BRINGING CSOUND TO A MODERN PRODUCTION ENVIRONMENT WITH *CSOUND FOR LIVE*

Mark Jordan-Kamholz
mjordankamholz AT berklee.edu

Dr. Richard Boulanger
rboulanger AT berklee.edu

Csound is a powerful and versatile synthesis and signal processing system and yet, it has been far from convenient to use the program in tandem with a modern Digital Audio Workstation (DAW) setup. While it is possible to route MIDI to Csound, and audio from Csound, there has never been a solution that fully integrated Csound into a DAW. *Csound for Live* attempts to solve this problem by using csound~, Max and Ableton Live. Over the course of this paper, we will discuss how users can use Csound for Live to create Max for Live Devices for their Csound instruments that allow for quick editing via a GUI; tempo-synced and transport-synced operations and automation; the ability to control and receive information from Live via Ableton's API; and how to save and recall presets. In this paper, the reader will learn best practices for designing devices that leverage both Max and Live, and in the presentation, they will see and hear demonstrations of devices used in a full song, as well as how to integrate the powerful features of Max and Live into a production workflow.

1 Using csound~ in Max and setting up *Csound for Live*

Rendering a unified Csound file with csound~ is the same as playing it with Csound. Sending a *start* message to csound~ is equivalent to running Csound from the terminal, or pressing play in CsoundQT. The main difference is that when making Csound for Live devices, we're mainly concerned with an orchestra that is controlled by knobs, sliders, buttons, and automation, rather than with an orchestra that is driven by a score, or internal modulation and event triggering. Thus, the .csd file must be set up to receive data in several ways from Max, and there are several things to consider when setting up the file.

First, make sure that your output flag is set to -odac. If you don't do this, Csound will write to an audio file in addition to playing through Max and Live, and depending on how long you might be jamming, this could slow down your system and you could end up with an audio file several gigabytes in size! Next, set an ftable, usually f0, to be

loaded in and at a very large number of seconds, this will keep Csound running for hours without quitting.

Setup in Max involves doing 3 things. After instantiating csound~, you will have to load a unified csd file, choose a sample rate, and then start Csound. See figure 1 for an example of a patch that does these things.

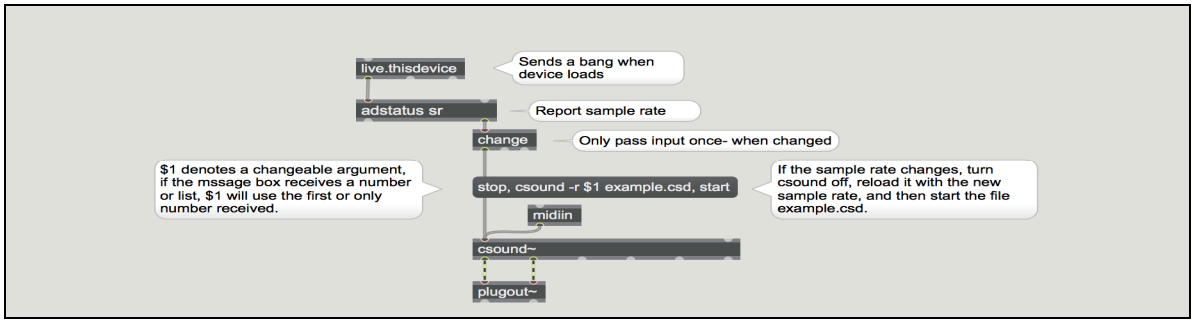

Figure 1 Loading a .csd file into csound~

Finally, it is advisable to use ksmps instead of krate. Given that the user can change the sample rate at any time, having a krate that is proportional to the sample rate means that krate signals are synced to the sample rate, and are more likely to work as intended.

2 Creating a UI

When designing an instrument or effect using csound~, it is often helpful to start with the interface. The goal is to have a clear idea of the structure of the instrument at the outset, and knowingwhat variables are needed as you implement the design. After you have sketched out the basic functionality of your device, you are ready to design your UI in Max. There are two parts to designing the UI. The first is to create UI objects in Max and configure them to respond properly to user input. The second is to send all of the parameters that the user can change from the UI to Csound as detailed in figure 2.

