Manatee Integration Guide

September 15, 2024

Contents

Introduction

One of the primary usecases for Manatee is integration of applications running on your desktop. Integration may take many forms but a often used approach (within healthcare) is by using the CCOW standard.

The CCOW standard provides a framework for build a "shared context" in which applications can agree on a shared state. This can be used to ensure that e.g. the same patient is selected in a suite of applications running on the same desktop. CCOW also specifies functionality to allow one application to send a message to another to trigger some internal state change e.g. causing the application receiving the message to trigger an internal function like navigating to specific view.

In order to integrate an application with Manatee you first have to consider what type of integration you need and then the application itself.

Integration types

When we talk about integration types we mean the functionality you want from your integration. We'll go through a few base cases here:

State sharing

State sharing is the standard CCOW functionality. You want you applications to share a part of their state and have that state synchronized across applications. An example here is sharing the patient and when you change the patient in one application you want all applications to switch to the same patient. This is the typical two-way state synchronization approach, but you can also vary this s.t. one application simply shuts down when e.g. the patient changes if for instance it is not possible to change patient here.

Launch with context

Another type of integration is to launch a secondary application from within a given other application. The newly launched application may have some context in which it is started e.g. showing the same patient as the application from which it was launched.

Navigate to internal view

Yet another example similar to launching with context is if you want to navigate to some internal view given a key (like a patient identifier) from one application into another. A healthcare example here is to select a patient identifier and then open e.g. the EMR for that patient.

These are the most typical examples other integration types are possible.

The target application

When considering which integration approach to take then the application itself is the most important aspect. First and foremost consider whether it is a possibility to make changes to the application itself or whether the application exposes an interface (API) that you can access. If neither of these options exist then an approach involving automation of the application is the obvious choise. Otherwise an integration where you augment the application to interact with Manatee using its external API or wrap the application in a .NET based solution using our plugin API.

Approaches to integration

Automation

Using our automation solution you can augment closed-source applications (web, java, native and others) with functionality s.t. they can be launched in a given context, integrate in context sharing and can handle custom actions (e.g. navigation).

Implementing **state sharing** is done by writing small snippets of code that interact with and read from the user interface of the application. A simple example is enabling an application to participate in a two-way context sharing session.

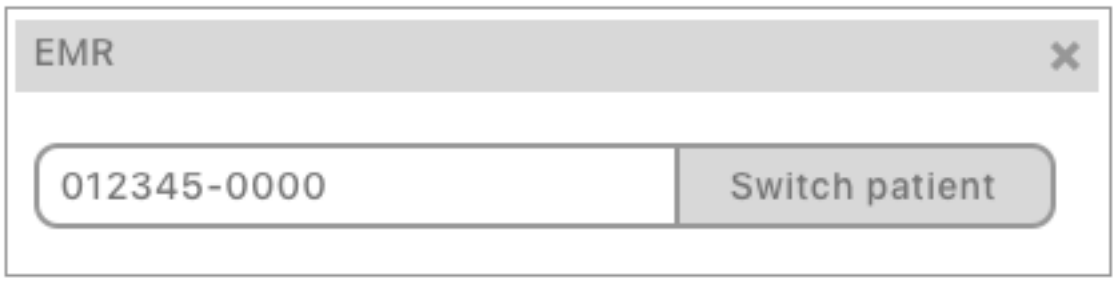

Figure 1: Our simple example application

To accomplish this you would need to write two code snippets; one to read the current value of the state from the application and one to set the state of the application to the desired value. Consider a simple application with a text-field that displays the current patient identifier and a button to switch to a patient if you change the identifier. The two snippets would then look like:

```
1 Value = new Field("**/PatientIdentifier").read();
```

Which creates a *Field* that represents the patient identifier text-field and then reads the content. Manatee would then take care of updating the shared context and all the applications that participate in the context when the patient identifier changes. To be able to switch the patient here if another application changes the patient in the shared context you would do:

```
1 new Field("**/PatientIdentifier").input(Value);
2 new Field("**/Switch patient").click();
```

Which simply writes the new patient identifier into the text-field and then clicks the button to actually switch the patient. It is also possible to implement one way sharing where an application can only read or only write its state. This is done by simply not configuring the relevant code snippet.

To implement **launching the application** you would simply need to define the application with the proper arguments in our configuration UI, e.g.:

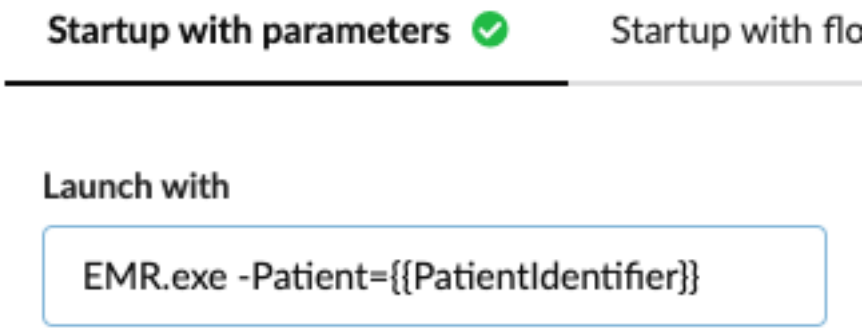

Figure 2: Path to the application and the arguments

Here the `PatientIdentifier` (enclosed in double curly brackes) refers to the value of the patient identifier stored in the shared context.

Navigating to an internal view is done with small code snippets similarly to how state sharing was implemented.

Using automation as an integration mechanism is a cheap and fast way to integrate an application in a shared CCOW context and works for most applications. For state sharing the state that you need to synchronise must be available in the user interface of the application in order for two-way sharing to be implemented.

Using the Manatee external CCOW API

If you are able to modify source code of the application, then you can use our external API to interact directly with the CCOW context manager that is responsible for coordinating state changes and invoking actions on participating applications. We support the following protocols;

- JSON-RPC over WebSockets
- JSON-RPC over NamedPipes
- JSON-RPC over the NetString protocol (raw TCP)
- gRPC
- Standard HTTP CCOW mapping
- Standard ActiveX CCOW mapping

Using these protocols you can implement state sharing, navigating to internal views and other actions as custom tasks in the target application and since you have access to the full internal state of your application you don't rely on the user interface to display the information you need.

The main drawback of this approach is that it tends to be costly since you often need to involve the application vendor and get custom development done.

Implementing a custom wrapper

If the target application exposes an API that has the handles you need for your use-case (read/write the required state and/or invoke the required actions like navigation), then you can write a custom wrapper in .NET (for which we have a small API) or any other language/platform (when using the APIs described above) which then maps between the APIs.

This is also something you can contract out since no special target application knowledge or access might be needed. This is likely still a costly task but probably not to the same degree as when the target application must be modified.