Oracle® Food and Beverage Oracle Linux for MICROS Integrators

Reference Guide

Release 19.x
F46324-01
September 2021

Oracle Linux for MICROS Integrators Reference Guide 19.x

F46324-01

Contents

Preface

The latest version of Oracle MICROS Simphony now runs on Oracle Linux for MICROS. Integrators wishing to run code on Workstations running Linux will want to take advantage of the unique capabilities offered by the platform, and will need to adhere to the constraints inherent by the platform.

Purpose

This guide explains those capabilities and constraints, starting with the foundations of the operating system platform, its genesis and business goals, and then moving on to the best practice guidelines to facilitate successful development and deployment of code extensions to Simphony on a workstation running Linux.

Audience

This document is intended for Integrators developing solutions to extend the capabilities of Simphony by running code on Oracle Workstations running Linux. This document does not apply for third party workstation hardware.

Customer Support

To contact Oracle Customer Support, access My Oracle Support at the following URL:

https://support.oracle.com

When contacting Customer Support, please provide the following:

- Product version and program/module name
- Functional and technical description of the problem (include business impact)
- Detailed step-by-step instructions to re-create
- Exact error message received
- Screen shots of each step you take

Documentation

Oracle MICROS Simphony product documentation is available on the Oracle Help Center at http://docs.oracle.com/en/industries/food-beverage/

Table 1 Revision History

Date	Description
September 2021	Initial Publication

1 Introduction

The latest version of Oracle MICROS Simphony now runs on the Oracle Linux for MICROS operating system, offering restaurants a simpler platform on which to run their business.

One of the many benefits of Oracle Linux is the lack of Microsoft Windows updates, which slow down workstations and consume valuable hard-drive space. With Linux, the maintenance is streamlined; Oracle manages the hardware and provides a single channel for operating system, software, hardware, maintenance, and support. Linux updates are sent down via the Client Application Loader (CAL), the same tool that distributes Simphony software updates. The lightweight Linux OS requires less computing power, freeing up more resources for Simphony and Integrator solutions to use, resulting in an improved point of sale performance.

Scope of Document

This document serves as a guidance for integrators of the Oracle Linux for MICROS Workstation Platform. It covers principles and guidance in the following areas:

- Oracle Linux for MICROS Platform Benefits and Principles
- Appliance Mission: Appliance vs Custom Model
- Hardware Considerations
- Runtime Environment
- Deployment
- Select Use Cases

2 Oracle Linux for MICROS Platform

Oracle Linux for MICROS was introduced in Simphony 19.1. Some features useful for Integrators were added in Simphony 19.2, notably the HTML5 Custom Dialogs.

Why Linux?

Several factors motivated the move to Linux for Simphony:

1. Simple platform in comparison to the multiple Microsoft Windows platforms currently supported.
2. Lower hardware resource requirements; enables Oracle to support older, and/or more cost-effective hardware, such as the Workstation 5A.
3. Lower licensing cost.
4. Alignment with industry and company direction.
5. Provides an upgrade path for RES 3700 & E7 customers using existing Workstations.
6. Enhanced security posture.

Key Benefits

- A complete solution (including hardware, software, and operating system) designed, maintained, and supported by Oracle.
- Oracle now manages and delivers platform updates through the Client Application Loader (CAL).
- Linux reduces computing resource, improving point-of-sale application performance.
- Appliance-like environment creates a more user-friendly, secure, and dedicated point-of-sale experience. This eliminates the potential for users to run other applications on the point-of-sale hardware.
- Reduced cost and time for implementation.
- Supports conversion path for the Workstation 5A series, providing customers that use the most popular legacy MICROS workstation a pathway to Simphony without having to upgrade hardware.

An Appliance

The Oracle Linux for MICROS platform was designed with the goal of an operating platform that acts and performs as if it was an appliance, with every device manufactured and provisioned identically, including the base OS image and installed Simphony software. Customizations are allowed only in strict circumstances, with defined constraints for specific intended use cases.

