# Overview

Consider the following guidelines and expected behaviors, which apply generally to all types of Adaptable Framework drivers (CA, Application, Log, Workflow, etc.).

- Drivers rely on `System.Management.Automation` in **.NET** to invoke PowerShell functions inside a locally hosted script
- PowerShell functions are responsible for ensuring that data meets the requirements of the integration point
- All functions (except where noted) must be present in the script, but they are optional from a logic standpoint.

## Hash Tables

Venafi passes variables to (and receives them from) PowerShell functions in the form of hash tables, which are generally easier to work with and allow for the addition of variables without changing the function definition.

- A general hash table, `$General`, which includes a common set of data available from Venafi, is passed to all the functions
- A specific hash table, `$Specific`, which includes data that is applicable only to the specific function, is passed to some functions that require additional data
- All functions must return a single hash table that includes a result along with any other variables that the function is required to return - check the documentation for the function in question to see what is expected in the return

## Naming Conventions

To avoid collisions, you should adopt a standard naming convention that will be used when creating objects (certificates, devices, applications) in Venafi. Collisions can occur when the name you define already exists in Venafi's inventory. One option to avoid collisions is to use the same prefix or suffix, and then add something unique to the name.

Suppose your driver connects to a **device object** via API and imports all the machine identities currently in use on the device. In addition to the certificates themselves being imported, a unique **application object** must be created as well, which will contain all the metadata (port, partition, profile, configuration options, etc.) needed for Venafi to automatically provision that certificate during a renewal.

!!! note

```
When creating new PowerShell scripts for use with adaptable drivers, keep in
mind that the file name is used to identify your script from within the
associated object (and within the policy for the Adaptable Application
driver) in Trust Protection Platform.
```

```
Using logical names can help you and other administrators recognize the
purpose and intent of each script.
```

## Credentials & Authentication

**Credential objects** store the credentials Trust Protection Platform uses to authenticate with devices, applications, CAs, and other systems in a user's infrastructure. The stored credential may be a password, a username and password, a certificate, a file paired with a password, or a private key. Trust Protection Platform requires these credentials so it can manage the certificates associated with these devices, applications, and CAs.

Credential objects provide an innovative way to centrally manage and share your system credentials. Each credential object can be associated with a single device or application, or it can be shared by multiple objects.

After you create your system’s credential objects, you do not have to repeat the credential configuration for each device or application. You simply reference the existing credential object. If the credential changes—for example, an organization might change username and password credentials every 90 days—you merely update the single credential object to give Trust Protection Platform access to all associated devices and applications.

- Reuse credential objects wherever standardization is possible to avoid creating extraneous objects.
- Recommend assigning credentials to a Policy folder that contains multiple devices. The policy credentials allow multiple devices to use the same credential.

## Certificate "Installations"

Within a customer’s environment, there may be circumstances that require the same certificate to be installed in multiple locations (high-availability, load balancing, traffic inspection, etc.). If that is the case, the Venafi platform must be made aware of each location. This ensures there are no blind spots for the Security or PKI team, and also provides Venafi the information necessary to automatically provision and activate that certificate during future renewal operations.

!!! danger "Possible Outage!"

```
If your application supports multiple installations of the same certificate,
it is **critical** that **ALL** those locations are reported back to Venafi.
```

## Data Validation

You should implement effective data validation in order to catch errors in the script:

- Check values for validity that are being passed into functions
- Check values for validity before passing data back to Venafi

## Logging

Generally, Adaptable Framework scripts don't need to do any logging unless it's for temporary, debugging purposes. The Adaptable drivers are logging to the standard Venafi logging channel when they call various PowerShell functions.

Beginning in v19.3, developers can toggle debug logging for individual Adaptable drivers. This sets a `$DEBUG_FILE` global variable in the Adaptable driver. If toggled on, the script should be logging – If toggled off, it shouldn't.

The `$DEBUG_FILE` variable gets set automatically to a suggested file path on the Venafi server for the script to write logs to, using a unique identifier. This is to avoid issues if there are multiple instances trying to write data to the same file.

The resulting log file appears in the `<Venafi Home>\Logs` directory by default. (e.g. `C:\Program Files\Venafi\Logs`)

!!! note "Permanent Logs"

```
If additional, permanent logs are needed, use Write-EventLog to capture
important logging and debug information to the Windows Event Log
```

## Error Handling

**PowerShell functions must not return errors;** rather, they must throw exceptions in the same way that actual PowerShell errors do. Adaptable drivers treat exceptions thrown by a PowerShell function as a fatal error and then halt processing.

!!! danger ""

```
Thrown exceptions are handled as unexpected. If there is an error, we
recommended you use `Result="Failure"` and pass the error description in the
`Error=""` parameter.

Please write effective error messages that tell users what went wrong and how
to fix the issue.
```