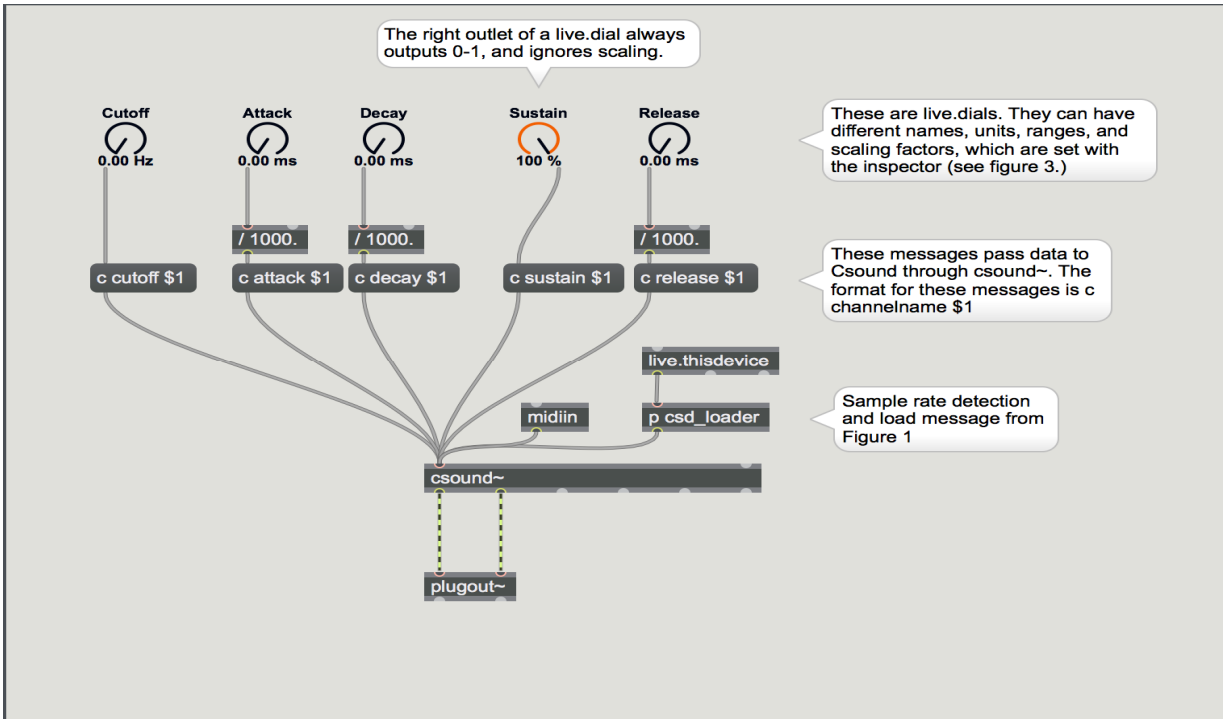

Figure 2 Sending data to Csound with channels

Designing the UI itself is a bit more complicated; but there are several tools that will give you more options over the way your interface looks and functions. Using the extensive set of *live* externals is a fine starting point, but their default capabilities are limited. The most useful tool for designing a UI in Max is the *inspector*, which can be opened by pressing *cmd+i* on a Mac, or *ctrl+i* on Windows. It allows you to change the colors of most objects. In addition, you can name objects, set their fonts, and manipulate the data they output and display. This is useful in several ways: it lets you create your own distinct look; it aids in consistency across different objects (allowing, for example, Live and non-Live Max objects to have the same color scheme); it allows you to change the text displayed by an object; and it allows you to scale the response of the UI objects to be logarithmic, linear, or exponential, which is very useful when designing controls that affect a frequency parameter in some way. For example, when creating a knob to control the cutoff-frequency of a lowpass filter, there is typically a desire to have very fine control over the range between 100Hz - 1000Hz, and less control over the range of 10000Hz and above. By setting the exponent parameter to 3, it is possible to make this range a third of the circumference of the knob, as opposed to less than a 20th.