Appliance vs Custom Model

Consider the two different models for the Linux based Workstation. In one model, the device is an appliance, signifying no changes and no customizations beyond the Oracle signed software. Enabling this simplicity is one of the motivations for the new Linux platform. To be considered an appliance, or to exhibit appliance like behavior and benefits, requires only core Simphony software on the device and Oracle approved and certified components.

The following section is a high-level overview between the two models, explaining the constraints of the Appliance Model, and the flexibility offered by the Custom Model.

At this time, the Appliance vs Custom model distinction is an unenforced concept that is subject to change in the future. We suggest that Integrators and customers decide which model they wish to adhere, to maintain compatibility moving forward.

Appliance Model

Will only run:

1. Core Simphony software installed by the CAL software loader.
2. Oracle signed extensibility software installed by the Oracle CAL software loader. Example: Fiscal extensibility apps.

Locked down security model.

Custom Model

The Custom Model includes the following capabilities and characteristics:

- Download and use custom CAL packages, all extensibility apps, and SIM scripts.
- EMC controls authorizing and signing extensibility apps and SIM scripts. In other words, responsibility and control for authorizing and signing custom software is expanded to include not only Oracle, but also a high privilege EMC user.
- Hashing verification of custom software package downloads.
- Ability to revert to Appliance Mode with a reimage.

Genesis of Oracle Linux for MICROS

The OS image used for Simphony on the new Linux workstations is a custom image of Oracle Enterprise Linux. Much of the customization involves hardening the security posture and slimming it down, which results in removing unnecessary components and allowing a more compact OS image to preserve RAM for the Simphony application, database, and associated components.

The outcome is a custom image known as Oracle Linux for MICROS. This section covers the principles, guidelines, and constraints imposed by the choices made for the custom image. The OS image is the same across all workstations.

Principles and Constraints

The following is a list of principles and constraints for Oracle Linux for MICROS in a production environment.

1. No desktop installed. The Linux workstation does not use a desktop; the POS display runs in a single dedicated X session. Standard Microsoft Windows practices, such as minimizing the application to access the OS or run a file explorer, are unavailable. Linux integrators must use the Linux shell to troubleshoot issues. Furthermore, integrators cannot install their own GUI applications on the Linux appliance.
2. No on screen keyboard available.
3. No interactive root login available.
4. All software run under unprivileged “posuser” account with the following exception:
   - COM port enabled (including USB virtualized COM port)
5. No direct access to the database. Data access is protected by security policy. Data access for integrators is provided solely through published Simphony APIs.
6. User interface is in HTML5; no XAML/WPF available within Mono.
7. Certificate handling on Linux with Mono is different.
   - This is relevant for Integrators connecting to external TLS endpoints. Refer to the section on certificate handling for more information.
8. Unmanaged Microsoft Windows code do not work under Mono on Linux.
   - Microsoft Windows specific .NET assemblies do not load on Mono.
   - Unmanaged code is possible outside Mono, e.g. for extensibility. Microsoft Windows extensibility applications can call into native Microsoft Windows DLLs. Linux extensibility applications can call into native Linux DLLs. Documentation exists on the internet outlining how to use the mono Interop facility (search “*linux mono call native dll*”).

9. Simphony 19.2 HTML5 dialog enhancements. See Chapter 4 for more information.
10. The Oracle Simphony CAL must install everything.

3 Workstation Hardware

There are two relevant categories of workstation hardware, primarily based on available RAM. Most workstations have 2GB+ or RAM, which is enough to allow integrators the suitable space to run most solutions.

However, for Workstation 5A, its RAM is extremely constrained (512MB) and only supports the most lightweight integrator solutions with very small RAM footprints. This section lays out the differences and notes the constraints between the different workstations.

Workstation 3xx/4xx/6xx

This section covers all Workstations supporting Linux that are not the Workstation 5A.

