Otherwise the internal ID will be used to reference the credential.
Supported Values:

- GUID formatted as string
- String

Example: `"KeySequenceID": "041471cd-1202-40a6-84e3-83f3ceb57a99"`
Example 2: `"KeySequenceID": "0001"`

KeySequenceName

Data Type: String
Applies To: All connection types, Folder
Description: The name of the key sequence task that is to be assigned to this object as connect key sequence.
Example: `"KeySequenceName": "root"`

SecureGatewayFromParent

Data Type: Boolean
Applies To: All connection types that support Secure Gateways, Folder
Description: Specifies whether or not to use the Secure Gateway from the parent folder of this object.
Example: `"SecureGatewayFromParent": false`

SecureGatewayID

Data Type: String
Applies To: All connection types that support Secure Gateways, Folder
Description: The ID of the Secure Gateway that is to be assigned to this object. This can either be a GUID (formatted as string) or any other generic string. If you pass in a GUID it will be directly used to set the Secure Gateway ID of the resulting object. No validation is performed in this case. If you pass in any other string, the external to internal ID mapping table will be searched for a match. If no match is found, the property will be ignored. Otherwise the internal ID will be used to reference the credential.
Supported Values:

- GUID formatted as string
- String

Example: `"SecureGatewayID": "041471cd-1202-40a6-84e3-83f3ceb57a99"`
Example 2: `"SecureGatewayID": "0001"`

RoyalServerFromParent

Data Type: Boolean
Applies To: All connection types that support Royal Server, Folder
Description: Specifies whether or not to use the Royal Server from the parent folder of this object.
Example: `"RoyalServerFromParent": false`

RoyalServerID

Data Type: String
Applies To: All connection types that support Royal Server, Folder
Description: The ID of the Royal Server that is to be assigned to this object. This can either be a GUID (formatted as string) or any other generic string. If you pass in a GUID it will be directly used to set the Royal Server ID of the resulting object. No validation is performed in this case. If you pass in any other string, the external to internal ID mapping table will be searched for a match. If no match is found, the property will be ignored. Otherwise the internal ID will be used to reference the credential.
Supported Values:

- GUID formatted as string
- String

Example: `"RoyalServerID": "041471cd-1202-40a6-84e3-83f3ceb57a99"`
Example 2: `"RoyalServerID": "0001"`

RemoteDesktopGatewayFromParent

Data Type: Boolean
Applies To: Remote Desktop connection, Folder
Description: Specifies whether or not to use the Remote Desktop Gateway from the parent folder of this object.
Example: `"RemoteDesktopGatewayFromParent": true`

RemoteDesktopGatewayID

Data Type: String
Applies To: Remote Desktop connection, Folder
Description: The ID of the Remote Desktop Gateway that is to be assigned to this object. This can either be a GUID (formatted as string) or any other generic string. If you pass in a GUID it will be directly used to set the Remote Desktop Gateway ID of the resulting object. No validation is performed in this case. If you pass in any other string, the external to internal ID mapping table will be searched for a match. If no match is found, the property will be ignored. Otherwise the internal ID will be used to reference the credential.
Supported Values:

- GUID formatted as string
- String

Example: `"RemoteDesktopGatewayID":`
`"041471cd-1202-40a6-84e3-83f3ceb57a99"`
Example 2: `"RemoteDesktopGatewayID": "0001"`

RemoteDesktopGatewayUsage

Data Type: Enumeration
Applies To: Remote Desktop connection, Folder
Description: The usage mode of the Remote Desktop Gateway.
Supported Values:

- Never
- Always
- OnDemand

Example: `"RemoteDesktopGatewayUsage": "Always"`

MACAddress

Data Type: String
Applies To: Most connection types
Description: The physical (MAC) address of this connection.
Example: `"MACAddress": "FF:FF:FF:FF:FF:FF"`

