

Table of Contents

RDT v1.0.0

Introduction & Overview

Introduction

Release Notes

RDT Settings

How To

Install and Uninstall RDT

Data Hydration

Legion of Laptops

RPS Provisioning

iPXE

Export CMDB

RDT v1.0.0

Last updated on March 28, 2022.

Last Reviewed and Approved on PENDING REVIEW

Introduction

RDT is an RPS specific tool that makes the hydrating and provisioning RPS processes user-friendly. RDT leverages the powerful RPS API and overlays an easy-to-use user interface to simplify interacting with RPS. It allows users the functionality of hydrating an RPS database, provisioning RPS on various nodes, and importing RPS data, all within one simple user interface.

i NOTE

RDT v1.0.0 is dependent on RPS v4.0.0 API.

App

RDT is a Windows Presentation Foundation(WPF) app . It is a standalone Windows application and is installed on the machine on which it is run. It is expected to be installed alongside a valid RPS installation. For more information about how to install and uninstall RDT, please refer to [How to Install and Uninstall RDT](#).

Use Cases

RDT supports two main use cases: Data Hydration and RPS Provisioning.

Data Hydration

RDT supports data hydration via the Data Hydration page. On this screen, users will be able to import templates in order to hydrate the CMDB. For more information on the Data Hydration page, see [Data Hydration](#).

RPS Provisioning

RDT supports RPS provisioning across two different screens: RPS Provisioning and Legion of Laptops.

In the RPS Provisioning screen, users can: select task maps, assign them to a node, and run the task map while monitoring the output on a console. For more information, see [RPS Provisioning](#).

Using Legion of Laptops, users can import RPS data from a previous export of the RPS database. For more information, see [Legion of Laptops](#).

Definitions

CMDB: Configuration Management DataBase

RPS: Rapid Provisioning System

API: Application Programming Interface

RDT Release Notes

Last updated on March 28, 2022.

Last Reviewed and Approved on PENDING REVIEW

What's New in 1.0.0 (January 31, 2022)

Added

- Added data hydration functionality
 - Allows a user to use instance definitions and data mapping to hydrate a CMDB with confirmation data.
- Added provisioning functionality
 - Using the RDTActions.json config file, a user can create actions and use those actions to assign task maps to targets. The user can also use RPS Task management to provision the targets with detailed status and logs.
- Added Legion of Laptops functionality
 - Allows a user to import data into the CMDB from an RPS XML export file.
- Added Export CMDB functionality
 - Allows a user to export data in RPS XML export file.

Known Issues

- Some icons may not be shown correctly to the user in a deployed environment. This issue does not effect functionality.

RDT Settings

Last updated on March 28, 2022.

Last Reviewed and Approved on PENDING REVIEW

Introduction

RDT contains many settings designed to be configured by the LSI. These settings are in the appsettings.json file at the installed location of RDT. The typical path for this file is: `C:\Program Files\RPS\RDT\`.

The table below contains all settings that are configurable by the LSI.

⚠ IMPORTANT

Any setting in the appsettings.json file that is not listed in [Table 1](#) should **not** be modified.

Settings

Table 1: RDT Settings

SETTING	DESCRIPTION	EXAMPLE
ActionFilePath	The path to the RDT Actions JSON file used to configure the provisioning page.	"C:\ContentStore\RDT\RDTActions.json"
HydrationConfigFilePath	The path to the RDT Hydration Config JSON used to configure the inputs for the data hydration page.	"C:\ContentStore\RDT\DataHydrationConfig.json"
InstanceDefFolderPath	The path to the instance definitions used for data hydration.	"C:\ContentStore\RDT\InstanceDefs"
DataMappingFolderPath	The path to the data mapping JSON files used for data hydration.	"C:\ContentStore\RDT\DataMappings"
Logging → LogLevel → Default	The log level of what is logged by default. See Table 2 for more information about log levels.	"Information"

Table 2: Log Levels

LEVEL	USAGE
Verbose	Verbose is the noisiest level, and rarely (if ever) enabled for a production app.
Debug	Debug is used for internal system events that are not necessarily observable from the outside, but useful when determining how something happened.
Information	Information events describe things happening in the system that correspond to its responsibilities and functions. Generally, these are the observable actions the system can perform.
Warning	When service is degraded, endangered, or behaving outside of its expected parameters, Warning level events are used.
Error	When functionality is unavailable or expectations are broken, an Error event is used.

LEVEL	USAGE
Fatal	The most critical level. Fatal events demand immediate attention.

How to Install and Uninstall RDT

Last updated on March 28, 2022.

Last Reviewed and Approved on PENDING REVIEW

Installation

The installation file, "RDT.Installer.msi", is located within `C:\ContentStore` on the local machine.

1. Double click on the installation file.
2. Click Next to proceed on Welcome Screen.

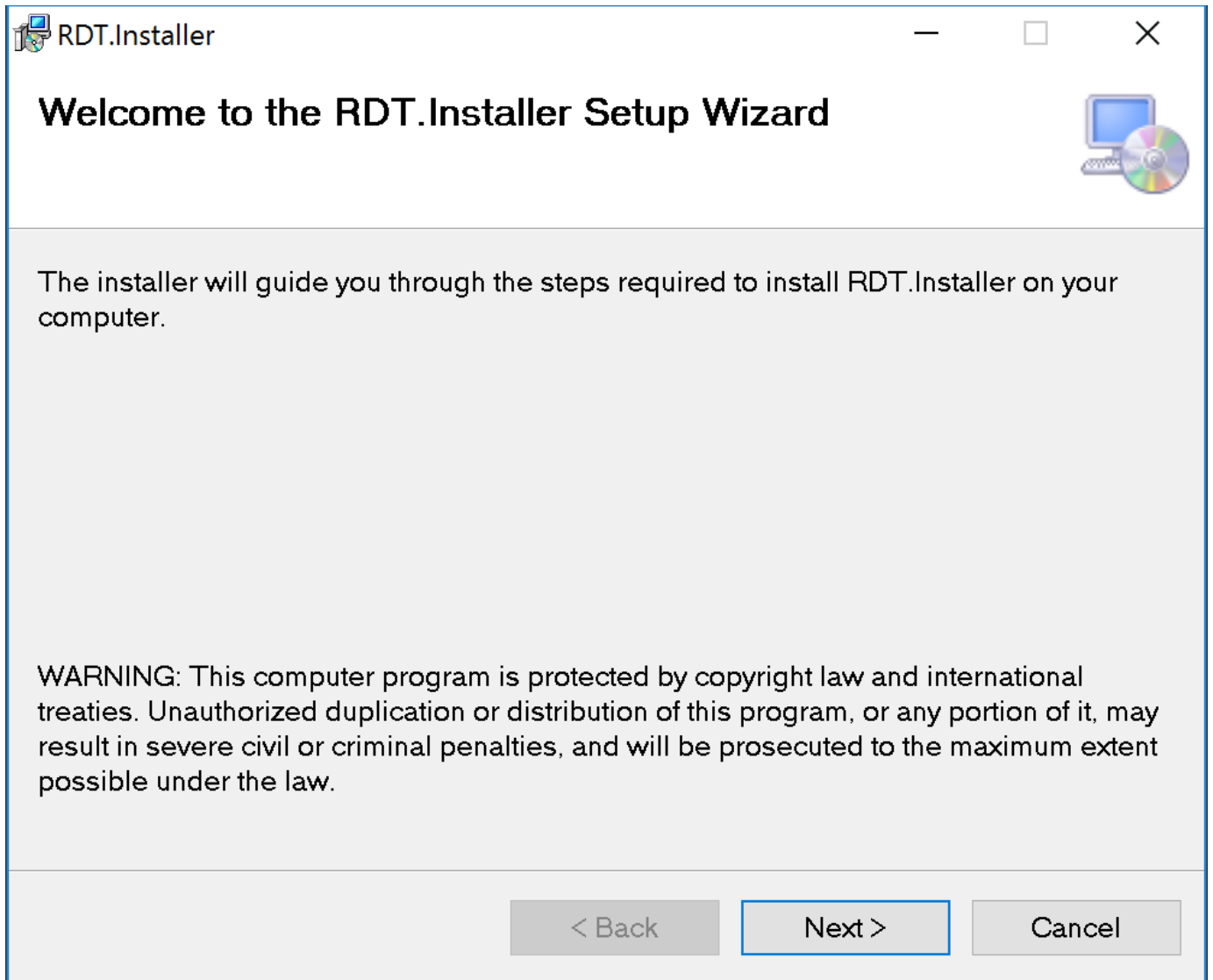


Figure 1: Installation screen.

3. Select Installation folder

Select Installation Folder



The installer will install RDT.Installer to the following folder.

To install in this folder, click "Next". To install to a different folder, enter it below or click "Browse".

Folder:

Install RDT.Installer for yourself, or for anyone who uses this computer:

Everyone

Just me

Figure 1: Select Installation Folder.

4. Confirm Installation

Confirm Installation



The installer is ready to install RDT.Installer on your computer.

Click "Next" to start the installation.

< Back

Next >

Cancel

Figure 2: Confirm installation.

5. Click on close to finish installation.

Installation Complete



RDT.Installer has been successfully installed.

Click "Close" to exit.

< Back

Close

Cancel

Figure 3: Complete installation.

Application Location

The RDT application, "RDT.exe", is located at `C:\Program Files (x86)\RPS\RDT` or at the location selected in [step 3 of Installation process](#). There is a shortcut created for the RDT on the desktop.

Uninstallation

The RDT application can be uninstalled via [Windows Settings](#) or using the [uninstallation file](#).

Uninstallation via Windows Settings

1. Navigate to "Settings", "Apps & features", and search for "RDT".

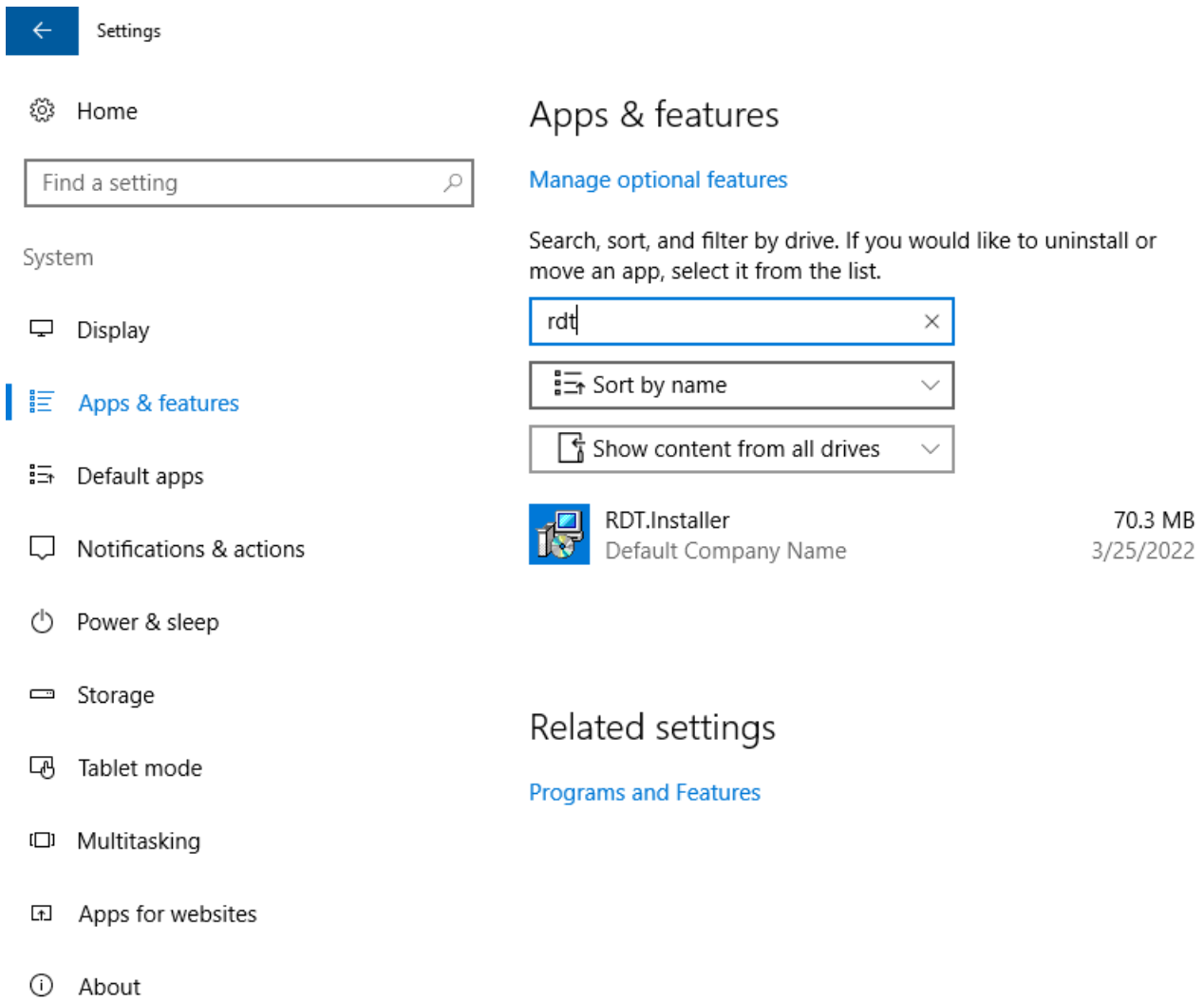


Figure 3: Search for "RDT" in "Apps & features" to uninstall RDT.