That includes the following:

- Workstation 610, 620, 650
- Compact Workstation 3 Series
- Workstation 625E, 625X, 655X
- KDC 210
- Express Station 4xx Series

Relevant characteristics and hardware access for these workstations include:

- Memory: 2GB on KDC unit. 4GB-8GB on 3xx/4xx/6xx.
- Storage: 64GB – 256GB.
- 64-bit BIOS required. See upgrade docs for instructions.
- Standard port access (Serial, USB, IP). See guidelines in 'Use Cases' section.
- Proprietary I/O support (cash drawer, customer display, mag card reader). See guidelines in 'Use Cases' section.

Workstation 5A

The Workstation 5A is a special case, due to the limited memory size, and extremely limited amount of RAM available for integrator use (approximately 30MB). This available RAM space varies depending on deployment configuration. Because of this, consider the Workstation 5A as an appliance, with limited to no ability to customize outside core Simphony features and components.

Differences between the Workstation 5A and other hardware include:

1. 512MB RAM total.
2. SQLite database instead of MySQL.

- While this is a distinction, it is irrelevant to an integrator since they are unable to access the database in this case. It helps Simphony core components to fit within the 512MB total memory available.

3. CAPS and KDS Controller do not run on a Workstation 5A.
   - Instead, CAPS and KDS Controller run on another machine, with a greater amount of RAM, allowing more RAM headroom to run integrator solutions. Due to this, another machine must run integrator solutions (for the entire location, and not on every workstation on location).

4 Runtime Environment

The Simphony extensibility is the primary avenue through which Integrators will develop code to run on Simphony workstations.

SIM / Extensibility Platform

Linux integrators should be mindful of the platform differences between Microsoft Windows, Linux, and Android platforms, as summarized here:

Feature	Microsoft Windows	Linux	Android
OpsContext core dialogs	X	X	X
HTML5 UI/dialogs	X	X	X
SIM	X	X	X
.NET Extensibility	X	X	
WPF dialogs	X		
.NET WinForms dialogs	X		
Custom Styles	xaml	CSS	CSS
.NET Framework	4.6.2	Mono 6	
C/C++ Native DLLs	X		
C/C++ Native Executables	X	X	

Custom Dialog Extensibility

Custom HTML5 Dialogs in Extensibility can be assumed to be available in all platform deployments (denoted by the "HTML5 UI/dialogs" line in the above table). This module was created in Simphony 19.2 specifically to allow cross-platform integration compatibility.

Simphony extensibility (SIM/C#) has traditionally only had very primitive user interface functionality. The SIM "Window" feature is limited to form entry and C# has no built-in user interface functionality.

Microsoft Windows extensibility applications are able to use .net forms/WPF to build complex user interfaces. These technologies are not available on Linux or Android.

Simphony 19.2 introduced a new feature which provides an extensibility API for displaying complex dialogs across all platforms.

The technology to achieve this is HTML5 running isolated in an iframe or separate browser instance. The benefits to this design are:

1. HTML5 is a standard and familiar technology, with industry wide population of HTML5 capable developers.
2. HTML5 renders rich, animated UI experiences.
3. HTML5 is platform-independent. HTML5 dialogs render identically on Microsoft Windows, Android, and Linux.
4. Isolating the HTML5 dialog gives the extensibility a safer sandbox to execute code.
5. The renderer is CEF (Chromium embedded framework). This technology is standard for most browsers.