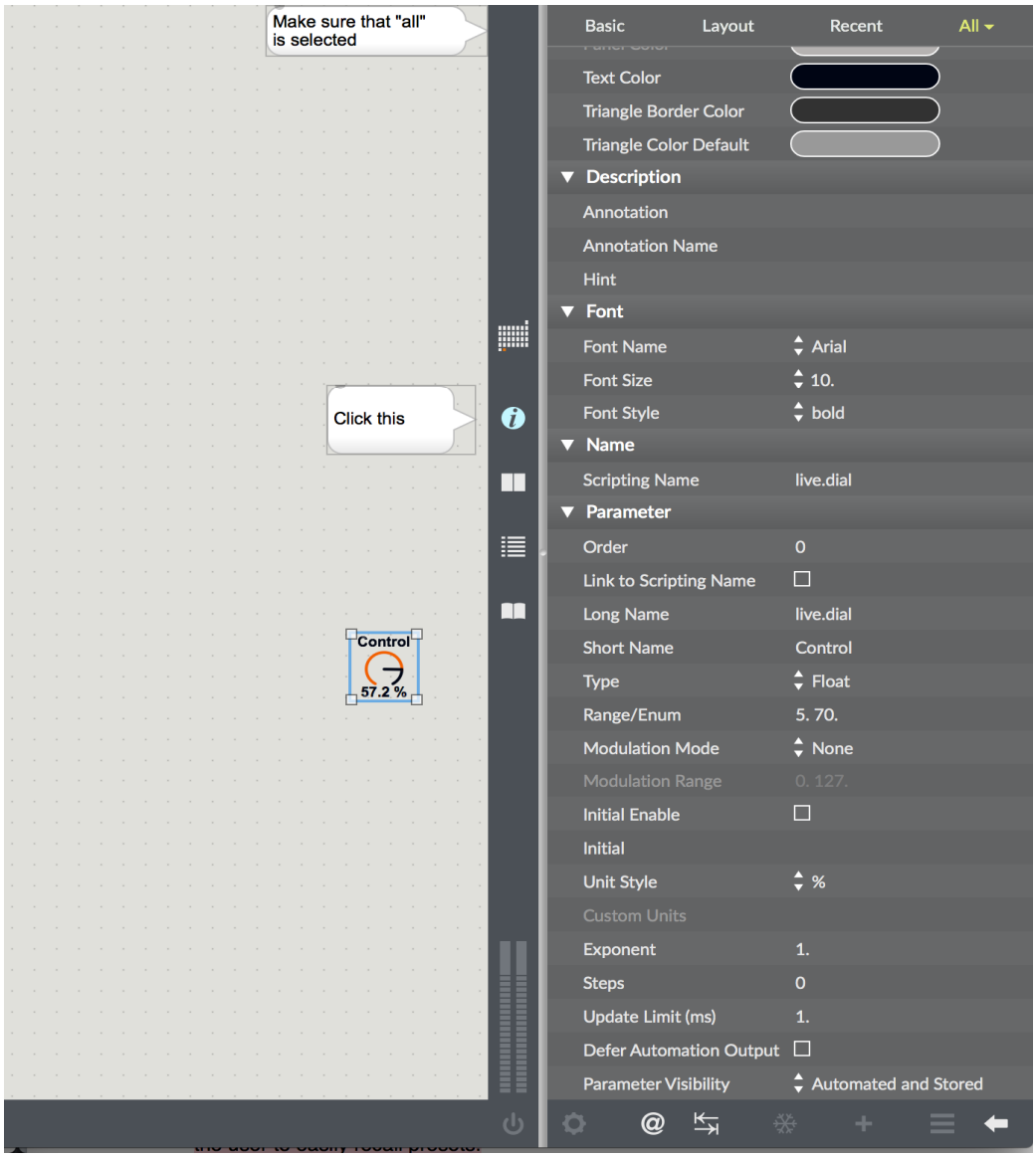

Figure 3a A view of the inspector, and the parameters it allows the programmer to edit

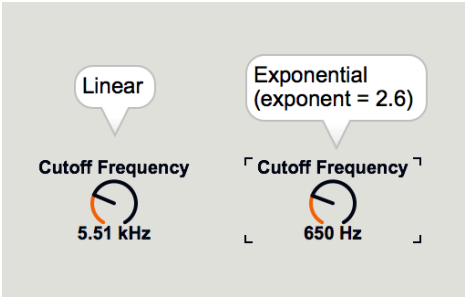

Figure 3b Two different live dials – one linear, one exponential

3 Automation

Automation allows a user to control a performance in real-time in a simple, repeatable way. Since automation allows greater precision and expressiveness in a performance, it is a highly desirable feature that enhances the usability of any device. Automating the parameters of Max for Live objects from within Ableton is the same as automating any other plugin's parameters. To allow a parameter to be automated, the *Parameter Visibility* setting in the inspector must be set to *Automated and Stored.*

If you look at the inspector for an object that is capable of editing a parameter, you will most likely see that it has a long name and a short name. An object's long name will be shown as a parameter on that device's list of parameters on a track or clip. As you create a device, your controls will be named non-indicatively (with long names like live.dial[1], live.dial[2], etc…), and so when you look at your devices' parameters in Live it will be unclear what parameter you are automating. It can be beneficial to set the long name at the same time as the short name when creating objects, to avoid forgetting to do so later. After you have set the long name of your UI objects, a user can then select your parameters in track or clip views in Ableton, and draw or perform automation for them as they would with any other plugin parameter.