WindowMode

Data Type: Enumeration
Applies To: Most connection types
Description: The window mode of this connection.
Supported Values:

- Embedded
- External
- FullScreen

Example: `"WindowMode": "External"`

ConsoleSession

Data Type: Boolean
Applies To: Remote Desktop connection
Description: Specifies whether or not to connect to the console/admin session of this Remote Desktop connection.
Example: `"ConsoleSession": true`

NLA

Data Type: Boolean

Applies To: Remote Desktop connection
Description: Specifies whether or not to use NLA (Network Level Authentication) for this Remote Desktop connection.
Example: `"NLA": true`

ResizeMode

Data Type: Enumeration
Applies To: Remote Desktop connection
Description: The resize mode of this Remote Desktop connection.
Supported Values:

- ScrollBars
- SmartSizing
- SmartReconnect

Example: `"ResizeMode": "SmartReconnect"`

HyperVMode

Data Type: Enumeration
Applies To: Remote Desktop connection
Description: The Hyper-V mode of this Remote Desktop connection.
Supported Values:

- None
- HyperVHost
- HyperVInstance

Example: `"HyperVMode": "HyperVInstance"`

HyperVPort

Data Type: Integer
Applies To: Remote Desktop connection
Description: The Hyper-V port of this Remote Desktop Connection.
Example: `"HyperVPort": 2179`

HyperVInstanceID

Data Type: String
Applies To: Remote Desktop connection
Description: The Hyper-V Instance ID (VM ID) of this Remote Desktop connection.
Example: `"HyperVInstanceID": "12345"`

UseEnhancedHyperVSessionMode

Data Type: Boolean
Applies To: Remote Desktop connection
Description: Specifies whether or not to use enhanced Hyper-V session mode for this Remote Desktop connection.
Example: `"UseEnhancedHyperVSessionMode": false`

TerminalConnectionType

Data Type: Enumeration
Applies To: Terminal connection
Description: The connection type of this terminal connection.
Supported Values:

- SSH
- Telnet
- SerialPort
- CustomTerminal

Example: `"TerminalConnectionType": "SSH"`

SerialPortName

Data Type: String
Applies To: Terminal connection
Description: The serial port name of this terminal connection.
Example: `"SerialPortName": "COM1"`

BaudRate

Data Type: Integer
Applies To: Terminal connection
Description: The baud rate of the serial port connection.
Supported since: Royal TS 5.1, Royal TSX 4.1.2
Example: `"BaudRate": 9600`

DataBits

Data Type: Integer
Applies To: Terminal connection

Description: The standard length of data bits per byte of the serial port connection.
Supported Values: 5 through 8
Supported since: Royal TS 5.1, Royal TSX 4.1.2
Example: `"DataBits": 8`

Parity

Data Type: Enumeration
Applies To: Terminal connection
Description: The parity bit for the serial port connection.
Supported Values:
- None
- Odd
- Even
- Mark
- Space

Supported since: Royal TS 5.1, Royal TSX 4.1.2
Example: `"Parity": "None"`

StopBits

Data Type: Enumeration
Applies To: Terminal connection
Description: The number of stop bits used for the serial port connection.
Supported Values:
- None
- One
- Two
- OnePointFive

Supported since: Royal TS 5.1, Royal TSX 4.1.2
Example: `"StopBits": "One"`

CustomCommand

Data Type: String
Applies To: Terminal connection
Description: The custom command of this terminal connection.
Example: `"CustomCommand": "ls -lh /"`

FileTransferConnectionType

Data Type: Enumeration
Applies To: File Transfer connection
Description: The connection type of this file transfer connection.
Supported Values:

- FTP
- FTPWithImplicitSSL
- FTPWithExplicitTLSSSL
- SFTP
- SCP

Example: `"FileTransferConnectionType": "SFTP"`

PassiveMode

Data Type: Boolean
Applies To: File Transfer connection
Description: Specifies whether or not to use passive mode for this FTP file transfer connection.
Example: `"PassiveMode": true`