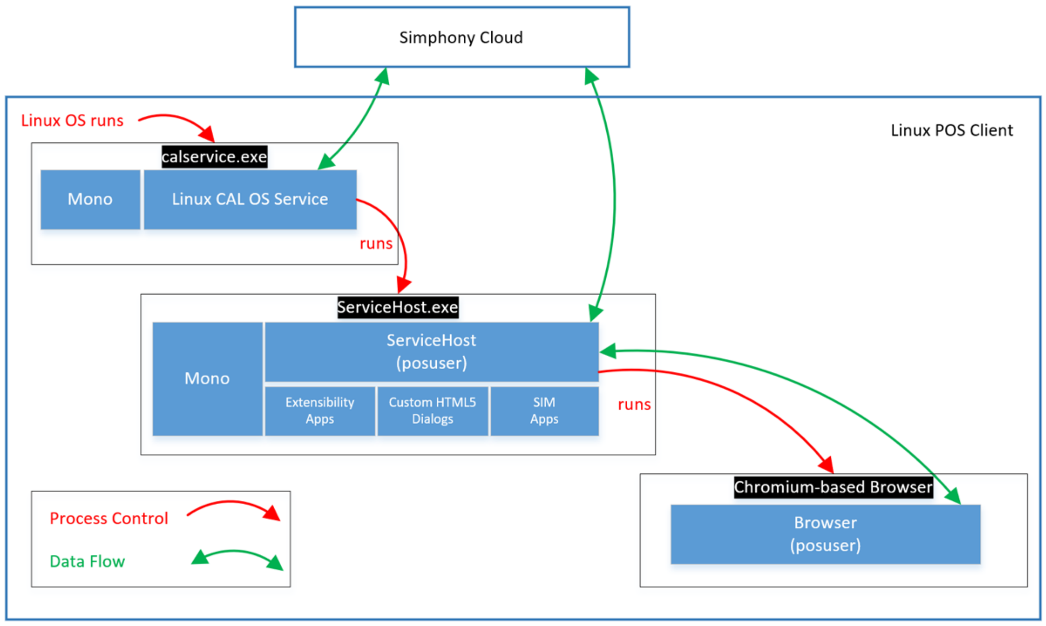
HTML5 Extensibility API Details

More information on this topic are available in the HTML5 Extensibility Developer Reference Guide. The document covers the following topics:

- API Basics
- Raising the Dialog, with sample code
- Example HTML5: simple dialog, functioning centered dialog, dialog styling, database image references, custom and unknown resource event
- Extensibility Callbacks: calling method, results status, system keyboard
- Web Directory Dialogs

Application Launch

- The CAL Service is the only process automatically started by the OS when the Workstation boots. At startup, CAL checks to see if any updated packages are available before its launches the *autostart* apps. Do not install or run any other OS service or daemon process.
- The CAL service runs applications using the unprivileged 'posuser' account.
- The CAL allows multiple applications to be launched (for example, Service host and KDSController). This is an enhancement from Microsoft Windows CAL which only allows a single app to be automatically launched. However, unlike Microsoft Windows, there will only ever be a single instance of ServiceHost itself.



Considerations on Oracle Linux for MICROS

1. In Linux, all UI interaction is through a Chromium browser showing Ops or the KDS Display in a single X display (there is no Desktop or Windows manager). XAML/WPF is not available.
2. The Linux client uses mySQL (or SQLite - see below) for its database.
    a. For the Workstation 5A, Linux computing resources are low, and SQLite operates as a database. Both may restrict possible functionality.
3. Managed 'non UI' code moves quite well to Linux but it may need review, as Mono is not a 100% implementation of the .Net Framework.
4. Microsoft Windows native code / native DLLs do not transfer directly to Linux.
5. IP network ports lock down, but can open for use in cases that need them via custom CAL packages. Refer to the CAL reference document for details on opening additional ports. Standard open ports are documented in the Simphony Security Guide.
6. TLS Certificate handling is different on Linux. Refer to the section below for more details.
7. Do not run anything with a root privilege. The POS apps only run as the restricted posuser.
8. Security-Enhanced Linux (SELinux): Enabled by default in *Enforce* mode.
9. Workstation firewall locked down with ‘deny all’ by default, and a collection of ‘allow’ entries for specific enumerated needs. This includes outbound as well as inbound traffic.
10. Linux does not allow more than one ‘UI’ application to be started (that is, KDS Display and ServiceHost applications cannot run concurrently.

