API security

HERE Core API security enables desktop owners and application providers to determine the API calls that are available for a HERE Core program. Programs must declare in their manifest that they use security-sensitive APIs (*secured* APIs) for functionality that is outside the security sandbox, such as launching an external program or using fullscreen mode. While these features can be beneficial, HERE understands that desktop owners may need to restrict certain APIs from running on a desktop computer, particularly from an unknown or untrusted program. API security enables this by giving the Desktop Owner tools to prevent application developers from implementing features that may be deemed sensitive to an organization.

> "API" in this article can refer to individual functions or methods (such as System.openUrlWithBrowser()) or to entire Web APIs (such as Geolocation).

Historically, desktop owners have been able to block access to such APIs on a per-app basis, if a particular app is not trusted or should not have elevated permissions. However, this arrangement is *opt-out*: desktop owners must explicitly block particular apps from using particular API functions; all others are allowed.

Starting in v20 of the HERE Core Runtime, the *enhanced security* model prevents apps from accessing secured API functions unless the apps are explicitly granted permission to use them.

Enhanced security model

In order to leverage secured APIs, application providers must declare the use of specific API functions in their manifest file. This assists desktop owners to recognize API intent up-front. If an API function is not permitted by the organization or needs to be enabled for ease of program functionality, the desktop owner can define settings to block or allow use of the API function.

For versions of HERE Core Runtime prior to v20, if a desktop owner does not block a

program's use of a secured API, it is permitted. Starting in v20, if the desktop owner (or their delegate) does not explicitly allow a secured API for an program, it is *blocked* by default. See API security for desktop owners for details on how desktop owners can control access to secured APIs.

Versions of HERE Core between v20 and v24 automatically allow secured APIs if the runtime fails to connect to the RVM or if the RVM doesn't exist. Starting in v24, secured APIs are blocked if the RVM cannot be reached (due to messaging failures) or if the RVM doesn't exist (the runtime was launched from the CLI).

> As an alternative to controlling secured APIs through desktop owner settings, a program can use a trusted application configuration, which grants permission to use requested APIs after validation by the RVM. However, desktop owner settings for secured APIs, if defined, take precedence over a trusted application configuration.

Development exception

To simplify development, if the program is running from `localhost`, API security is the same as in v19 and earlier: Only the app manifest declaration is needed to access the API; no desktop owner setting or other permission is required.

For developers, this exception removes the need to take an extra step to allow APIs to be used. Be sure to test your program on a non-local server, and with both enabling and disabling the secured APIs it uses, to ensure that it behaves correctly under those circumstances.

Desktop owners can override this exception for developers' desktops by defining a specific registry key. If the key is not defined or is set to a non-zero value, the exception for `localhost` is allowed. See API security for desktop owners for details.

Secured APIs

In HERE Core, the following APIs are secured. Therefore, in order for a program to use them, they must be declared in the program's manifest, and be allowed by the desktop owner or their delegate.

API	Type
`Clipboard.write`	HERE Core 42+
`Clipboard.writeText`	HERE Core 42+
`Clipboard.writeHtml`	HERE Core 42+
`Clipboard.writeRtf`	HERE Core 42+
`Clipboard.writeImage`	HERE Core 42+
`Clipboard.readText`	HERE Core 42+
`Clipboard.readRtf`	HERE Core 42+
`Clipboard.readHtml`	HERE Core 42+
`Clipboard.readImage`	HERE Core 42+
`Clipboard.setClipboard`	HERE Core 42+
`System.checkCustomProtocolState`	HERE Core (RVM 12+, Runtime v34+)
`System.downloadAsset`	HERE Core
`System.getAllExternalWindows`	HERE Core (deprecated in Runtime v20; removed in v22)
`System.getOSInfo`	HERE Core (Runtime v38.126.82.64+)
`System.launchExternalProcess`	HERE Core
`System.launchLogUploader`	HERE Core (Runtime v42+)