2. Select the RDT application and click "Uninstall".

Uninstallation via Uninstallation File

RDT can also be uninstalled by running RpsInstaller.msi file.

1. Double click on the uninstallation file.
2. Select to remove RDT option and Click Finish on the confirmation prompt.

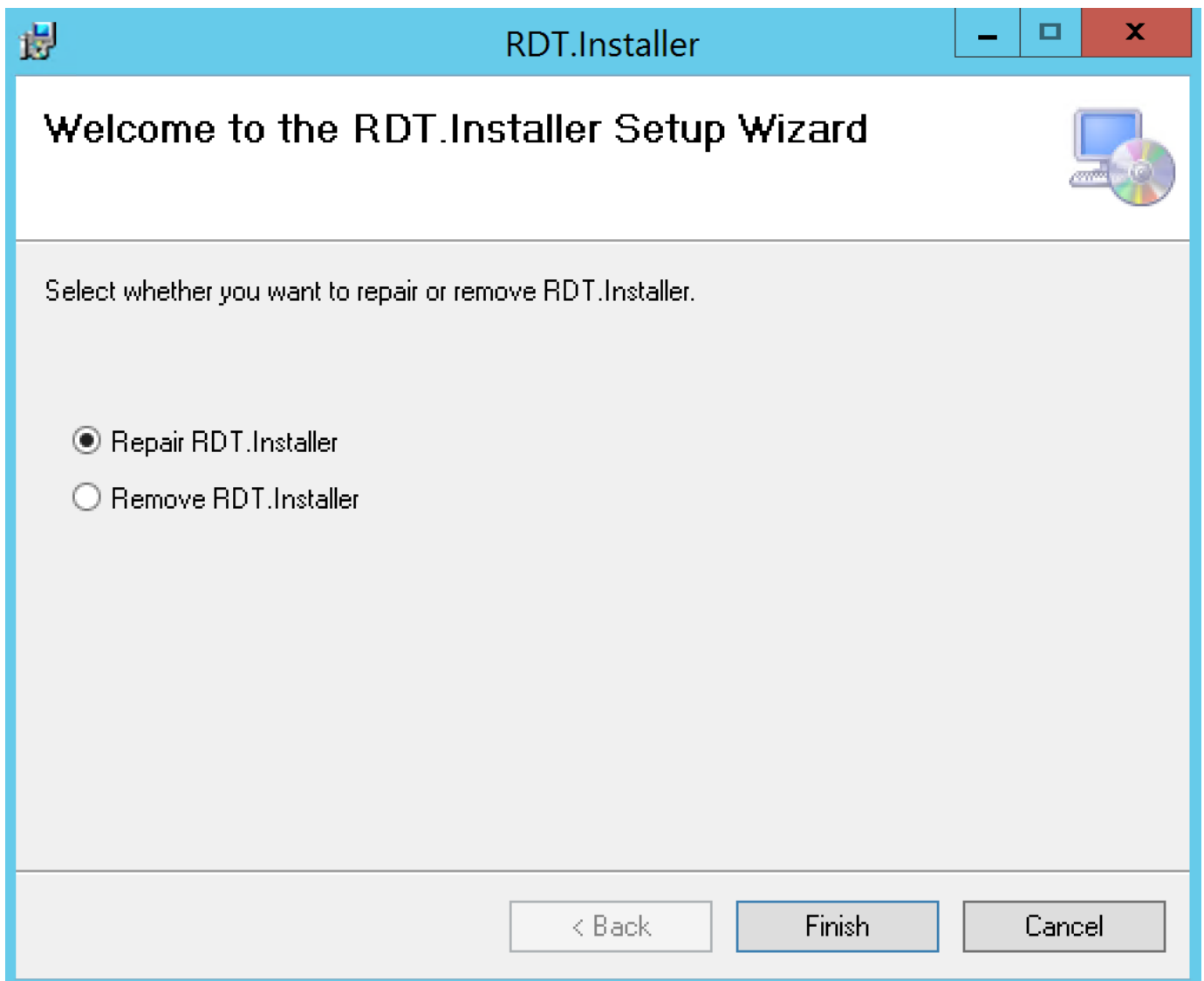


Figure 4: Click Finish to proceed with uninstallation.

3. Click close to complete the uninstallation process.

Data Hydration

Last updated on March 28, 2022.

Last Reviewed and Approved on PENDING REVIEW

Introduction

Data Hydration is a service within the RDT, which provides the ability to hydrate the CMDB. This service is intended to be used before the provisioning process, to import existing data from a previous installation of RPS.

Usage

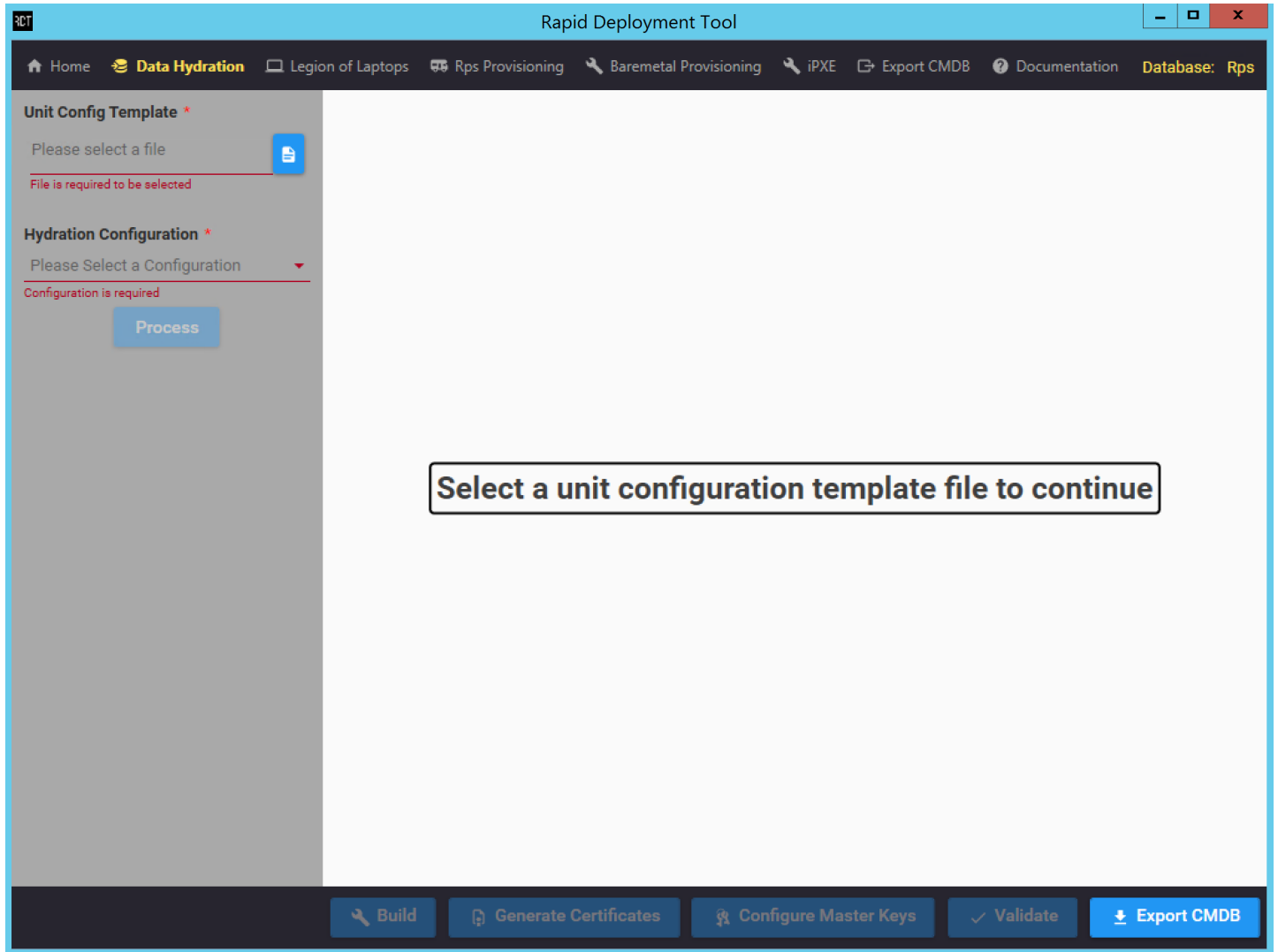


Figure 1: This is the initial view the user sees when first navigating to the Data Hydration page.

The specific fields present on the screen is dynamically populated from the Data Hydration Configuration file, located in the ContentStore\RDT\DataHydrationConfig.json.

Inputs

In order for the RPS data to be hydrated, the user must select an Instance Definition file containing the definitions they would like to hydrate. This is done via the Unit Config template input. Additionally, the user must supply the files containing the data which should be hydrated for the listed Instance Definitions as per the Data Hydration Configuration.

The Unit Config template input supports .xlsx, .xls, and .csv file types. The other inputs support filetypes as listed in the Data Hydration Configuration file.

Once the Unit Config template file has been uploaded, the table on the right displays the data read in from the file. See Figure 2 for an example of this with a single Instance Definition:

The screenshot shows the Rapid Deployment Tool interface. The top navigation bar includes 'Home', 'Data Hydration', 'Legion of Laptops', 'Rps Provisioning', 'Baremetal Provisioning', 'iPXE', 'Export CMDB', and 'Documentation'. The 'Database' is set to 'Rps'. On the left, there is a 'Unit Config Template' section with 'UnitData.csv' and a 'Hydration Configuration' section with a dropdown menu and a 'Process' button. The main area displays a table titled 'Unit Configurations' with the following data:

SystemType	TelephonyId	CIName	Nomenclature	Parent	UnitName
Provision	101	Provision	Provision.InstanceDefinition		Provision

At the bottom, there are buttons for 'Build', 'Generate Certificates', 'Configure Master Keys', 'Validate', and 'Export CMDB'.

Figure 2: An example of a populated Instance Definition table. In this case, there is only 1 Instance Definition in the uploaded Unit Config template.

Starting the Hydration Process

Once all inputs have been selected, press the Process button to read in the data and prepare for the hydration.

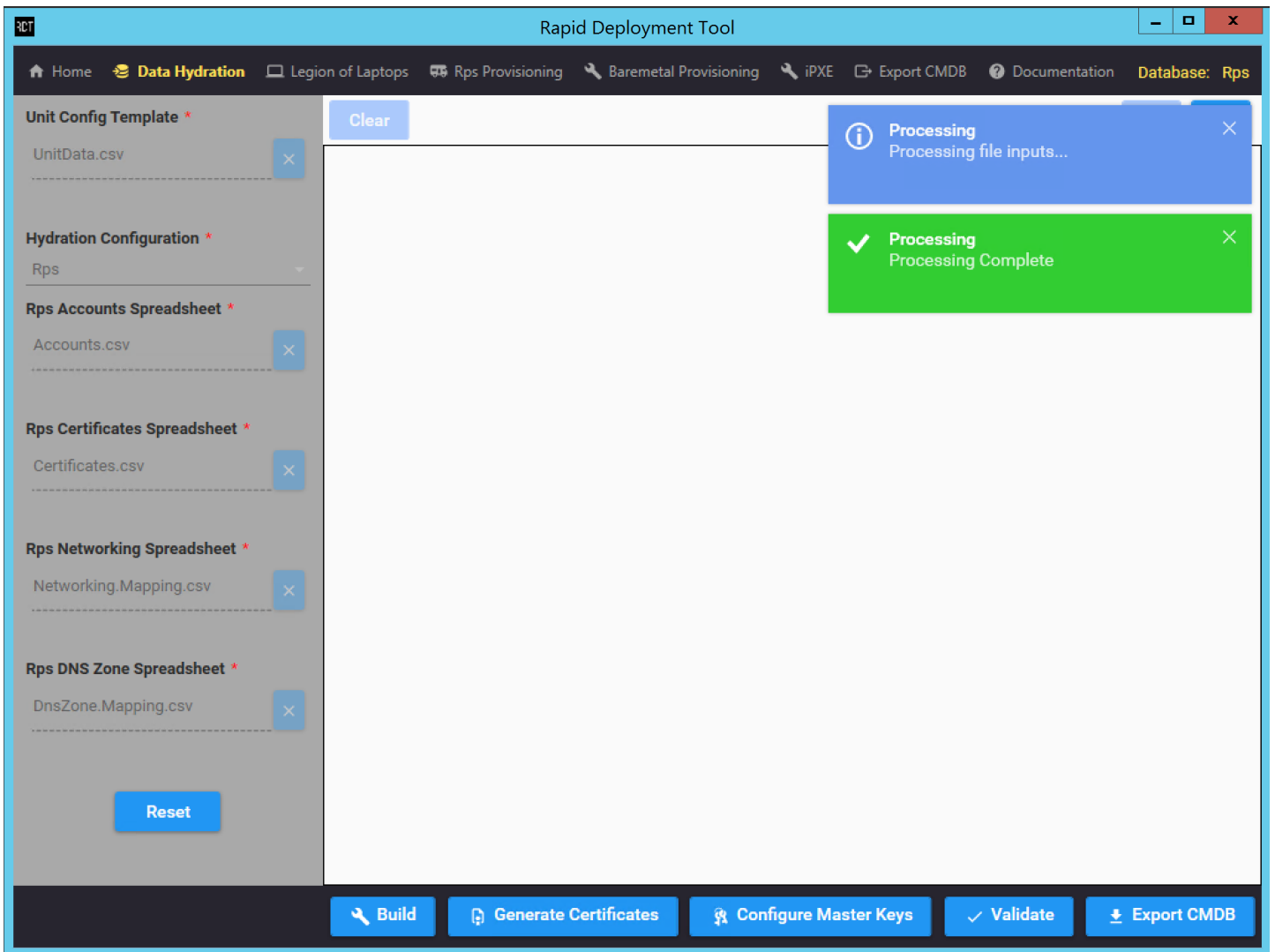


Figure 3: Processed Data Hydration.