11. There are six Linux virtual consoles available:
    a. VC1 and VC2 are reserved for CAL and Simphony.
    b. VC3 is reserved for future Oracle use.
    c. VC4 through VC6 are available for use by Integrators.
    d. VC Switching: During the development process, virtual console switching can be accomplished with Ctrl Alt F1-F6 on a physical keyboard.
12. 64-bit OS:
    a. Note 64-bit BIOS required. See hardware upgrade docs for instructions.
13. There is no native registry on Linux. Mono provides registry emulation using XML files.

Certificate Handling

This section is relevant for Integrators who connect to external TLS endpoints that have non-standard certificate utilization not covered by pre-installed certificates.

Note: Changing the security posture of the workstation is a privileged operation and should be carefully considered.

Under Microsoft Windows, .NET integrates with the Microsoft Windows certificate store. In Linux, there is no single Certificate Store. Simphony focuses on TLS communications using the Mono framework. Current Mono versions (v6 in our case) use Boring TLS for TLS communications by default, and this has its own Certificate storage and structure. Additional low-level details are out of scope for this document, and are not needed for Integrators. The following sections are a summary for the Mono scenario and some others.

Best Practices and Guidance:

1. Mono 6
    a. Add cert to ‘system’ Certificate Store as below
    b. Run Mono utility *cert-sync* which will synchronize to Mono Store
2. Chromium
    a. Obtain Root CA certificate (and intermediate certificate if applicable)
    b. Convert to PEM format (use *openssl* if required)
    c. Use *certutil* to import into NSS store
3. System Certificate Store (assuming suitable CAL permissions)
    a. Get Root CA certificate (and intermediate certificate if applicable)
    b. Convert to PEM format (use *openssl* if required)
    c. Put PEM format version of the certificate into */etc/pki/ca-trust/source/anchors*
    d. Run *update-ca-trust*

Note: Oracle does not recommend use of self-signed certificates in production workstations.

Standard Linux Considerations

This is a list of notable differences when moving Windows Integrator code to run on Linux:

1. Linux is case-sensitive for pathing and file names.
2. Directory separator, use *System.IO.Path* to generate paths.
3. Runtime directory is different for Linux/Windows.
4. Linux user permissions lock down more than Windows.
5. Path names use only forward slashes.

5 Deployment

As it is the only method with privileges to install software, the Oracle Simphony CAL (Client Application Loader) must install everything. CAL packages are a logical grouping of loose files that reside on the CAL application server. These files typically include client software that needs copying and/or executing. In Simphony, all packages exist in the database.

Installing Packages

A special UTF-8 encoded text file named setup.dat is required for each CAL package. This file contains all of the instructions needed to perform the installation of the software included within a package.

CAL client requests this file for any package that requires installation on the client device. After the file transfers locally, each line of a setup.dat file processes synchronously until reaching the end of the file.

Lines beginning with a recognized command process immediately. All operations associated with the command found in the current line must complete successfully before the next line processes. Some commands can override this behavior. If a command did not complete successfully, the entire package installation aborts. Once a package installation has failed, the setup.dat processes from the beginning of the file once the CAL client (or device) restarts.

Lines are ignored without generating an error when:

- they begin with a comment character
- the command name is unrecognized
- the required parameters for the command are not provided

Platform Differences

Application Root

This is a significant change from Windows for Integrators, and aligns with the Appliance concept mentioned earlier.

Background

- Windows CAL allows the install folder to be defined using a combination of setup.dat commands and the 'Application Root' field in the CAL UI. The install location can be changed at any time, which potentially can lead to inconsistent installs with software scattered around the system.

Linux Application Root

- The Change: All software will be installed in a fixed, predefined directory: */opt/oracle/simphony*.