API	Type
`System.openUrlWithBrowser`	HERE Core
`System.readRegistryValue`	HERE Core
`System.registerCustomProtocol`	HERE Core (RVM 12+, Runtime v34+)
`System.serveAsset`	HERE Core (Runtime v40.130.101.1+)
`System.terminateExternalProcess`	HERE Core
`System.unregisterCustomProtocol`	HERE Core (RVM 12+, Runtime v34+)
`ExternalWindow.wrap`	HERE Core (deprecated in Runtime v20; removed in v22)
`Application.getFileDownloadLocation`	HERE Core (Runtime v32+)
`Application.setFileDownloadLocation`	HERE Core
`audio`	Web
`video`	Web
`clipboard-read`	Web
`clipboard-sanitized-write`	Web (Runtime v23+)
`fullscreen`	Web
`geolocation`	Web
`midiSysex`	Web

API	Type
`notifications`	Web
`openExternal`	Web (from Electron)
`pointerLock`	Web

Declare APIs in a manifest file

A program must declare the secured APIs that it uses in its manifest file. If it attempts to use secured APIs that it hasn't declared, they are blocked. For complete details on defining a manifest, refer to Manifest settings.

The properties used to declare use of secured APIs depend on the version of the HERE Core Runtime being used. Runtime 37 and later version support domain-based permissions, where you can declare default permissions and exceptions based on domain match patterns.

In Runtime v36 and earlier versions, you declare permissions for the program and also for viewing contexts (windows and views).

In either case, regardless of Runtime version, you must list secured APIs within a top-level `permissions` object, categorized by the namespace they belong to (which may be `webAPIs`). For most APIs, the name is a key with a Boolean value, except Web APIs, where listing the API name in an array indicates that it is used.

For `System.openUrlWithBrowser`, the key is an object with two members:

- `enabled`, a Boolean for whether the method is used
- `protocols`, an array of custom protocols

For methods related to custom protocols, the permission can be either a plain Boolean or an object similar to the one used for `System.openUrlWithBrowser`; the `protocols` member in this case declares custom protocol schemes that are used with each method.

The methods related to custom procotols are the following:

- System.registerCustomProtocol()
- System.unregisterCustomProtocol()
- System.checkCustomProtocolState()

Domain-based permissions

In v37 or later, in addition to the top-level `permissions` object, you can define default permissions for nested content in the `domainSettings.default.api.permissions` object. If this object exists with no contents, then all secured APIs are blocked by default. If this object is not defined, then context-based permissions are used, as described in the next section. You can define exceptions to the default permissions based on domain match patterns, as described in Domain-based permissions.

Example manifest excerpt, with domain-based permissions

```
{
  "platform": {
...
    "permissions": {
    //These APIs are all used by the platform
      "System": {
        "launchExternalProcess": true,
        "openUrlWithBrowser": true,
        "registerCustomProtocol": true,
        "checkCustomProtocolState": true,
        "readRegistryValue": true,
        "terminateExternalProcess": true
      },
      "Clipboard": {
        "write": true,
        "writeText": true,
        "writeHtml": true,
        "writeRtf": true,
        "writeImage": true,
        "readText": true,
        "readRtf": true,
```

```
      "readHtml": true,
      "readImage": true,
      "setClipboard": true
    },
    "webAPIs": [
      "notifications",
      "audio",
      "video"
    ]
  },
  "domainSettings": {
    "default": {
    //By default, these APIs are blocked
      "permissions": {
        "System": {
          "launchExternalProcess": false,
          "openUrlWithBrowser": false,
          "registerCustomProtocol": false,
          "checkCustomProtocolState": false,
          "readRegistryValue": false,
          "terminateExternalProcess": false
        },
        "webAPIs": [
          "notifications",
          "audio",
          "video"
        ]
      }
    },
    "rules": [
      {
        "match": ["*://example.com/*"],
        "options": {
          "api": {
            "permissions": {
            //Domains that match the pattern can use these APIs
              "System": {
                "launchExternalProcess": {
                  "enabled": true,
                  "assets": {
                    "enabled": true
                  },
                  "downloads": {
```

```
                    "enabled": true
                  },
                  "executables": {
                    "enabled": true
                  }
                },
                "openUrlWithBrowser": {
                  "enabled": true,
                  "protocols": [
                    "slack",
                    "msteams"
                  ]
                },
                "registerCustomProtocol" : true,
                "unregisterCustomProtocol": {
                  "enable": true,
                  "protocols": [
                    "PROTOCOL1",
                    "PROTOCOL2"
                  ]
                },
                "checkCustomProtocolState": {
                  "enable": true,
                  "protocols": [
                    "PROTOCOL1",
                    "PROTOCOL2"
                  ]
                },
                "terminateExternalProcess": true
              },
              "webAPIs": [
                "notifications",
                "audio",
                "video"
              ]
            }
          }
        }
      }
    ]
  }
}
...
```

```
}
```