TeamViewerConnectionType

Data Type: Enumeration
Applies To: TeamViewer connection
Description: The connection type of this TeamViewer connection.
Supported Values:

- RemoteControl
- FileTransfer
- Meeting
- VPN
- ManagementConsole
- PromptForType

Example: `"TeamViewerConnectionType": "SFTP"`

UseCIM

Data Type: Boolean
Applies To: Some management connections
Description: Specifies whether or not to use CIM for this management connection.
Example: `"UseCIM": true`

Script

Data Type: String
Applies To: PowerShell connection
Description: The script of this PowerShell connection.
Example: `"Script": "Write-Host \"Hello World\""`

RoyalJSONCustomProperty

Type

Required: Yes
Data Type: Enumeration
Description: The type of the custom property.
Supported Values:

- Text
- Protected
- URL
- Email
- Address
- Phone
- Date
- YesNo
- Header

Example: `"Type": "Text"`

Name

Required: Yes
Data Type: String
Description: The (display) name of the custom property.
Example: `"Name": "Credit Card Number"`

Value

Data Type: String or Boolean
Description: The value of the custom property.
Example: `"Value": "123456789"`
Example 2: `"Value": true`

RoyalJSONAutoFillElement

Element

Required: Yes
Data Type: String
Description: The selector for the auto fill element.
Example: `"Element": "#username"`

Action

Required: Yes
Data Type: Enumeration
Description: The type of action to perform on this auto fill element.
Supported Values:

- Fill
- Click
- Focus

Example: `"Action": "Fill"`

Value

Data Type: String or Boolean
Description: The value to fill the element with.
Applies to: Auto fill elements with the "Action" set to "Fill".
Example: `"Value": "$$EffectiveUsername$$"`

RoyalJSONDynamicCredential

Username

Data Type: String
Description: The username of this object.
Example: `"Username": "administrator"`

Password

Data Type: String
Description: The password of this object.
Example: `"Password": "!dfhe8yr498hh@"`

KeyFilePath

Data Type: String
Description: The private key file path of this object.
Example: `"KeyFilePath": "/Path/To/Key/File"`
Example 2: `"KeyFilePath": "c:\Path\To\Key\File"`

KeyFileContent

Data Type: String
Description: The content of the private key file of this object.
Example: `"KeyFileContent": "-----BEGIN RSA PRIVATE KEY----- ..."`

KeyFilePassphrase

Data Type: String
Description: The passphrase of this object's private key file.
Example: `"KeyFilePassphrase": "^234hsd9344hf@"`

JSON Schema

Here's the JSON schema to validate your JSON content against:

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "title": "RoyalJSONDocument",
  "type": "object",
  "additionalProperties": false,
  "properties": {
    "Name": {
      "type": [
        "null",
        "string"
      ]
    },
    "Objects": {
      "type": [
        "array",
        "null"
      ],
      "items": {
        "$ref": "#/definitions/RoyalJSONObject"
      }
    }
  },
  "definitions": {
    "RoyalJSONObject": {
      "type": "object",
      "additionalProperties": false,
      "required": [
        "Type",
        "Name"
      ],
      "properties": {
        "Type": {
          "description": "The type of the object.",
          "oneOf": [
            {
              "$ref": "#/definitions/RoyalJSONObjectTypes"
            }
          ]
        },
        "Name": {
          "type": "string",
          "description": "The (display) name of the object."
        },
        "ID": {
          "type": [
            "null",
            "string"
          ],
          "description": "The ID of the object. This can either be a GUID (formatted
as string) or any other generic string. If you pass in a GUID it will be directly used
to set the ID of the resulting object. If you pass in any other string, a GUID will be
generated for the object and the string you pass in will be stored as external ID."
        },
        "ComputerName": {
          "type": [
            "null",
            "string"
          ],
          "description": "The computer name (IP address/FQDN) of this object. Can also
be used to set the URL of web connections and credentials."
        },
        "URL": {
          "type": [
            "null",
            "string"
          ],
          "description": "Alias for ComputerName."
        },
        "Port": {
          "type": "integer",
          "description": "The port of this object.",
          "format": "int32"
        },
        "Description": {
          "type": [
            "null",
            "string"
          ],
          "description": "The description of the object."
        },
        "Favorite": {
          "type": "boolean",
          "description": "Specifies whether or not the object is a favorite."
        },
        "ColorFromParent": {
          "type": "boolean",
          "description": "Specifies whether or not the object gets its color from its
parent."
        },
        "Color": {
          "type": [
            "null",
            "string"
          ],
          "description": "The color of the object, specified as Hex (HTML) color (i.e.
#FFFFFF for white)."
        },
        "IconName": {
          "type": [
            "null",
            "string"
          ],
          "description": "The icon library icon name of the object."
        },
        "Notes": {
          "type": [
            "null",
            "string"
          ],
          "description": "The notes of the object."
        },
        "Path": {
          "type": [
            "null",
            "string"
          ],
          "description": "The folder path of the object. This may be used as an
alternative to nesting objects in Folder objects. The \"Path\" property takes
precedence over folder nesting, so if both are used, the object will be placed in the
folder pointed to by the \"Path\" property."
        },
        "CustomField1": {
          "type": [
            "null",
            "string"
          ],
          "description": "Custom Field 1 (non-protected)."
        },
        "CustomField2": {
          "type": [
            "null",
            "string"
          ],
          "description": "Custom Field 2 (non-protected)."
        },
        "CustomField3": {
          "type": [
            "null",
            "string"
          ],
          "description": "Custom Field 3 (non-protected)."
        },
        "CustomField4": {
```