Integrator Root

Integrator binaries and all data files will exist under the Integrator root directory: */opt/{integrator}*, where *{integrator}* is the integration company or product name. Within this directory, there are the following sub-directories:

- Binaries shall be placed in /opt/{integrator}/bin
- Logs shall be placed in /opt/{integrator}/log
- Temp files shall be placed in /opt/{integrator}/tmp
- Persistent data (including databases) shall be placed in /opt/{integrator}/data

We recommend that Integrators regularly perform the following:

- Remove old binaries after upgrade
- Remove temp files when done with them
- Trim logs
- Auto-purge databases

Failed Packages

Failed packages are handled differently between the Linux and Microsoft Windows platforms:

- On Microsoft Windows, a failed package displays an error message, which requires the user to dismiss manually before any package processing can continue.
- On Linux, a failed package does not display any error message if another application has taken control of the display. Failed packages are still marked as failed, but any subsequent packages requiring installation continue processing without user intervention.

General Syntax

The CAL client utilizes a simple language with only two types of production rules: commands and single line comments. The grammar for a single command consists of the command name and up to four parameters, all delimited by a ‘,’ character, as shown below:

[COMMAND NAME],[parameter1],[parameter2],[parameter3],[parameter4]

Command names are predefined case sensitive values. Each command may have zero or more required and/or optional parameters. Leading commas are only required for command parameters.

Comments start with the ‘#’ character followed the comment text.

CAL Startup Commands Reference

CAL Startup command details are available as a reference document on MOS.

Additional Notes

1. Cannot install a custom Linux OS service.
2. Workstation IP ports can open via a custom CAL package. Refer to CAL documentation for details on opening ports.

6 Development Strategies

Extensibility application developers can write their extensions for the Simphony POS using .net assemblies or SIM. SIM is available on all platforms: Microsoft Windows, Linux, and Android. .NET assemblies are available only on Microsoft Windows and Linux. It is important to consider that .NET assemblies do not load on Android.

Multi-platform .NET Assembly Development

It is possible to write a single .NET assembly that will run on both Microsoft Windows and Linux. As with any multi-platform development, make sure to either abstract out any OS-specific logic in the same assembly, or refactor into a different assembly.

If you are starting an extensibility application from scratch, then we recommend partitioning the code to account for platform differences.

Porting from Microsoft Windows to Linux

If a Microsoft Windows application already exists, porting it to run on both Linux and Microsoft Windows may be a challenge. This is true for all multiplatform development, and not just Simphony.

Many applications are replete with Microsoft Windows-specific code, from trivial (directory separators) to architectural (dozens of WPF dialogs and Microsoft Windows device drivers). While it may be desirable to maintain one assembly for your application, time and resource constraints may force the application to have a Microsoft Windows assembly and a Linux assembly.

Scenario: A developer develops an extensibility application for Microsoft Windows and now wants to port it to Linux. The DLL will not run on Linux as-is because it references Microsoft Windows-specific assemblies. Possible solutions are:

1. Create two separate versions of the same application and mark them as Microsoft Windows-only or Linux-only in the EMC configuration.
2. Create one application assembly that runs on both Linux and Microsoft Windows, and abstract out the OS-specific functionality into separate assemblies. (The Simphony core code uses this strategy).
3. Make runtime decisions in code based on OS type.
4. Take advantage of .NET runtime for directory separator, combining paths.

There is no universal best practice strategy for refactoring assemblies to run on Linux and Microsoft Windows. The decision depends in part on how tightly the application is bound to Microsoft Windows and how much effort can be devoted to refactoring.

Building a Cross-Platform Assembly

Troubleshooting Strategy

Developing and troubleshooting extensibility applications is more difficult on Linux than Microsoft Windows. Frequently, extensibility assemblies have Microsoft Windows-specific references but they are hard to reconcile. To determine if code will load on Linux, we recommend downloading the mono compiler on Microsoft Windows and building the extensibility application solution. Any Microsoft Windows-specific references then result in a compile-time error.

In general, if the same extensibility solution successfully compiles using Visual Studio and Mono, then it should load on both platforms. The custom HTML5 dialog feature works identically on Microsoft Windows as it does on Linux. Speed up development by first testing the custom dialog on Microsoft Windows and then on Linux.