Once the input fields have been processed, they can no longer be changed. In order to modify an input field once the Process button has been selected, the Process button becomes the Reset button. Pressing "Reset" clears all inputs and return the Data Hydration screen to the original default view.

Upon processing the data, the Instance Definition table will be hidden and the console will be shown. Toast status messages appears in the top right showing the status of the processing and whether it succeeded or failed.

Data Hydration Order of Operations

The order of operations for a Data Hydration process is [Build](#) → [Generate Certificates](#) → [Configure Master Keys](#) → [Validate](#).

1. Build

When the Build button is selected, the Instance Definition and the data mappings listed in the input sheets are imported into the CMDB. The progress is tracked through log statements, which appear in the console. An example of a successful build is shown in Figure 4:

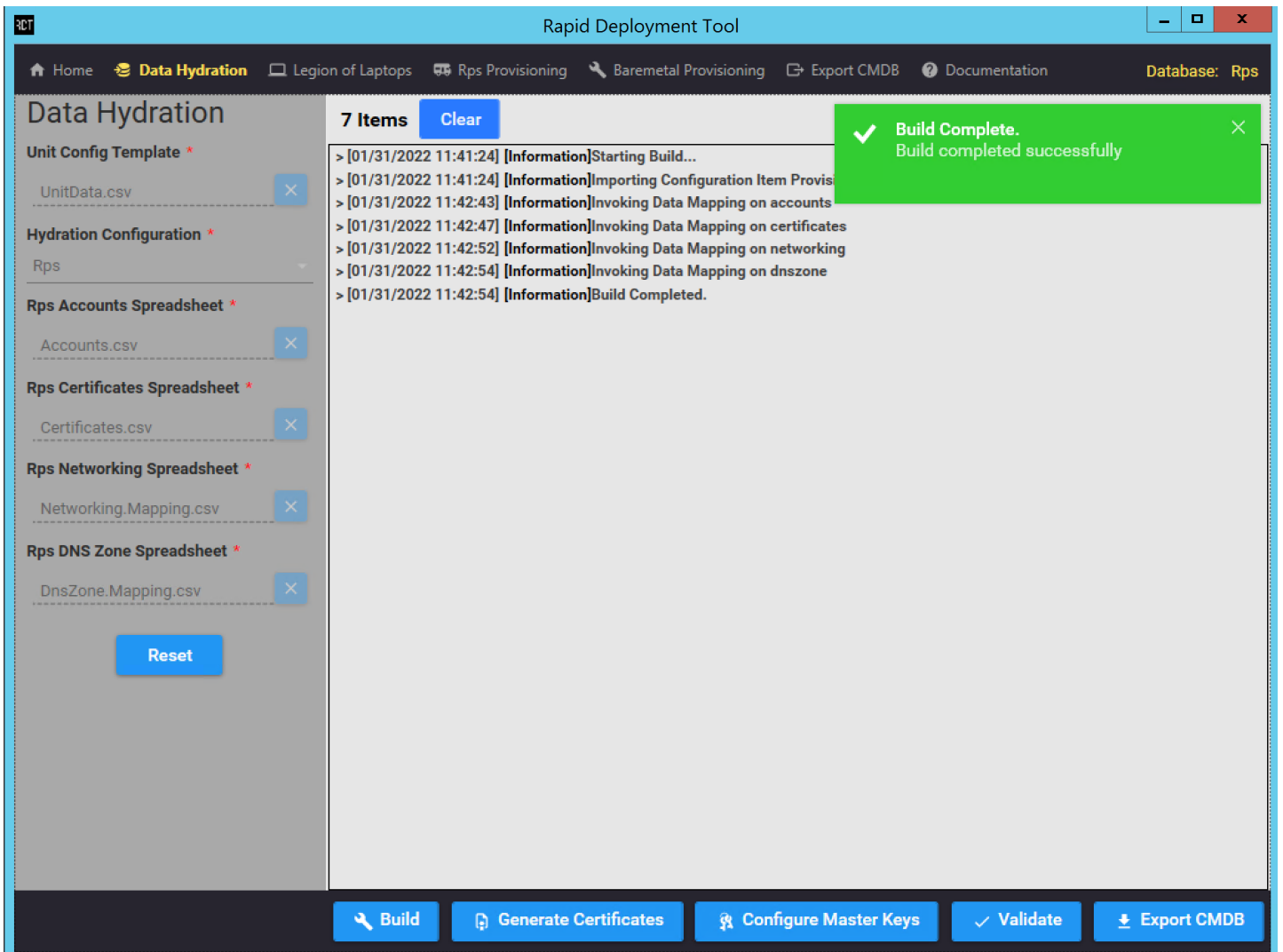


Figure 4: Successful build.

If the build fails, a toast message appears stating that the build failed, in addition to a console log message. See Figure 5 for an example of this:

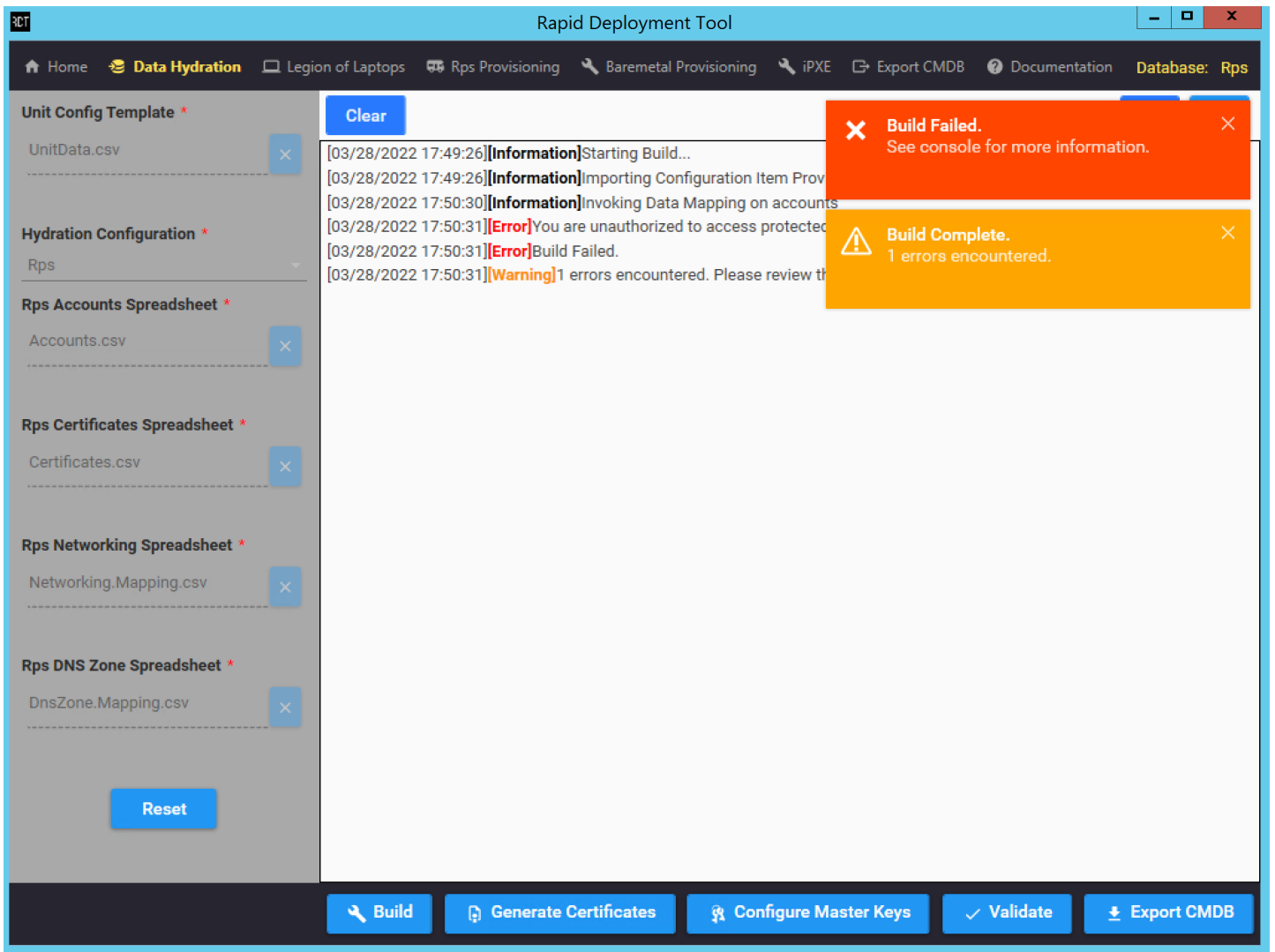


Figure 5: Build failure.

NOTE

Error messages in the console are shown in an orange font.

2. Generate Certificates

Once the build has succeeded, self-signed certificates need to be generated. This can be done by clicking the "Generate Certificates" button. Upon pressing this button, a PowerShell window appears that shows the output of this operation. The user needs to press the Enter key to close the PowerShell window. Any errors in the script would be displayed in the PowerShell window and is not visible in the console log window on RDT.

3. Configure Master Keys

Once the certificates are generated, Master Keys needs to be configured. This can be done by clicking the "Configure Master Keys" button. This opens a PowerShell window which displays the output of the operation. Like Generate Certificates, this PowerShell window does not exit until the user presses the Enter key, and all relevant messages and errors would be displayed in the PowerShell window, not on the RDT console log.

4. Validate

Once the previous steps have been completed, the user can press the Validate button to begin the validation process. The result of this process would be displayed in the console log window.

Export CMDB

Data Hydration Configuration File

The Data Hydration Configuration file is used to configure the inputs that appear on the Data Hydration screen. An example file is shown below:

```
{
  "ImportFields": [
    {
      "Name": "Input 1",
      "DisplayName": "The First Input",
      "FileTypes": [
        ".xlsx",
        ".xls",
        ".csv"
      ],
      "Description": "Some description",
      "WorksheetMappings": [
        {
          "DestSheetName": "Input1",
          "SourceSheetName": "sheet1"
        }
      ],
      "DataMappings": [
        {
          "Table": "Tablename",
          "MappingName": "Tablename.mapping.json",
          "FilterName": "filtername",
          "CIName": "ciname"
        }
      ]
    }
  ]
}
```

The Data Hydration Configuration file contains the following fields:

- **ImportFields:** An array of the input fields present on the Data Hydration screen. Each entry in this array corresponds to a new file input on the screen. This field is required. This is the only allowed root property. Each object in the array must be unique. The properties listed below are the only valid values in each object of this array.
 - **Name:** The name of the input. Type is a string. This field is required.
 - **DisplayName:** The text to display above the input on the Data Hydration page. Type is a string. This field is required.
 - **FileTypes:** The array of supported filetypes. Must have at least a single item. The available options are: ".xlsx", ".xls", and ".csv". This field is required.
 - **Description:** The description for this input. Type is a string. This field is required.
 - **WorksheetMappings:** The array of worksheet mappings for the input file. Must have at least a single item. Each object in the array must be unique. The properties listed below are the only valid values in each object of this array. This field is required.
 - **DestSheetName:** The destination worksheet name. Type is a string. This field is required.
 - **SourceSheetName:** The worksheet name in the input file. Type is a string. This field is optional.
 - **DataMappings:** The array of data mappings for the input file. Must have at least a single item. Each object in the array must be unique. The properties listed below are the only valid values in each object of this array. This field is required.
 - **Table:** The table name. Type is a string. This field is required.
 - **MappingName:** The mapping JSON file. Type is a string. This field is required.
 - **FilterName:** The filter name, if any. Type is a string. This field is required.
 - **CIName:** The CIName, if any. Type is a string. This field is required.

The Data Hydration Configuration file is validated against the following schema:

```
{
  "type": "object",
  "properties": {
    "ImportFields": {
      "type": "array",
      "uniqueItems": true,
      "items": {
        "type": "object",
        "properties": {
          "Name": {
            "type": "string"
          },
          "DisplayName": {
            "type": "string"
          }
        }
      },
      "FileTypes": {
        "type": "array",
        "minItems": 1,
        "uniqueItems": true,
        "items": {
          "type": "string",
          "enum": [
            ".xlsx",
            ".xls",
            ".csv"
          ]
        }
      }
    },
    "Description": {
      "type": "string"
    },
    "WorksheetMappings": {
      "type": "array",
      "minItems": 1,
      "uniqueItems": true,
      "items": {
        "type": "object",
        "properties": {
          "DestSheetName": {
            "type": "string"
          },
          "SourceSheetName": {
            "type": "string"
          }
        }
      },
      "required": [
        "DestSheetName"
      ],
      "additionalProperties": false
    }
  },
  "DataMappings": {
    "type": "array",
    "minItems": 1,
    "uniqueItems": true,
    "items": {
      "type": "object",
      "properties": {
        "Table": {
          "type": "string"
        },
        "MappingName": {
          "type": "string"
        }
      }
    }
  }
}
```

```

    },
    "FilterName": {
      "type": "string"
    },
    "CIName": {
      "type": "string"
    }
  },
  "required": [
    "Table",
    "MappingName",
    "FilterName",
    "CIName"
  ],
  "additionalProperties": false
}
},
"required": [
  "Name",
  "DisplayName",
  "FileTypes",
  "Description",
  "WorksheetMappings",
  "DataMappings"
],
"additionalProperties": false
}
},
"required": [
  "ImportFields"
],
"additionalProperties": false
}

```

The **DataHydrationConfigSchema.json** file is used to validate the **DataHydrationConfig.json** file. Config validation takes place upon navigating to the Data Hydration page. The **DataHydrationConfigSchema.json** file can be found at the following relative path for the RDT executable: `"..\resources\bin\Config\Schemas\DataHydrationConfigSchema.json"`.