```
      "type": [
        "null",
        "string"
      ],
      "description": "Custom Field 4 (non-protected)."
    },
    "CustomField5": {
      "type": [
        "null",
        "string"
      ],
      "description": "Custom Field 5 (non-protected)."
    },
    "CustomField6": {
      "type": [
        "null",
        "string"
      ],
      "description": "Custom Field 6 (non-protected)."
    },
    "CustomField7": {
      "type": [
        "null",
        "string"
      ],
      "description": "Custom Field 7 (non-protected)."
    },
    "CustomField8": {
      "type": [
        "null",
        "string"
      ],
      "description": "Custom Field 8 (non-protected)."
    },
    "CustomField9": {
      "type": [
        "null",
        "string"
      ],
      "description": "Custom Field 9 (non-protected)."
    },
    "CustomField10": {
      "type": [
        "null",
        "string"
      ],
      "description": "Custom Field 10 (non-protected)."
    },
    "CustomField11": {
      "type": [
      "null",
      "string"
    ],
    "description": "Custom Field 11 (protected)."
  },
  "CustomField12": {
    "type": [
      "null",
      "string"
    ],
    "description": "Custom Field 12 (protected)."
  },
  "CustomField13": {
    "type": [
      "null",
      "string"
    ],
    "description": "Custom Field 13 (protected)."
  },
  "CustomField14": {
    "type": [
      "null",
      "string"
    ],
    "description": "Custom Field 14 (protected)."
  },
  "CustomField15": {
    "type": [
      "null",
      "string"
    ],
    "description": "Custom Field 15 (protected)."
  },
  "CustomField16": {
    "type": [
      "null",
      "string"
    ],
    "description": "Custom Field 16 (protected)."
  },
  "CustomField17": {
    "type": [
      "null",
      "string"
    ],
    "description": "Custom Field 17 (protected)."
  },
  "CustomField18": {
    "type": [
      "null",
            "string"
          ],
          "description": "Custom Field 18 (protected)."
        },
        "CustomField19": {
          "type": [
            "null",
            "string"
          ],
          "description": "Custom Field 19 (protected)."
        },
        "CustomField20": {
          "type": [
            "null",
            "string"
          ],
          "description": "Custom Field 20 (protected)."
        },
        "Properties": {
          "type": [
            "null",
            "object"
          ],
          "description": "Can be used to specify properties of the object that aren't
exposed in the RoyalJSON object model. To be able to use this, knowledge of the
RoyalDocument data model is required. Documentation is available here.",
          "additionalProperties": {
            "type": "string"
          }
        },
        "CustomProperties": {
          "type": [
            "array",
            "null"
          ],
          "description": "The custom properties of the object. You can either pass in
key/value pairs or an array of objects of type RoyalJSONCustomProperty. If using
key/value pairs, the key will be the name of the custom property and the value will be
its value. All custom properties specified that way will be of type “Text”. It’s not
possible to specify a different type when using key/value pairs. If using an array of
objects instead, specify the name of the custom property using \"Name\", the value
using \"Value\" and the type using \"Type\" (Supported types are: Text, Protected,
URL, Email, Address, Phone, Date, YesNo, Header)",
          "items": {
            "$ref": "#/definitions/RoyalJSONCustomProperty"
          }
        },
        "CredentialsFromParent": {
          "type": "boolean"
        },
"CredentialsFromContextConnection": {
  "type": "boolean"
},
"CredentialID": {
  "type": [
    "null",
    "string"
  ]
},
"CredentialName": {
  "type": [
    "null",
    "string"
  ]
},
"Username": {
  "type": [
    "null",
    "string"
  ]
},
"Password": {
  "type": [
    "null",
    "string"
  ]
},
"KeyFilePath": {
  "type": [
    "null",
    "string"
  ]
},
"KeyFileContent": {
  "type": [
    "null",
    "string"
  ]
},
"KeyFilePassphrase": {
  "type": [
    "null",
    "string"
  ]
},
"AutoFillElements": {
  "type": [
    "array",
    "null"
  ],
          "description": "The auto fill elements for the credential or web
connection.",
          "items": {
            "$ref": "#/definitions/RoyalJSONAutoFillElement"
          }
        },
        "AutoFillDelay": {
          "type": "integer",
          "description": "The delay in milliseconds before the auto fill script is
executed.",
          "format": "int32",
          "default": 200
        },
        "RetryAutoFillUntilSuccess": {
          "type": "boolean",
          "description": "Whether or not auto fill is retried until all fields have
been found."
        },
        "Objects": {
          "type": [
            "array",
            "null"
          ],
          "items": {
            "$ref": "#/definitions/RoyalJSONObject"
          }
        },
        "ScriptInterpreter": {
          "type": [
            "null",
            "string"
          ]
        },
        "DynamicCredentialScriptInterpreter": {
          "type": [
            "null",
            "string"
          ]
        },
        "DynamicCredentialScript": {
          "type": [
            "null",
            "string"
          ]
        },
        "Status": {
          "$ref": "#/definitions/RoyalJSONToDoStatus"
        },
        "NoConfirmationRequired": {
          "type": "boolean"
```

```
},
"CommandMac": {
  "type": [
    "null",
    "string"
  ]
},
"ArgumentsMac": {
  "type": [
    "null",
    "string"
  ]
},
"WorkingDirectoryMac": {
  "type": [
    "null",
    "string"
  ]
},
"CommandWindows": {
  "type": [
    "null",
    "string"
  ]
},
"ArgumentsWindows": {
  "type": [
    "null",
    "string"
  ]
},
"WorkingDirectoryWindows": {
  "type": [
    "null",
    "string"
  ]
},
"KeySequence": {
  "type": [
    "null",
    "string"
  ]
},
"UseSSL": {
  "type": "boolean",
  "default": true
},
"SecureGatewayPort": {
  "type": "integer",
  "format": "int32"
```