7 Use Cases

This section highlights the best practice guidance relevant to common Integrator use cases. Suggestions listed here are not a substitute for understanding the entirety of the guidance in the rest of this document. Rather, this is intended to point out a few additional items worthy of consideration when developing for these use cases, and some relevant cautions.

Hardware Integrations

For Integrators accessing and using hardware peripherals connected to Linux Workstations, such as:

- Coin changers
- Scales
- RFID readers
- Liquor dispensing/tracking
- Scanners
- Automated coffee machines

Communication to connected hardware peripherals and food prep hardware with APIs can be accomplished through standard device ports (serial, USB, IP), or through proprietary I/O. The following sections provide more detail.

Standard ports (Serial, USB, IP)

- Use standard Linux access. IP open ports are limited. Refer to the Simphony Security Guide for specifics.
- The port range from 8200-8400 is reserved for integration use.
- Access for additional ports can be enabled in one of two ways:
    1. Through custom CAL package. See CAL discussion elsewhere in this document for details.
    2. EMC Workstation configuration.

Proprietary I/O

Examples: cash drawer, customer display, mag card reader.

Best practice recommendation and notes:

- *Concurrent Access*: Only access peripherals for which you are the only application accessing that peripheral. Access conflicts will occur if two different applications or components attempt to access the same peripheral concurrently. This includes attempting to access a peripheral that the Simphony application itself is configured to utilize.

API Options:

- PCWS API (Preferred).
  - Please note *Concurrent Access* comment above.
- Updated JavaPOS drivers. Allows Integrators using Java to use industry standard calls rather than integrate to PCWS API.
  - Please note *Concurrent Access* comment above.
- There is no OPOS for Linux.

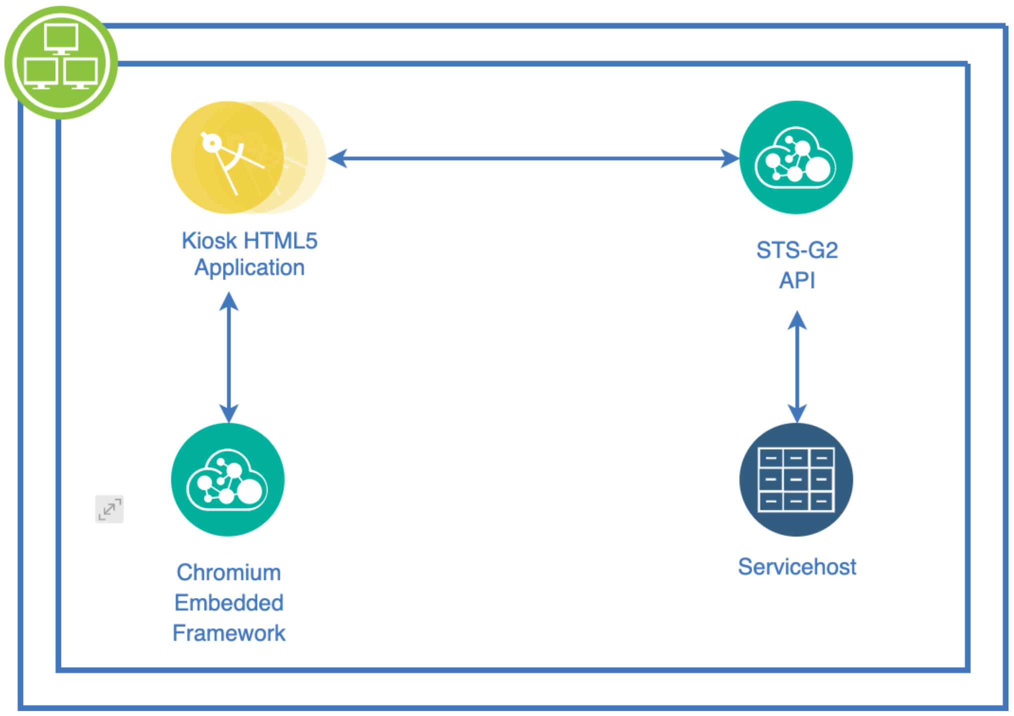
Kiosk

A kiosk integrator builds their solution directly on top of a platform similar to a headless Simphony workstation, comprising of Oracle hardware, Oracle Linux for MICROS, and Simphony. The differences that make it a kiosk include:

- Custom CAL package with Integrator kiosk application
- Simphony Ops UI configured to not start on boot. Configured in EMC.
- Chromium will start kiosk URL on boot. Configured via custom CAL.