Context-based permissions

In v36 and earlier, the location of the `permissions` object varies, depending on the way the program uses HERE Core APIs.

- For programs that use the Platform API, you can set permissions inside the following objects:
  - `platform`: The `permissions` object declares APIs for the platform only (and not for windows or views). If you specify a `providerUrl`, then this URL is able to execute the enabled permissions.
  - `defaultWindowOptions`: The `permissions` object declares APIs for every window launched.
  - `viewDefaultOptions`: The `permissions` object declares APIs for every view launched. Be sure to consider whether you load third-party content into views, which might also use APIs.
- `startup_app`: Use this object for apps that do not use the Platform API.

Example platform manifest excerpt with context-based permissions

```
{
  "platform": {
...
    "permissions": {
      "System": {
        "launchExternalProcess": true,
        "openUrlWithBrowser": true,
        "registerCustomProtocol" : true,
        "unregisterCustomProtocol": true,
        "checkCustomProtocolState": true,
        "readRegistryValue": false,
        "terminateExternalProcess": true
      },
      "webAPIs": [
```

```
      "notifications",
      "audio",
      "video"
    ]
  },
  "defaultWindowOptions": {
    "permissions": {
      "System": {
        "launchExternalProcess": {
          "enabled": true,
          "assets": {
            "enabled": true
          },
          "downloads": {
            "enabled": true
          },
          "executables": {
            "enabled": true
          }
        },
        "readRegistryValue": false,
        "terminateExternalProcess": true
      },
      "webAPIs": [
        "notifications",
        "audio",
        "video"
      ]
    },
    "defaultViewOptions": {
      "permissions": {
        "System": {
          "launchExternalProcess": {
            "enabled": true,
            "assets": {
              "enabled": true
            },
            "downloads": {
              "enabled": true
            },
            "executables": {
              "enabled": true
            }
```

```
            },
            "readRegistryValue": false,
            "terminateExternalProcess": true
          },
          "webAPIs": [
            "notifications",
            "audio",
            "video"
          ]
        }
      }
    }
  }
}
```

Example manifest excerpt with context-based permissions

```
{
  "startup_app": {
...
    "permissions": {
      "System": {
        "launchExternalProcess": true,
          "openUrlWithBrowser": true,
          "registerCustomProtocol" : true,
          "unregisterCustomProtocol": true,
          "getCustomProtocolState": true,
          "terminateExternalProcess": true
        },
        "webAPIs": [
          "notifications",
          "audio",
          "video"
        ]
    }
  },
...
}
```

Query secure API permissions

You might want your app to take preemptive action when certain capabilities are not available. For example, a **Launch External App** button could be disabled when that capability is not permitted.

Starting in v21, the `System.queryPermissionForCurrentContext()` method returns the permission state of a specific secured API. By using this method, an app can alter the UI to disable or hide features that cannot be performed due to the permission status.

Example usage

The `System.queryPermissionForCurrentContext()` method has one parameter, the name of the secure API to query.

Here is what it looks like to use it:

```
System.queryPermissionForCurrentContext('System.launchExternalProcess'
=> {
  if (result.granted) {
    renderLaunchExternalProcessButton();
  }
});
```

The shape of the return value is `QueryPermissionResult` which is as follows:

```
interface QueryPermissionResult {
  permission: string,
  state: 'granted' | 'denied' | 'unavailable',
  granted: boolean,
  rawValue?: unknown
}
```