Users see a red error box in the top right-hand corner on the RDT window if validation fails, as seen in the following figure:

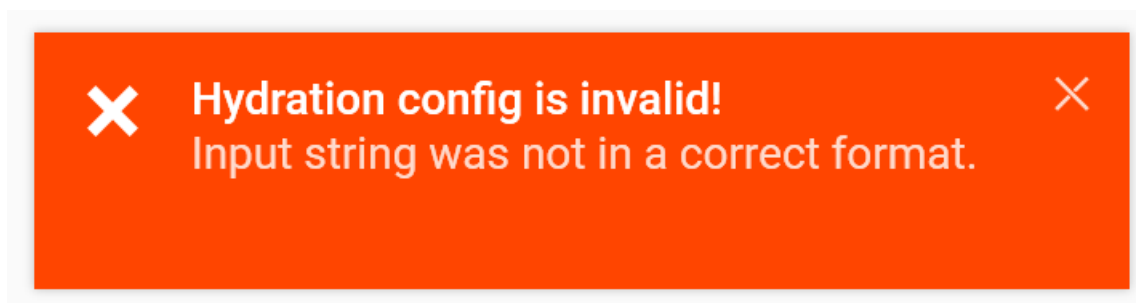


Figure 10: Example showing a failed validation error message.

Legion of Laptops

Last updated on March 28, 2022.

Last Reviewed and Approved on PENDING REVIEW

Introduction

Legion of Laptops (LOL) is a service within the RDT that provides the ability to import an RPS data or security export file. This service is intended to be used during the provisioning process to import existing data from a previous installation of RPS.

Usage

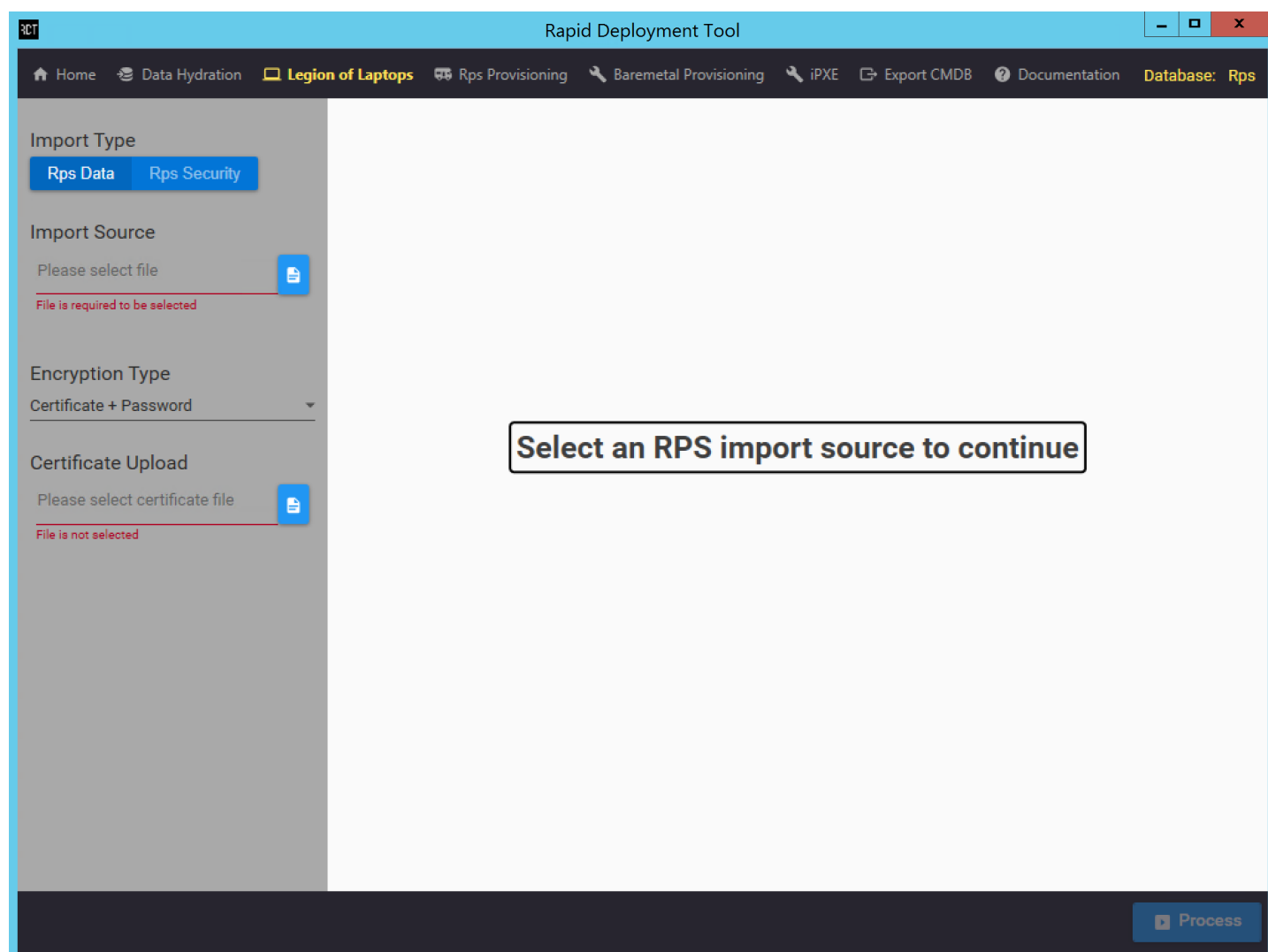


Figure 1: This is the view that the user will be presented with when initially navigating to the Legion of Laptops page.

Inputs

Before Legion of Laptops can be run, the user must select which import type they would like to perform. Additionally, the user must supply a corresponding import source for the selected import type. The following file extensions are supported for each import type:

- **Data Import:** *.xml, .json*
- **Security Import:** *.xml*

The import source must be a valid RPS-generated export file obtained from a previous installation of RPS. Export files can be generated either through the RPS Web GUI or through PowerShell. Please refer to [RBAC: How to Import and Export Users with](#)

[the Web User Interface](#) for detailed information on how to generate an RPS security export for an existing installation.

Because RPS supports encrypted exports, Legion of Laptops is capable of importing encrypted files. The user must select the appropriate encryption type for the import source they have provided. The following encryption types are supported:

- Certificate + Password
- Thumbprint

Starting the Import Process

The *Process* button will become enabled once an import source is provided. Click *Process* to begin validation on the import source. The user will then be notified if there were any issues with parsing or decrypting the import source. Please be sure that the provided import source matches the selected import type.

Once the import source passes validation and the file content is successfully read, the user will be presented with a *LogViewer* window. This window contains relevant log messages for the import process. It displays number of log messages during the process.

The *Clear* button clears all the messages in the log viewer.

The *Live* button, seen in the upper right-hand corner of the screen, allows to toggle "live scrolling" of the log messages on or off.

Possible Outcomes

The user will be notified about whether the import succeeded upon completion of the process (as seen in Figure 2, below).

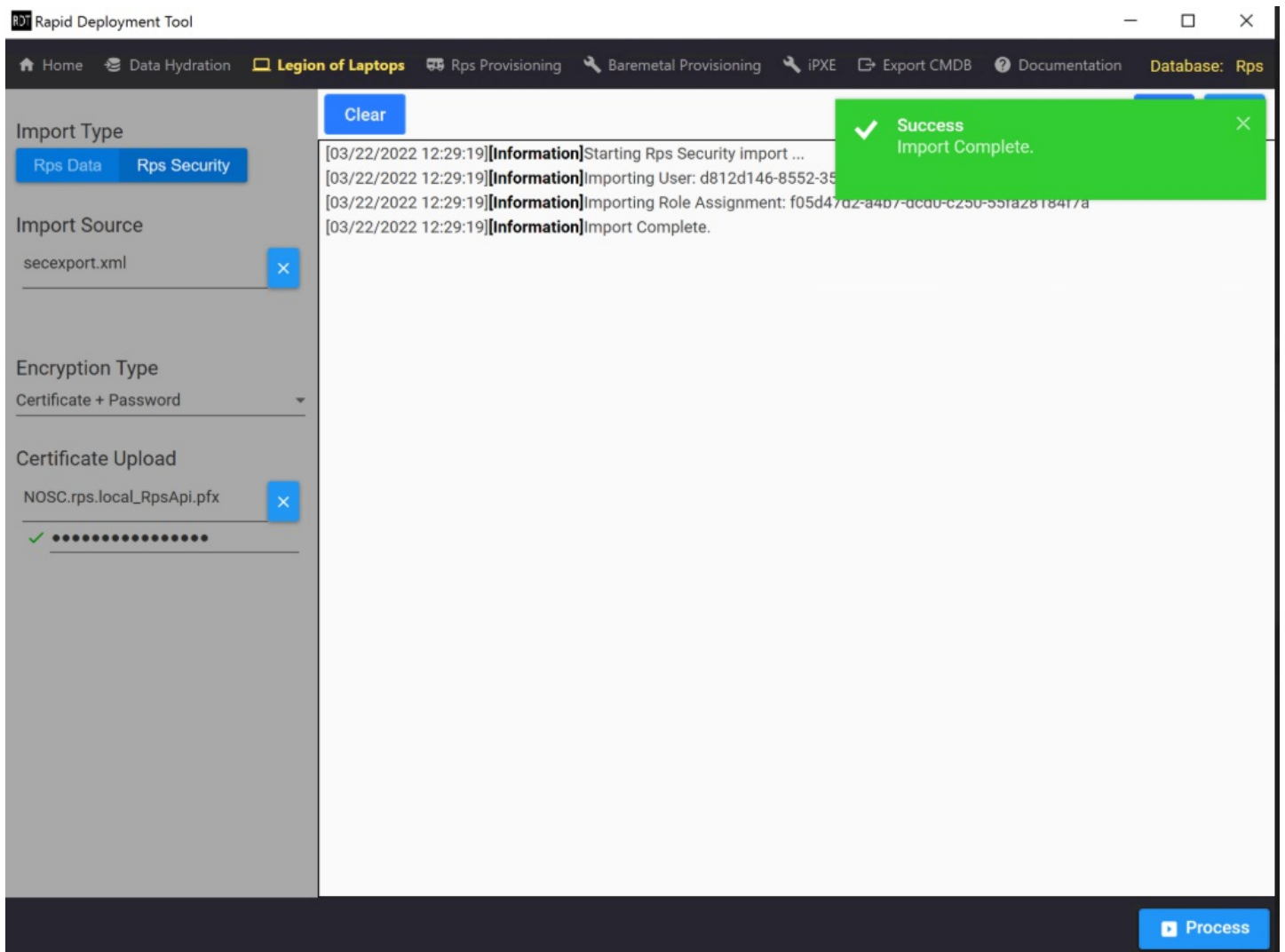


Figure 2 : This is an example of an RPS Data Import

Currently, RPS only supports imports through the legacy RPS Importer. The legacy importer does not handle conflicts if data being imported already exists in the database (based off of the GUID). If data being imported already exists in the database, the legacy importer will show errors. This does **not** indicate that the import process failed. The following figure offers an example of this scenario:

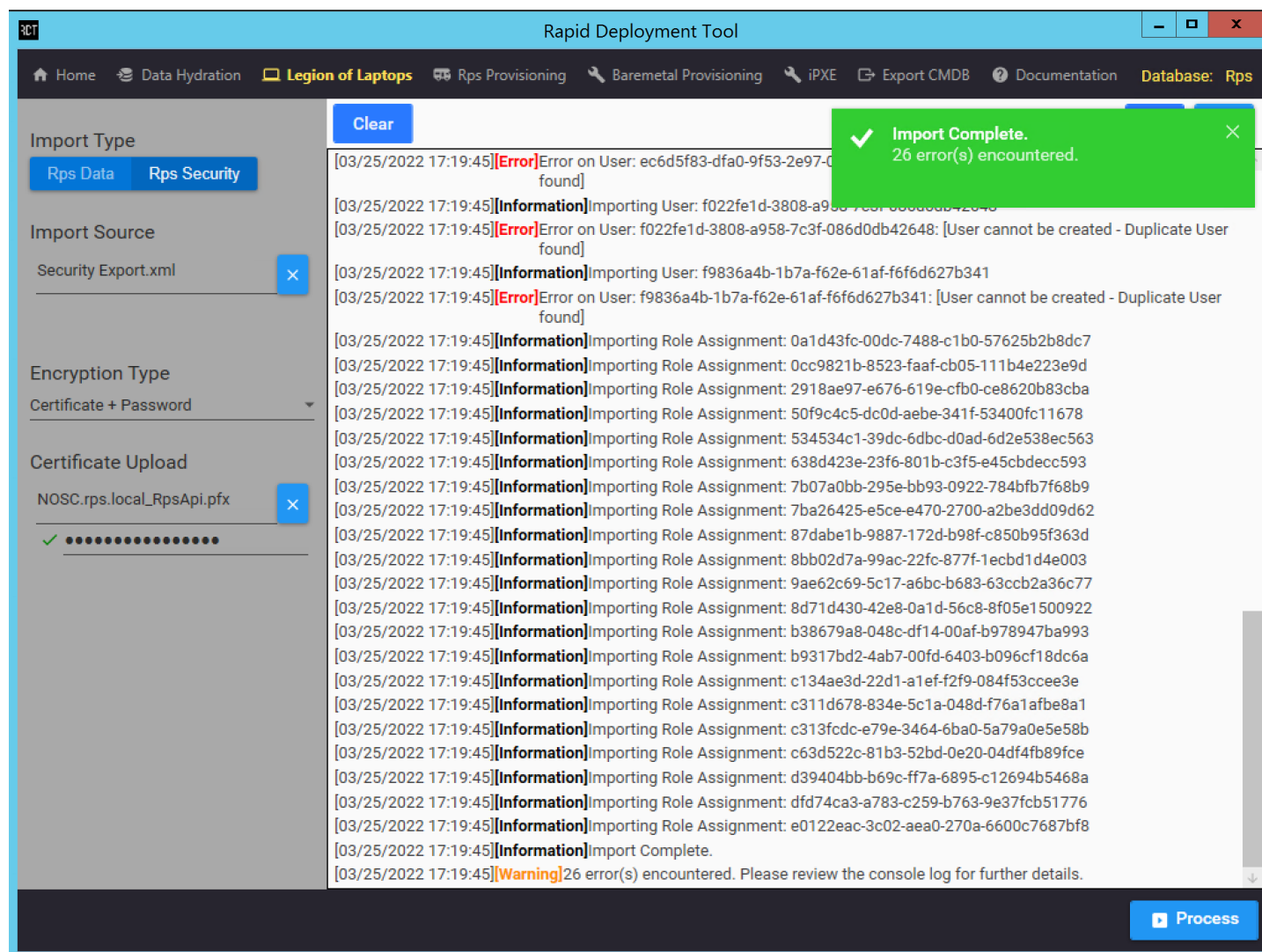


Figure 3 : This is an example of an RPS Data Import which completed with errors encountered.

There are three possible outcomes of performing an import through LOL:

- Import Complete. (Green toast notification & no errors).
- Import Complete. (Green toast notification & error(s) encountered).
- Import Failed. (Red toast notification).

A green notification indicates that the import was successful. A yellow notification indicates that some data could not be imported. A green notifications with message *Import Complete*. indicate that the import process at least completed. Import errors can occur for multiple reasons. Review the errors carefully to determine if further action is needed.

A red notification with message *Import Failed*. indicates that the import process encountered an exception and did not complete.

RPS Provisioning

Last updated on March 28, 2022.

Last Reviewed and Approved on PENDING REVIEW

Introduction

RPS Provisioning is a service within the RDT that provides the ability to provision and configure Nodes through use of the Task Management Service (TMS).

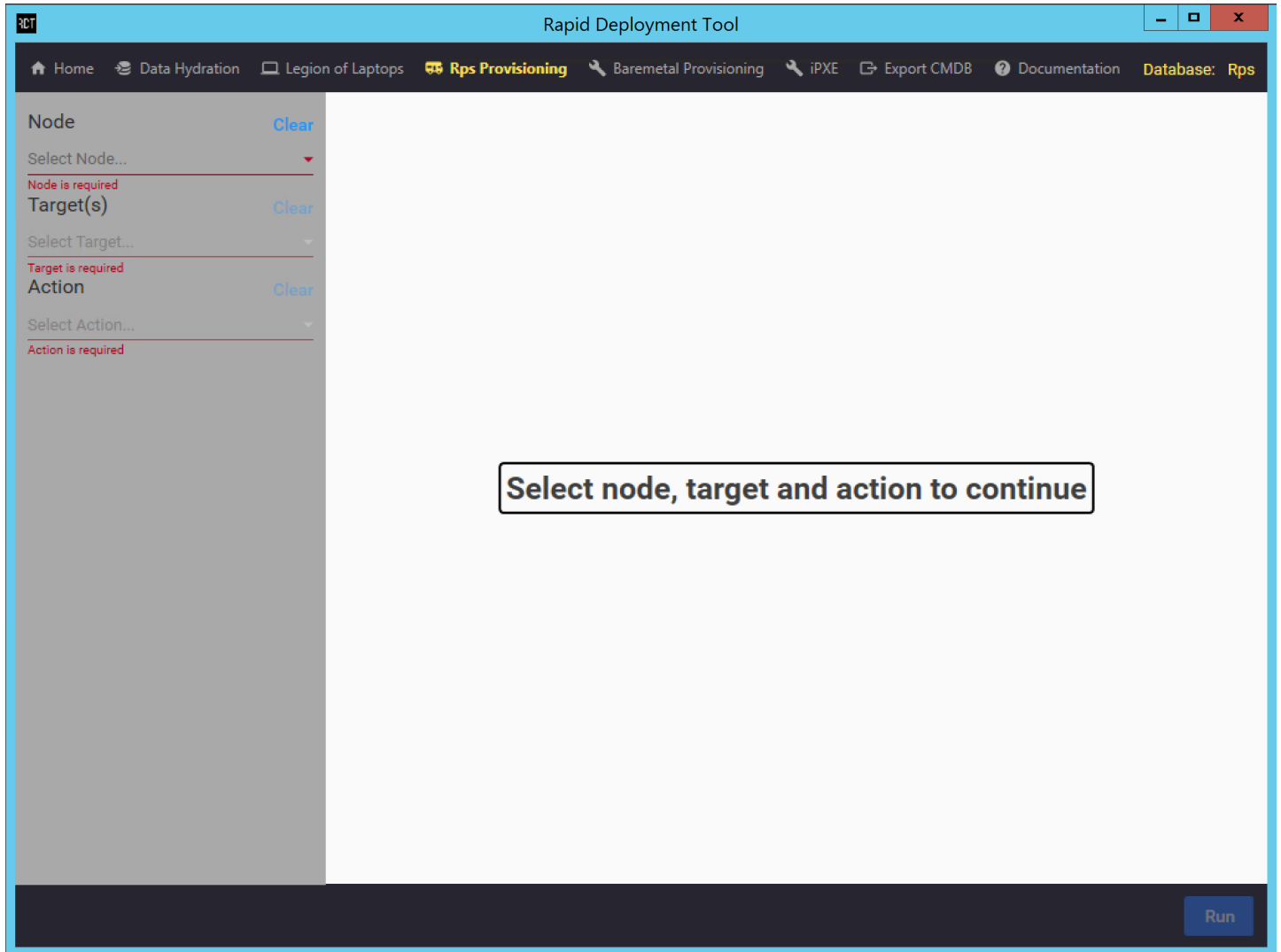


Figure 1: This is the view the user sees when navigating to the RPS Provisioning page.

The provisioning process consists of running through a sequence of *Steps* which are defined when the user selects an *Action* from the dropdown. Each provisioning *Step* corresponds to a *TaskMap* assigned to the selected top level target and its children. When the provisioning process is started, the *TaskItems* for each generated *TaskAssignment* are executed by TMS.

The functionality of RPS Provisioning can also be achieved through existing PowerShell cmdlets.

Please refer to the [RPS Tasking Guide](#) for more information on the aforementioned PowerShell cmdlets and tasking.

Configuration

The options available in the *Action* dropdown menu in RPS Provisioning are defined via a configuration file named **RDTActions.json**. This file can be found at the following relative path for the RDT executable:

"..\resources\bin\Config\RDTActions.json".

The following snippet is an example of an **RDTActions.json** configuration for RPS Provisioning which contains a single *Action* called "Install-Rps"; however, a configuration file may contain multiple *Actions*.

```
[
  {
    "Name": "Install-Rps",
    "Steps": [
      {
        "DisplayName": "Install",
        "TaskMapName": "Install-Rps"
      }
    ]
  }
]
```

An *Action* may contain multiple *Steps*; however, in the example above, only one *Step* is defined. The "DisplayName" property is used to reference the *Step* in the UI. The "TaskMapName" property is used to associate the *Step* with a particular *TaskMap*.

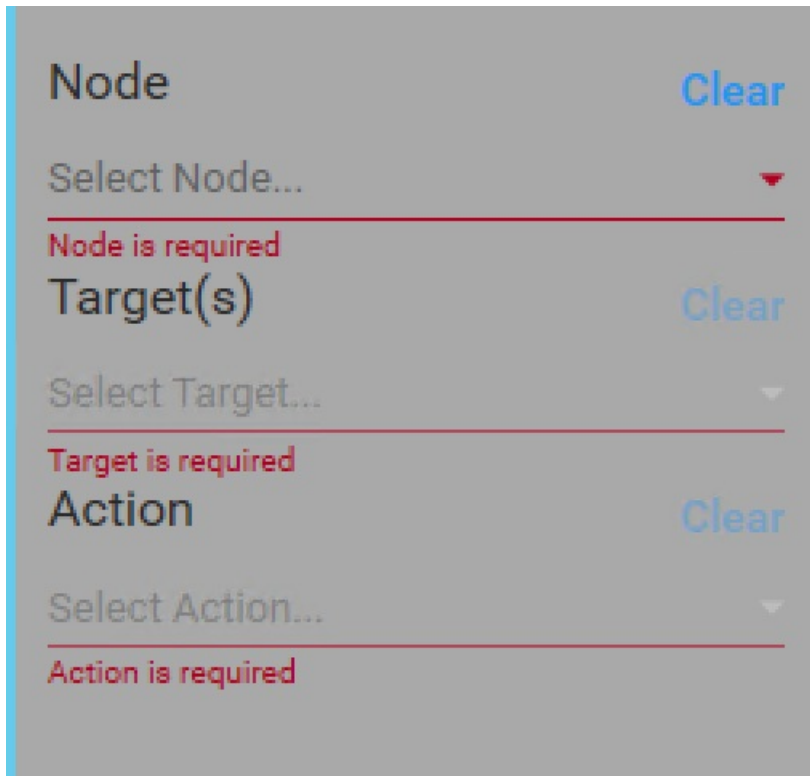


Figure 2: The **RDTActions.json** file is used to populate the dropdown for the *Action* input seen in this example. Users may select which *Action* they would like to execute.

Users may add or remove an *Action* from the configuration by editing the **RDTActions.json** file.

Validation

The **RDTActionsSchema.json** file is used to validate the **RDTActions.json** file and initiates upon navigating to the RPS Provisioning page. The **RDTActionsSchema.json** file can be found at the following relative path for the RDT executable : **"..\resources\bin\Config\Schemas\RDTActionsSchema.json"**.

If validation fails, users sees a red error box in the top right-hand corner of the RDT window, as seen in the following figure:

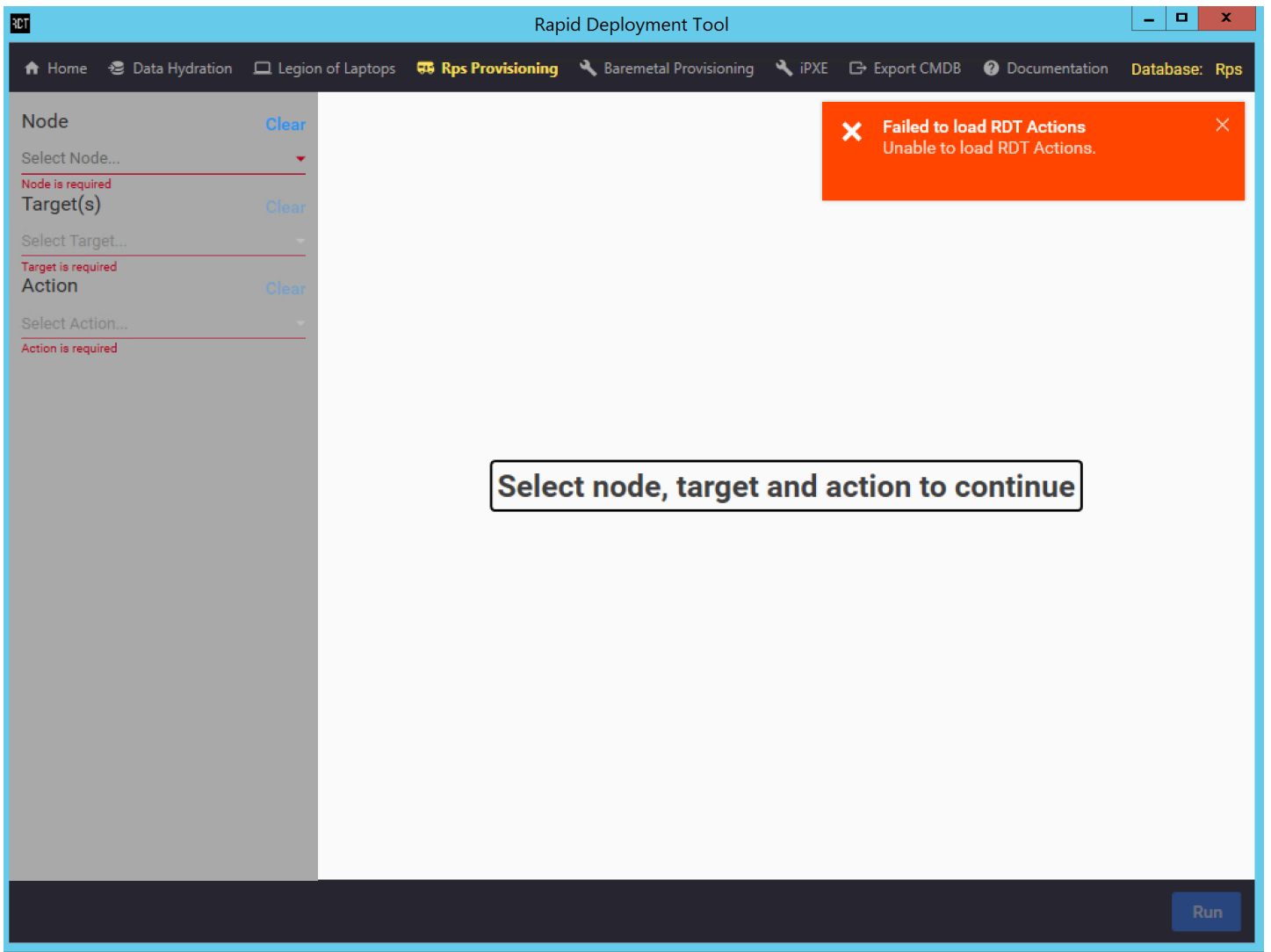


Figure 3: Example showing failed validation error message.