4 The Live API

The Live API, when used in Max for Live, allows a user to send data to, and receive data from, Ableton Live. This is especially useful when making devices that have features that should be tempo-synced. It is also possible to have parameters change in response to certain sections of a song being reached, and to manipulate clips from within a device. The entirety of the Live API is outside the scope of this paper, so we will focus on a simple example that best allows Csound to exploit Ableton – making a tempo-synced delay that receives time signature and tempo, and passes an amount of time, in seconds, to Csound.

When using the Live API, the live.path and live.observer Max objects will allow a user to receive information from Live objects inside the Live API. By querying a specific parameter (called a property in Max for Live) via the live.observer object (after supplying it a path to the object we want to observe), we can receive its value, updated in real time, from the left outlet of the live.observer object. We can supply a path to the right inlet of the live.observer object via the live.path object. To get tempo information for a device, we first have to query the live.set object for its tempo property. This is shown in figure 4a.

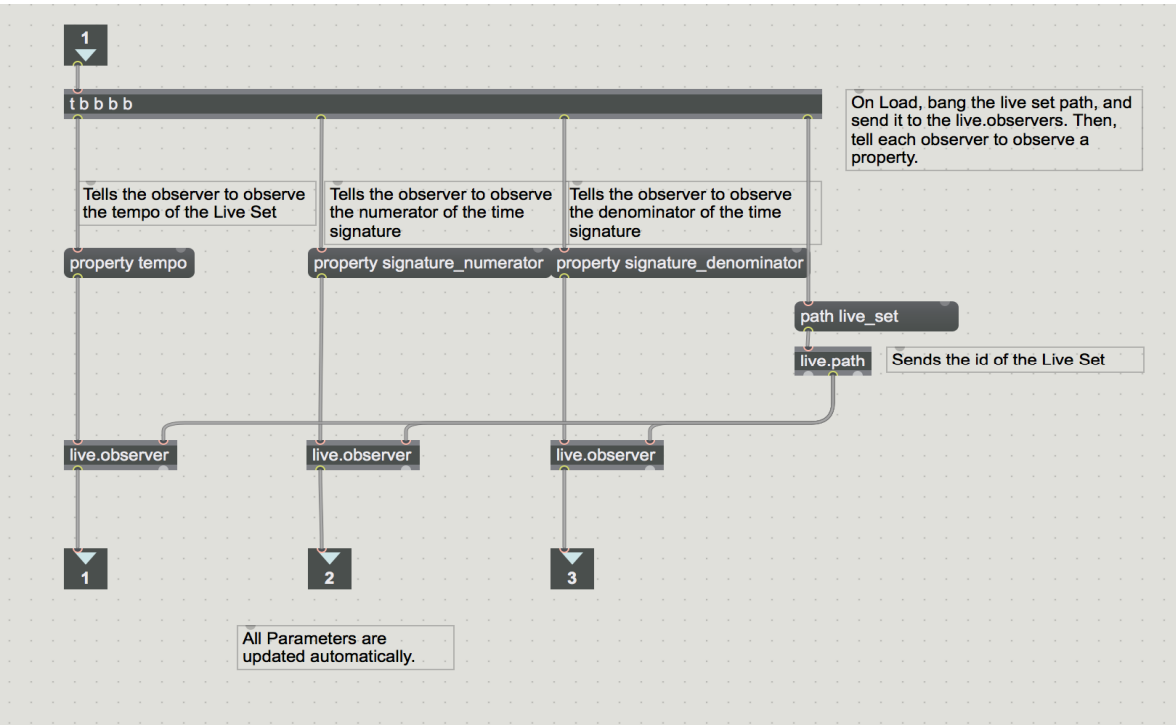

Figure 6a Setting up live.observer to receive tempo and time signature information

Using a live.tab Max object, we allow the user to choose a subdivision of the bar, which is then processed into a numerator and a denominator. We convert the BPM from the tempo property to the amount of seconds per division the user has specified in figure 4b.