```
},
"SecureGatewayCredentialID": {
  "type": [
    "null",
    "string"
  ]
},
"SecureGatewayCredentialName": {
  "type": [
    "null",
    "string"
  ]
},
"SecureGatewayUsername": {
  "type": [
    "null",
    "string"
  ]
},
"SecureGatewayPassword": {
  "type": [
    "null",
    "string"
  ]
},
"MACAddress": {
  "type": [
    "null",
    "string"
  ]
},
"WindowMode": {
  "$ref": "#/definitions/RoyalJSONWindowModes"
},
"ConsoleSession": {
  "type": "boolean"
},
"NLA": {
  "type": "boolean",
  "default": true
},
"ResizeMode": {
  "$ref": "#/definitions/RoyalJSONRemoteDesktopResizeModes"
},
"HyperVMode": {
  "$ref": "#/definitions/RoyalJSONRemoteDesktopHyperVModes"
},
"HyperVPort": {
  "type": "integer",
  "format": "int32"
```

```
},
"HyperVInstanceID": {
  "type": [
    "null",
    "string"
  ]
},
"UseEnhancedHyperVSessionMode": {
  "type": "boolean"
},
"RemoteDesktopGatewayComputerName": {
  "type": [
    "null",
    "string"
  ]
},
"RemoteDesktopGatewayCredentialsFromConnection": {
  "type": "boolean"
},
"RemoteDesktopGatewayUsername": {
  "type": [
    "null",
    "string"
  ]
},
"RemoteDesktopGatewayPassword": {
  "type": [
    "null",
    "string"
  ]
},
"RemoteDesktopGatewayCredentialID": {
  "type": [
    "null",
    "string"
  ]
},
"RemoteDesktopGatewayCredentialName": {
  "type": [
    "null",
    "string"
  ]
},
"TerminalConnectionType": {
  "$ref": "#/definitions/RoyalJSONTerminalConnectionTypes"
},
"SerialPortName": {
  "type": [
    "null",
    "string"
```

```
          ]
        },
        "BaudRate": {
          "type": "integer",
          "format": "int32"
        },
        "DataBits": {
          "type": "integer",
          "format": "int32",
          "default": 8
        },
        "Parity": {
          "$ref": "#/definitions/RoyalJSONSerialPortParity"
        },
        "StopBits": {
          "default": "One",
          "oneOf": [
            {
              "$ref": "#/definitions/RoyalJSONSerialPortStopBits"
            }
          ]
        },
        "CustomCommand": {
          "type": [
            "null",
            "string"
          ]
        },
        "FileTransferConnectionType": {
          "$ref": "#/definitions/RoyalJSONFileTransferConnectionTypes"
        },
        "PassiveMode": {
          "type": "boolean"
        },
        "TeamViewerConnectionType": {
          "$ref": "#/definitions/RoyalJSONTeamViewerConnectionTypes"
        },
        "UseCIM": {
          "type": "boolean"
        },
        "Script": {
          "type": [
            "null",
            "string"
          ]
        },
        "ConnectTaskFromParent": {
          "type": "boolean"
        },
        "ConnectTaskID": {
```