The following snippet is an example of an **RDTActionsSchema.json** file used for validation:

```

{
  "type": "array",
  "uniqueItems": true,
  "minItems": 1,
  "items": {
    "type": "object",
    "properties": {
      "Name": {
        "type": "string"
      },
      "Steps": {
        "type": "array",
        "minItems": 1,
        "uniqueItems": true,
        "items": {
          "type": "object",
          "properties": {
            "DisplayName": {
              "type": "string"
            },
            "TaskMapName": {
              "type": "string"
            }
          }
        },
        "required": [
          "DisplayName",
          "TaskMapName"
        ],
        "additionalProperties": false
      }
    }
  },
  "required": [
    "Name",
    "Steps"
  ],
  "additionalProperties": false
}

```

Based on the above schema, the **RDTActions.json** file may contain the following fields:

- An array of *Actions* to be presented within the *Action* dropdown on the RPS Provisioning screen. Each entry in this array corresponds to a single *Action* in the dropdown. At a minimum, one *Action* must be provided. Additionally, each *Action* must be unique.
 - **Name:** The name of the *Action*. Type is a string. This field is required.
 - **Steps:** The array of *Steps* for the *Action*. Must have at least a single *Step*. Each *Step* in the array must be unique. The properties listed below are the only valid values in each object of this array. This field is required.
 - **DisplayName:** The name to display in the UI for the *Step*. Type is a string. This field is required.
 - **TaskMapName:** The name of the *TaskMap* for this *Step*. Type is a string. This field is required.

Additional properties are not allowed anywhere in the **RDTActions.json** file based on the schema.

Usage

To use the Provisioning service, users must select an item from the *Node*, *Target(s)*, and *Action* dropdown fields. The fields must be populated in order, beginning with *Node* and ending with *Action*. The *Node* field dictates which targets are available to provision.

When the *Action* field is populated, the user will be shown a preview of *TaskAssignments* that will execute when the RDT Tool is run. After all three fields are populated, the **Run** button will be made available in the bottom right-hand corner.

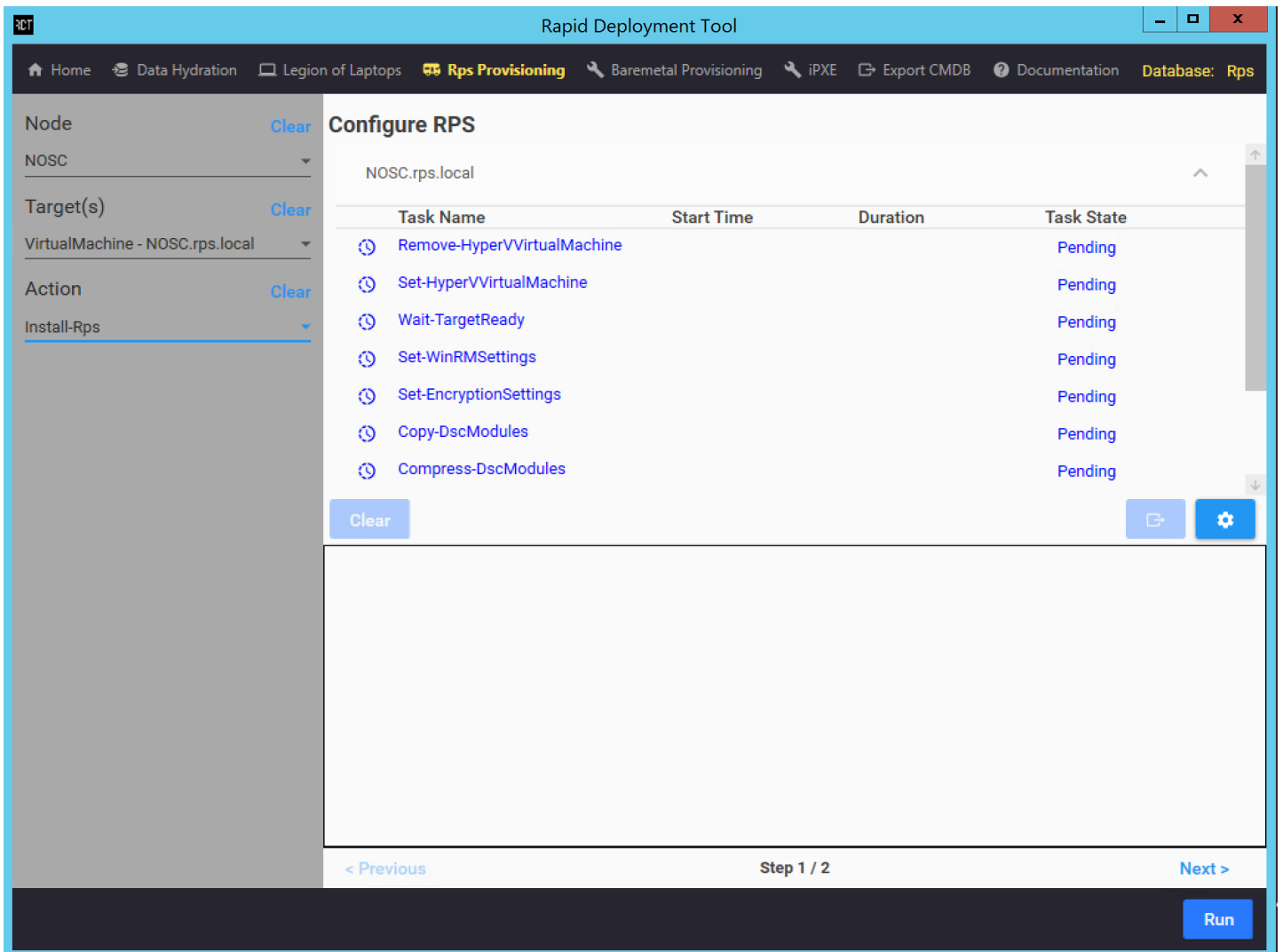









Figure 4: Example of the preview that users will see after populating the *Action* field.

The *Steps* for the selected *Action* are shown at the top of the page above the header *Tasks*. In this example there is only one *Step*, "Install", as indicated by the blue bubble. Executions requiring multiple *Steps* will be listed sequentially from left to right.

The items listed as *Tasks* are also ordered sequentially and are based on the *TaskMap* associated with the current *Step*. Each *Task* can be clicked on to view which *Targets* have an assignment. Please visit [RPS Task Assignment Diagram](#) for more information on how *TaskAssignments* are made.

When the **Run** button is clicked, the assignments shown in the preview window will be executed by TMS. The icon to the right of each *Task* indicates its current status and will automatically update progress as each *Task* completes. The following list defines the corresponding *Task* statuses for each icon:

-  **Not Ready, Ready, Pending, Assigned**
-  **Running**
-  **Retry**
-  **ErrorContinue, PendingUserAction, Warning, Removed**
-  **Canceled, ErrorStop**
-  **Completed**
-  **Questions**

NOTE

Only the icon will be displayed in RDT; however, the specific statuses listed above can be seen if the *TaskAssignments* are queried via PowerShell.

The status for each *Task* is refreshed every 3 seconds. Additionally, if a user briefly navigates away from this page and then returns, the existing *Tasks* will be loaded from the CMDB if the same inputs are selected. In this situation, the **Run** button would remain disabled since the assignments have already been created and executed by TMS. The user would also be notified via a toast (pop-up) notification that the current *Step* was loaded from the CMDB.

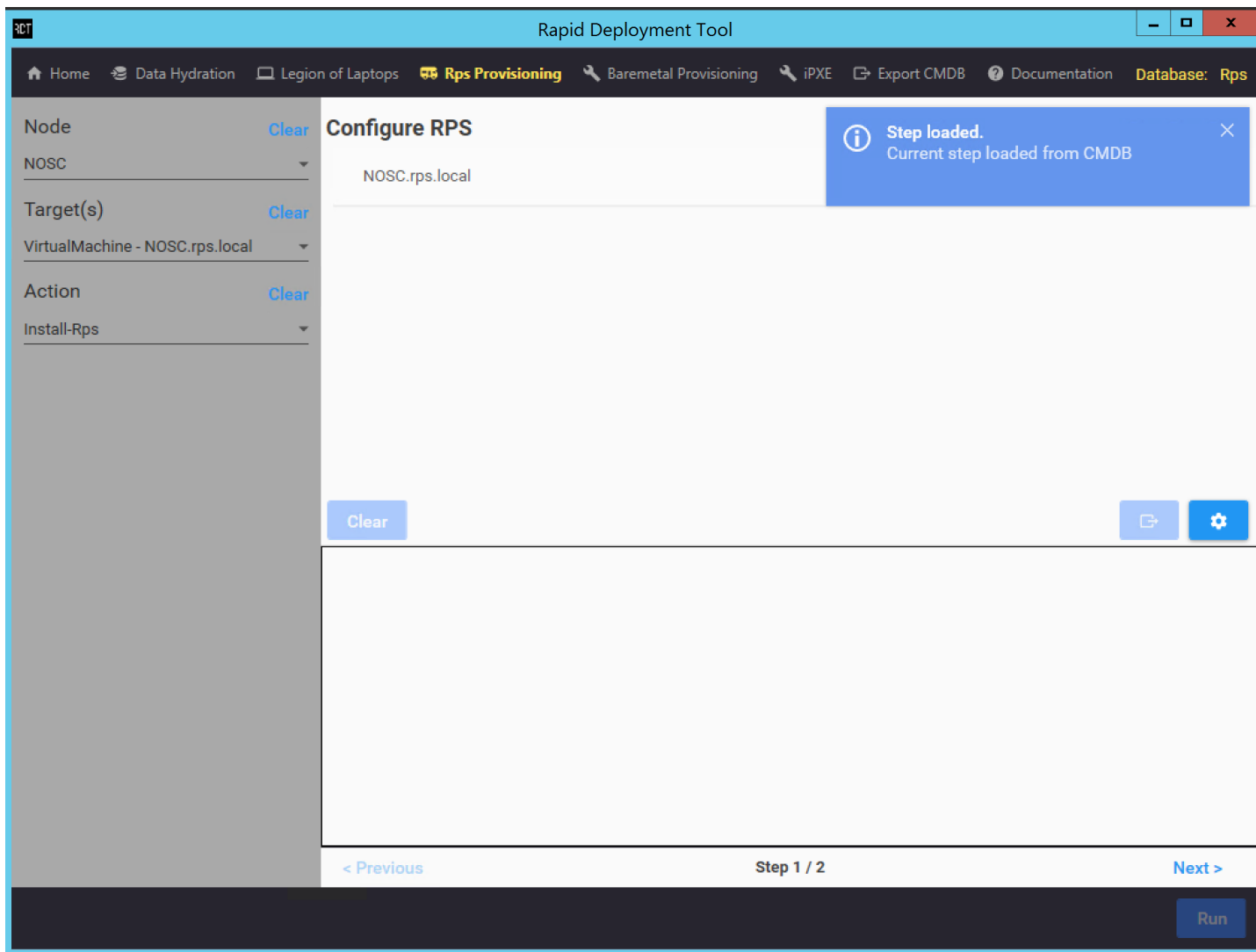


Figure 5: Example of a toast (pop-up) notification for an existing assignment.

Users may also track RPS Provisioning status and learn more detailed information through the logs displayed in the *Console Logs* box. Users may also navigate between *Steps*, to view their corresponding *Tasks* and statuses, by clicking the **Previous** and **Next** buttons below the console.