The Integrator developed app runs on the above platform, and utilizes the following UI and API pattern:

- UI via HTML5 on Chromium browser (Chromium Embedded Framework).
  - This is a direct HTML5 in Chromium, and not a Simphony HTML5 Custom Dialog Extensibility.
- STS API endpoint consumption on local host. STS Location API can be enabled for the kiosk workstation through EMC.

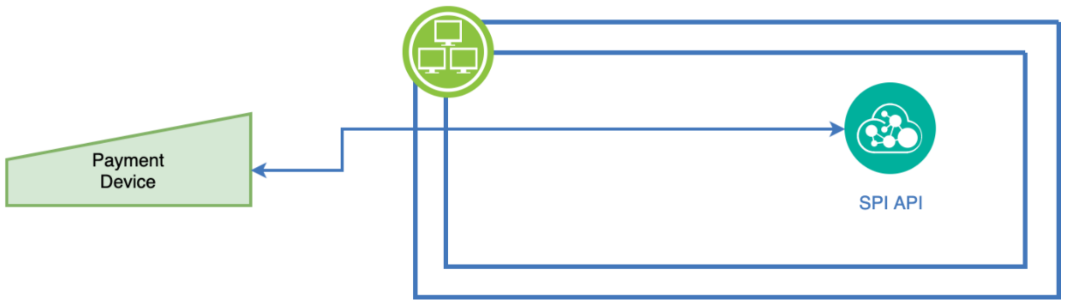


Payment

Payment integrations complete by consuming the Simphony Payment Interface (SPI) API.

No Footprint Solution

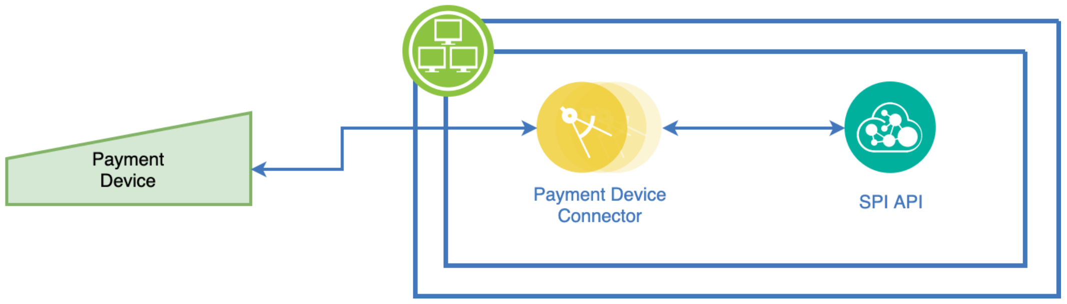
In the preferred no footprint approach, there is no integrator code footprint running on the workstation. The payment device interfaces directly to the SPI API through a network IP connection on the workstation. This approach is suitable when interfacing to the resource constrained Workstation 5A.



Payment Connector Solution

If the no footprint solution is not feasible, a lightweight extensibility object acting as a payment driver/connector between the payment device and SPI. On the payment device side, connect using the ‘Hardware Integrations’ use case pattern. On the SPI side, consume the standard SPI API.

Note: Lightweight needs to be quite light if the Workstation 5A is a target hardware platform. See the Workstation 5A section for more details on constraints, especially due to the limited available RAM for integrations.


Payment Device
Payment Device Connector
SPI API