```
  "type": [
    "null",
    "string"
  ]
},
"ConnectTaskName": {
  "type": [
    "null",
    "string"
  ]
},
"DisconnectTaskFromParent": {
  "type": "boolean"
},
"DisconnectTaskID": {
  "type": [
    "null",
    "string"
  ]
},
"DisconnectTaskName": {
  "type": [
    "null",
    "string"
  ]
},
"KeySequenceFromParent": {
  "type": "boolean"
},
"KeySequenceID": {
  "type": [
    "null",
    "string"
  ]
},
"KeySequenceName": {
  "type": [
    "null",
    "string"
  ]
},
"SecureGatewayFromParent": {
  "type": "boolean"
},
"SecureGatewayID": {
  "type": [
    "null",
    "string"
  ]
},
        "RoyalServerFromParent": {
          "type": "boolean"
        },
        "RoyalServerID": {
          "type": [
            "null",
            "string"
          ]
        },
        "RemoteDesktopGatewayFromParent": {
          "type": "boolean"
        },
        "RemoteDesktopGatewayUsage": {
          "$ref": "#/definitions/RoyalJSONRemoteDesktopGatewayUsage"
        },
        "RemoteDesktopGatewayID": {
          "type": [
            "null",
            "string"
          ]
        }
      }
    },
    "RoyalJSONObjectTypes": {
      "type": "string",
      "description": "",
      "x-enumNames": [
        "__Unknown",
        "Folder",
        "DynamicFolder",
        "Credential",
        "DynamicCredential",
        "ToDo",
        "CommandTask",
        "KeySequenceTask",
        "SecureGateway",
        "RoyalServer",
        "RemoteDesktopGateway",
        "RemoteDesktopConnection",
        "TerminalConnection",
        "WebConnection",
        "VNCConnection",
        "FileTransferConnection",
        "TeamViewerConnection",
        "ExternalApplicationConnection",
        "PerformanceConnection",
        "VMwareConnection",
        "HyperVConnection",
        "WindowsEventsConnection",
        "WindowsServicesConnection",
    "WindowsProcessesConnection",
    "TerminalServicesConnection",
    "PowerShellConnection"
  ],
  "enum": [
    "__Unknown",
    "Folder",
    "DynamicFolder",
    "Credential",
    "DynamicCredential",
    "ToDo",
    "CommandTask",
    "KeySequenceTask",
    "SecureGateway",
    "RoyalServer",
    "RemoteDesktopGateway",
    "RemoteDesktopConnection",
    "TerminalConnection",
    "WebConnection",
    "VNCConnection",
    "FileTransferConnection",
    "TeamViewerConnection",
    "ExternalApplicationConnection",
    "PerformanceConnection",
    "VMwareConnection",
    "HyperVConnection",
    "WindowsEventsConnection",
    "WindowsServicesConnection",
    "WindowsProcessesConnection",
    "TerminalServicesConnection",
    "PowerShellConnection"
  ]
},
"RoyalJSONCustomProperty": {
  "type": "object",
  "additionalProperties": false,
  "required": [
    "Name",
    "Type"
  ],
  "properties": {
    "Name": {
      "type": "string"
    },
    "Type": {
      "$ref": "#/definitions/RoyalJSONCustomPropertyTypes"
    },
    "Value": {
      "oneOf": [
        {},
        {
          "type": "null"
        }
      ]
    }
  }
},
"RoyalJSONCustomPropertyTypes": {
  "type": "string",
  "description": "",
  "x-enumNames": [
    "Text",
    "Protected",
    "URL",
    "Email",
    "Address",
    "Phone",
    "Date",
    "YesNo",
    "Header"
  ],
  "enum": [
    "Text",
    "Protected",
    "URL",
    "Email",
    "Address",
    "Phone",
    "Date",
    "YesNo",
    "Header"
  ]
},
"RoyalJSONAutoFillElement": {
  "type": "object",
  "additionalProperties": false,
  "required": [
    "Element"
  ],
  "properties": {
    "Element": {
      "type": "string",
      "description": "The selector for the auto fill element."
    },
    "Action": {
      "description": "The type of action to perform on this auto fill element.",
      "oneOf": [
        {
          "$ref": "#/definitions/RoyalJSONAutoFillElementActions"
        }
        ]
      },
      "Value": {
        "type": [
          "null",
          "string"
        ],
        "description": "The value to fill the element with."
      }
    }
  },
  "RoyalJSONAutoFillElementActions": {
    "type": "string",
    "description": "",
    "x-enumNames": [
      "Fill",
      "Click",
      "Focus"
    ],
    "enum": [
      "Fill",
      "Click",
      "Focus"
    ]
  },
  "RoyalJSONToDoStatus": {
    "type": "string",
    "description": "",
    "x-enumNames": [
      "Active",
      "Completed"
    ],
    "enum": [
      "Active",
      "Completed"
    ]
  },
  "RoyalJSONWindowModes": {
    "type": "string",
    "description": "",
    "x-enumNames": [
      "Embedded",
      "External",
      "FullScreen"
    ],
    "enum": [
      "Embedded",
      "External",
      "FullScreen"
    ]
```