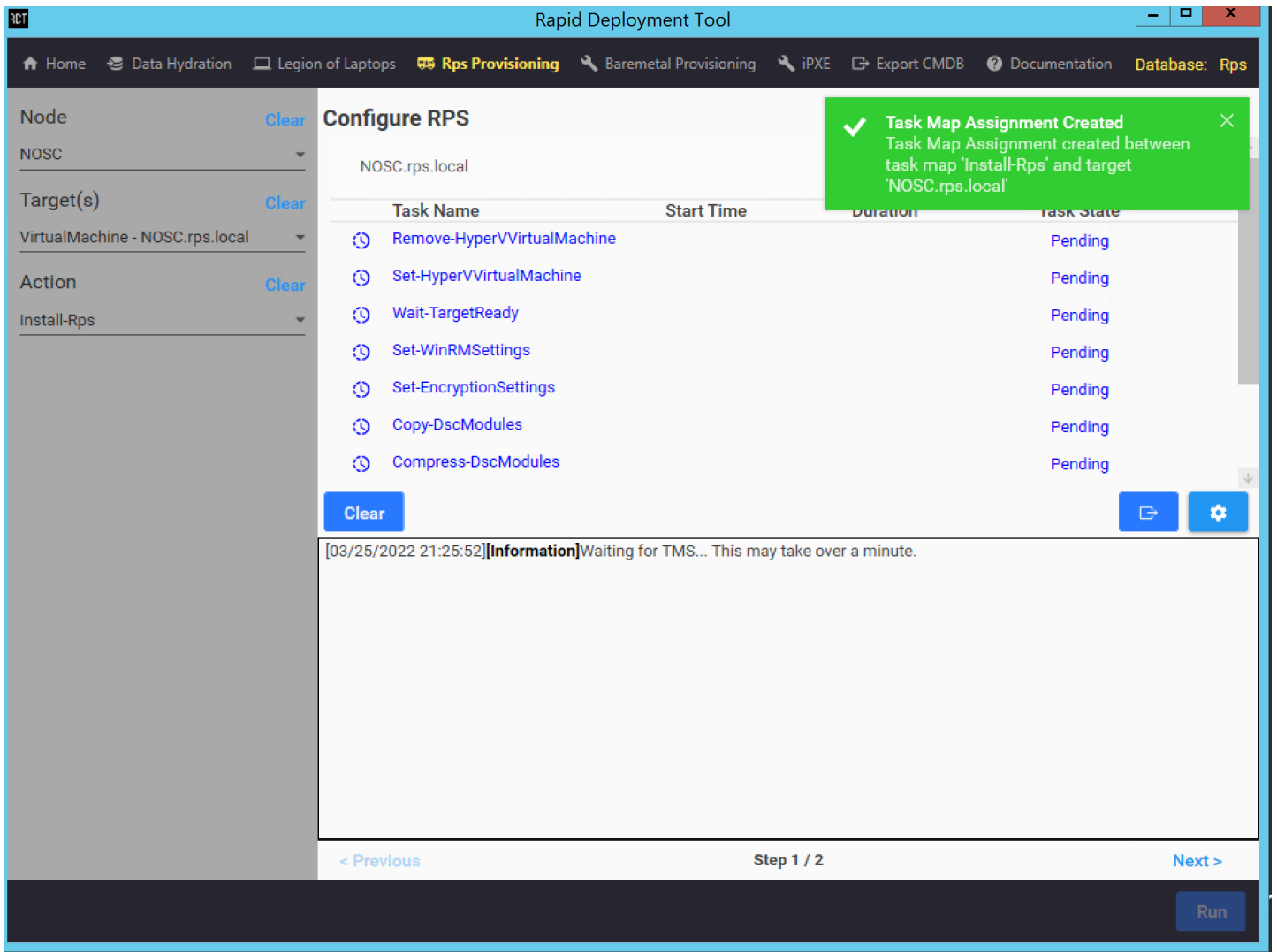


Figure 6: Example of a provisioning process that has been started, and is waiting on TMS.

The **Live** button, seen just above the *Console Logs* box, allows users to toggle "live scrolling" of the console messages on or off.

Possible Outcomes

RPS Provisioning will complete once TMS has executed all of the *Tasks* within each *Step*. The following statuses indicate that the *Task* has been executed:

- Completed
- ErrorContinue
- Canceled, ErrorStop

Any other status than listed above indicates that the *Task* has not yet been run. Once all *Tasks* have been executed, the **Run** button will become enabled again, which signifies that the provisioning process has completed.

To confirm successful completion, expand and examine each *Task* status, and review the console logs.

iPXE

Last updated on March 28, 2022.

Last Reviewed and Approved on PENDING REVIEW

Overview

iPXE is an open source network boot firmware that can boot in various ways, such as from a web server via HTTP.

The iPXE feature of RDT provides a way to generate an iPXE image that can be used to make a bootable disk.

Inputs

- iPXE script - A script must be selected to override the default behavior of iPXE. This script will be embedded within the iPXE (binary) firmware that is generated.
- iPXE source code location - The iPXE source code location must be selected so that RDT knows where the code is to build the firmware. By default this location will be a relative path based on the local node's content store, e.g. `C:\ContentStore\Utilities\ipxe\src\`.
- Certificate - A certificate must be selected to enable HTTPS communication. This certificate will be used when the iPXE firmware is generated as the certificate and trusted root.

Outputs

- Bootable device - A device must be selected for the iPXE firmware to be applied to. This will make the device bootable via the iPXE firmware.

Building a Bootable iPXE disk

- Fill out the required fields:
 - iPXE script
 - iPXE source code location
 - Certificate
 - Bootable device
- Click on the Build button to generate the iPXE firmware and to create the bootable device.

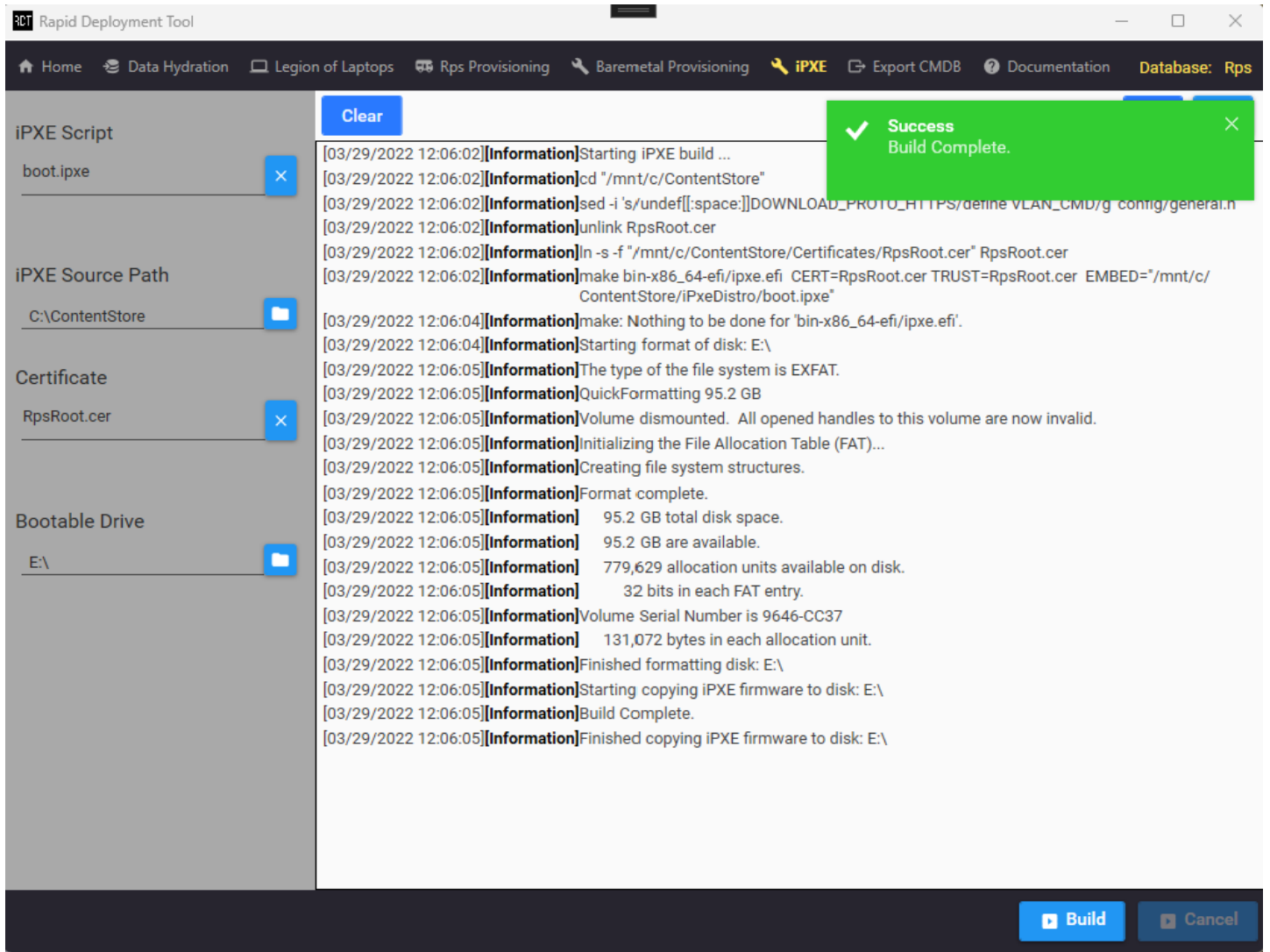


Figure 4: Successful build.

Export CMDB

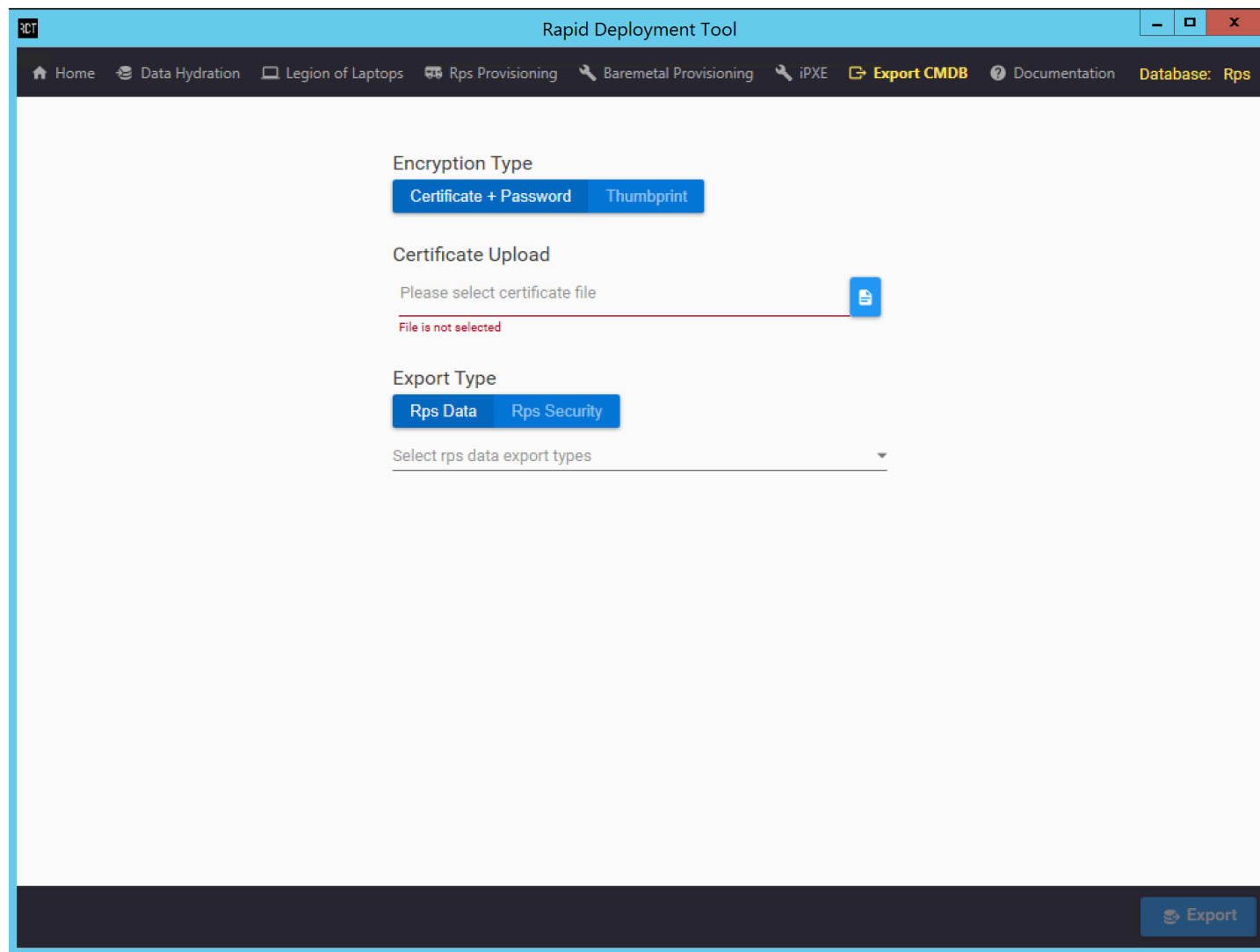
Last updated on March 28, 2022.

Last Reviewed and Approved on PENDING REVIEW

Introduction

Export CMDB is a service within the RDT that provides the ability to export the newly hydrated CMDB.

Usage



The screenshot displays the 'Export CMDB' interface within the 'Rapid Deployment Tool' (RDT). The top navigation bar includes links for Home, Data Hydration, Legion of Laptops, Rps Provisioning, Baremetal Provisioning, iPXE, Export CMDB (highlighted), and Documentation, along with the current database 'Rps'. The main content area features three sections: 'Encryption Type' with buttons for 'Certificate + Password' and 'Thumbprint'; 'Certificate Upload' with a file selection prompt, a red error message 'File is not selected', and a file upload icon; and 'Export Type' with buttons for 'Rps Data' and 'Rps Security', followed by a dropdown menu labeled 'Select rps data export types'. An 'Export' button is positioned in the bottom right corner.

Figure 1: Export CMDB page.

Inputs

The following are the two encryption types:

- Certificate + Password
- Thumbprint

If encryption type is **Certificate + Password**, user must enter password associated with the selected certificate.

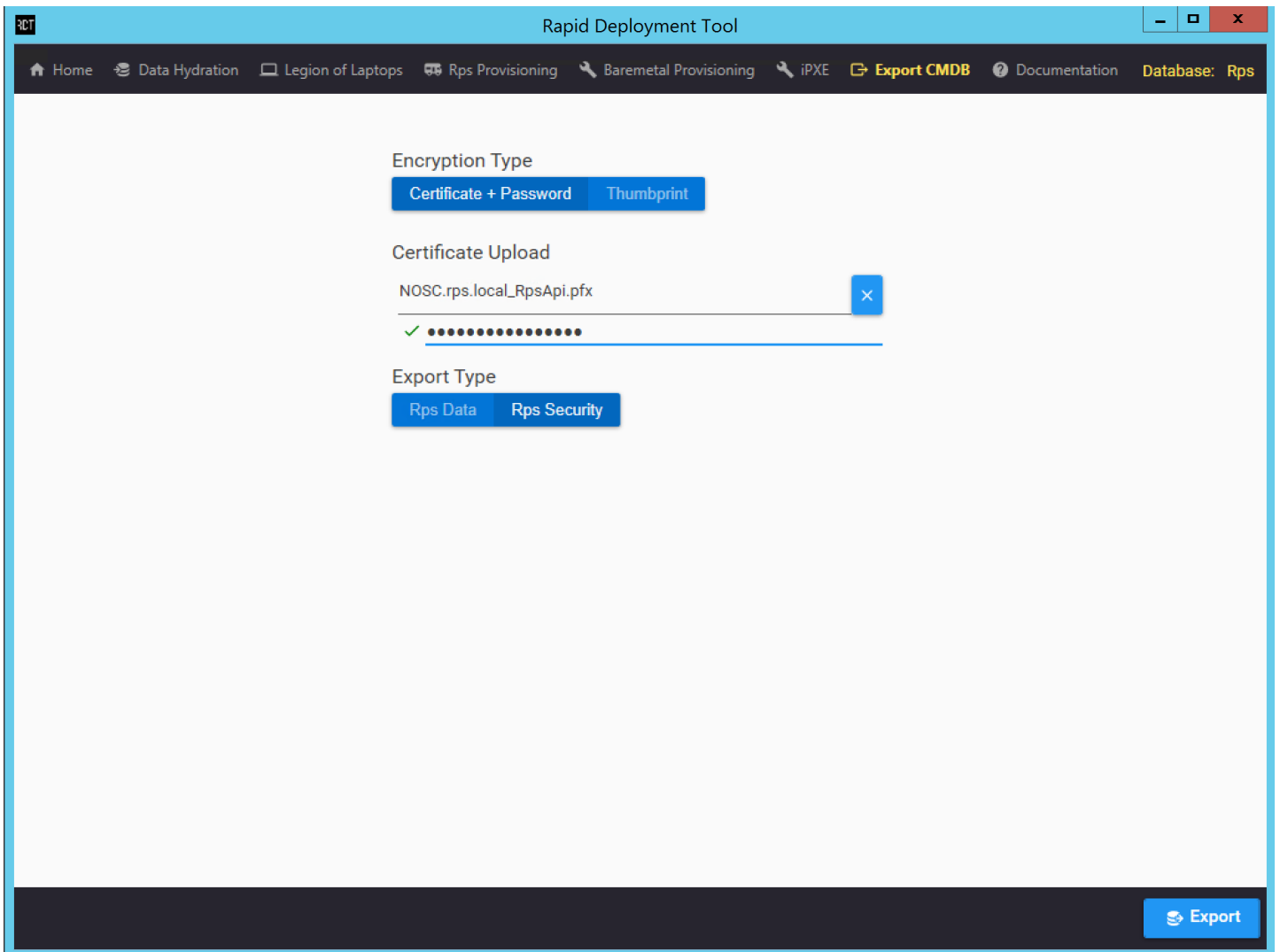


Figure 2: Certificate with Password as Encryption Type.

If the encryption type is **Thumbprint**, user must select valid thumbprint value from the dropdown list.

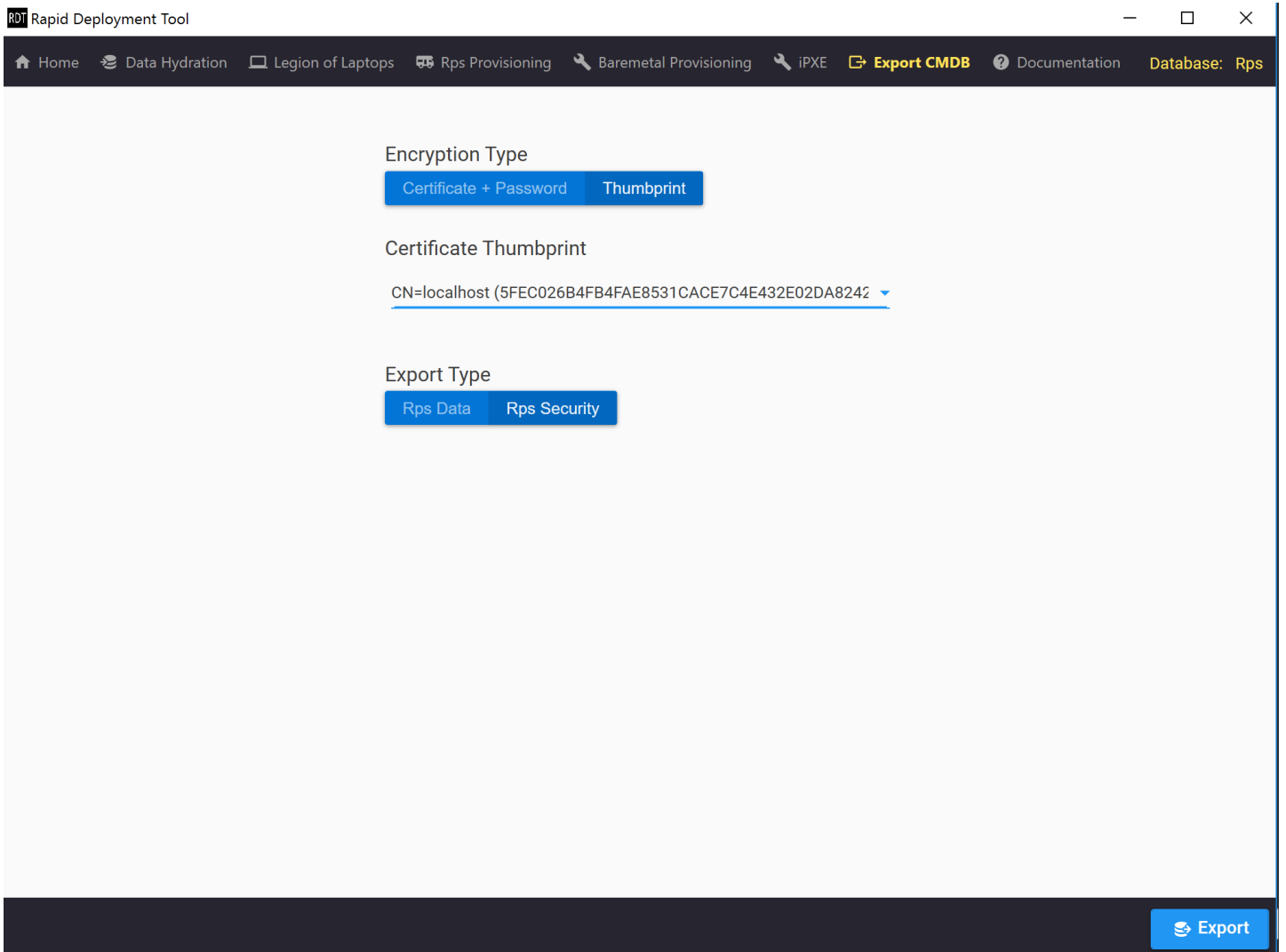


Figure 3: Thumbprint as Encryption Type.

The following are the two export types:

- Rps Data
- Rps Security

If the export type is **Rps Data**, user must select one of the appropriate available option from the dropdown list.

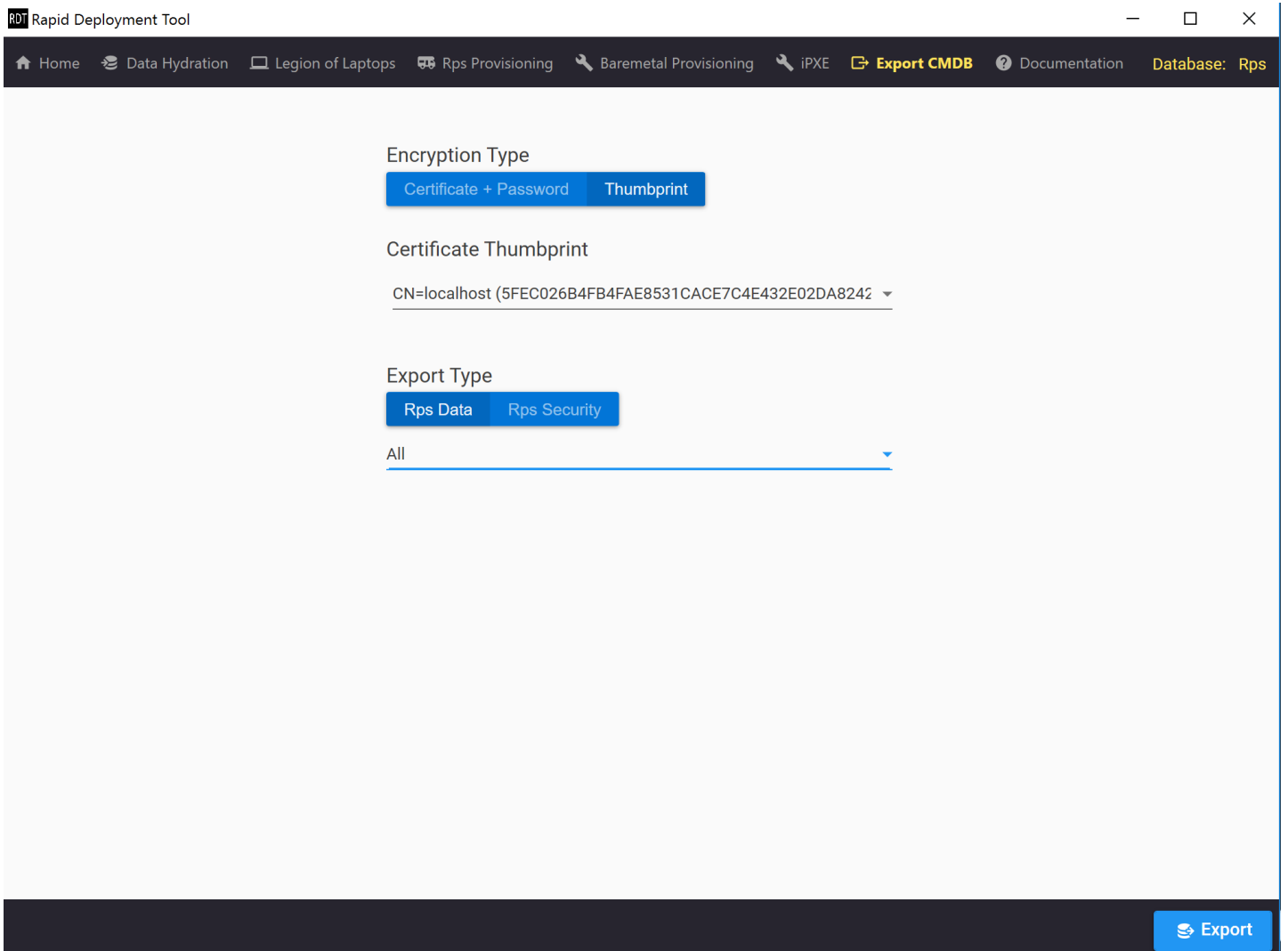


Figure 4: Rps Data as Export Type.

Rps Security does not have any options.

Starting the Export Process

The *Export* button becomes enabled when valid encryption type and export type is provided. Click *Export* to begin hydrated cmdb. This exported file can be used in Legion of Laptops and import source file.

Regardless which method is used, upon completion of the export, status messages will be shown in the upper right-hand corner alerting the outcome of the export, as seen in Figure 9.

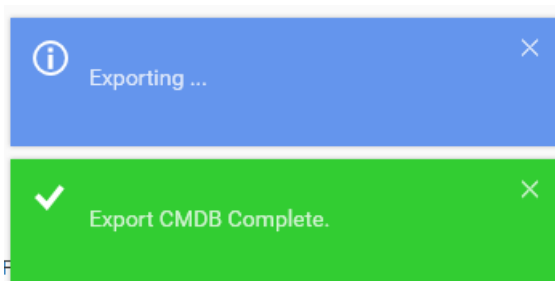


Figure 5: Export complete status messages.