```
},
"RoyalJSONRemoteDesktopResizeModes": {
  "type": "string",
  "description": "",
  "x-enumNames": [
    "ScrollBars",
    "SmartSizing",
    "SmartReconnect"
  ],
  "enum": [
    "ScrollBars",
    "SmartSizing",
    "SmartReconnect"
  ]
},
"RoyalJSONRemoteDesktopHyperVModes": {
  "type": "string",
  "description": "",
  "x-enumNames": [
    "None",
    "HyperVHost",
    "HyperVInstance"
  ],
  "enum": [
    "None",
    "HyperVHost",
    "HyperVInstance"
  ]
},
"RoyalJSONTerminalConnectionTypes": {
  "type": "string",
  "description": "",
  "x-enumNames": [
    "SSH",
    "Telnet",
    "SerialPort",
    "CustomTerminal"
  ],
  "enum": [
    "SSH",
    "Telnet",
    "SerialPort",
    "CustomTerminal"
  ]
},
"RoyalJSONSerialPortParity": {
  "type": "string",
  "description": "",
  "x-enumNames": [
    "None",
    "Odd",
    "Even",
    "Mark",
    "Space"
  ],
  "enum": [
    "None",
    "Odd",
    "Even",
    "Mark",
    "Space"
  ]
},
"RoyalJSONSerialPortStopBits": {
  "type": "string",
  "description": "",
  "x-enumNames": [
    "None",
    "One",
    "Two",
    "OnePointFive"
  ],
  "enum": [
    "None",
    "One",
    "Two",
    "OnePointFive"
  ]
},
"RoyalJSONFileTransferConnectionTypes": {
  "type": "string",
  "description": "",
  "x-enumNames": [
    "FTP",
    "FTPWithImplicitSSL",
    "FTPWithExplicitTLSSSL",
    "SFTP",
    "SCP"
  ],
  "enum": [
    "FTP",
    "FTPWithImplicitSSL",
    "FTPWithExplicitTLSSSL",
    "SFTP",
    "SCP"
  ]
},
"RoyalJSONTeamViewerConnectionTypes": {
  "type": "string",
  "description": "",
        "x-enumNames": [
          "RemoteControl",
          "FileTransfer",
          "Meeting",
          "VPN",
          "ManagementConsole",
          "PromptForType"
        ],
        "enum": [
          "RemoteControl",
          "FileTransfer",
          "Meeting",
          "VPN",
          "ManagementConsole",
          "PromptForType"
        ]
      },
      "RoyalJSONRemoteDesktopGatewayUsage": {
        "type": "string",
        "description": "",
        "x-enumNames": [
          "Never",
          "Always",
          "OnDemand"
        ],
        "enum": [
          "Never",
          "Always",
          "OnDemand"
        ]
      }
    }
}

{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "title": "RoyalJSONDynamicCredential",
  "type": "object",
  "additionalProperties": false,
  "properties": {
    "Username": {
      "type": [
        "null",
        "string"
      ]
    },
    "Password": {
      "type": [
        "null",
        "string"
        ]
      },
      "KeyFilePath": {
        "type": [
          "null",
          "string"
        ]
      },
      "KeyFileContent": {
        "type": [
          "null",
          "string"
        ]
      },
      "KeyFilePassphrase": {
        "type": [
          "null",
          "string"
        ]
      }
    }
  }
```