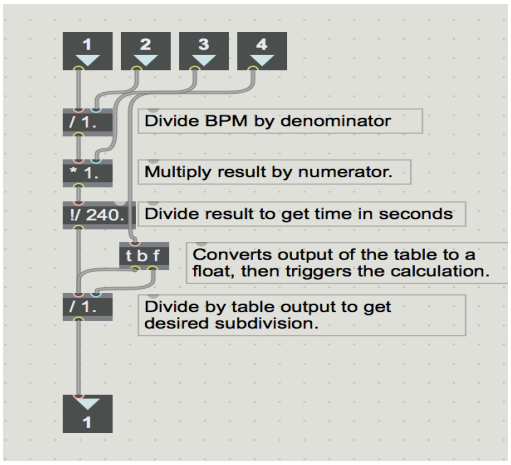

Figure 4b Converting BPM to subdivision time in seconds

By converting from a note division to a time in seconds, the delay opcode in Csound will be able to read that value and delay our audio input by the amount we want. After we have the tempo information, we still need to send Csound that data; and so we pass the value in seconds to csound~ with a *c delay $1* message.

There are many more uses for the Live API than what has been covered here. The Live Object Model and Max for Live Building Tools are great starting points and references when designing Max for Live devices.

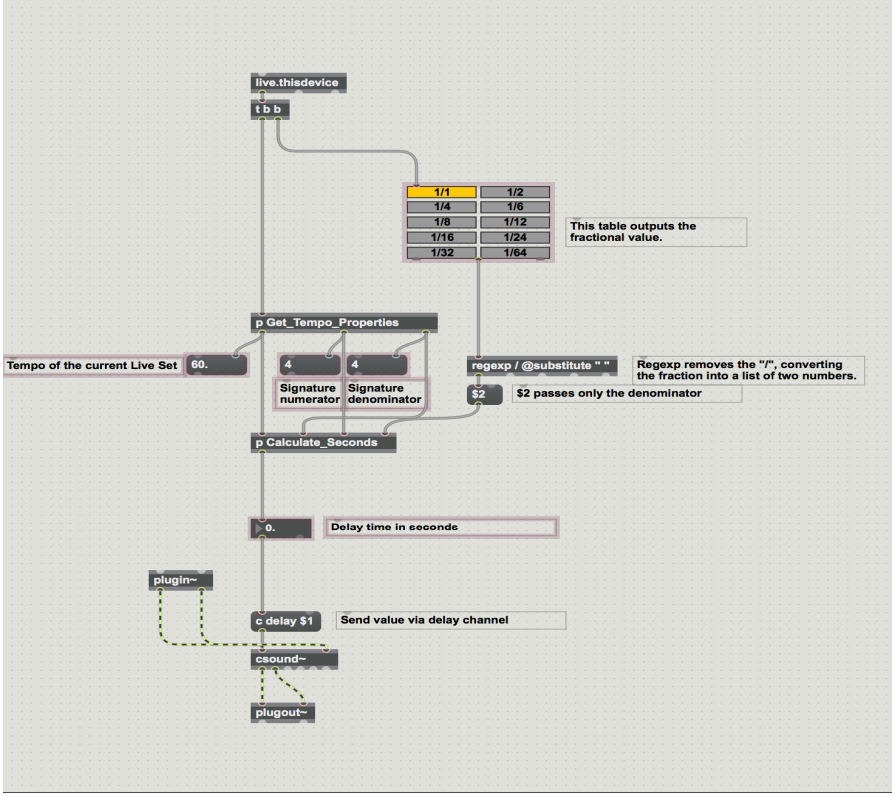

Figure 4c The whole delay patch

5 Presets

Uninitialized variables can cause certain opcodes to respond unexpectedly and so, when starting a Csound for Live device, it is important to make sure that parameters are initialized. There are two ways to initialize a device. The more efficient way is to use the init opcode in Csound, before channel declarations, to ensure that all of your variables have a value before they receive any data from the UI. While this is efficient, it means that any changes done to the UI in Max (that involve initial parameter settings) must be changed in the Csound score too, and this required step is easily forgotten. A simpler method is to use the outlet that is second to the right on csound~, which sends a bang whenever csound~ finishes compiling a .csd file. By using this bang to send an index to a preset device, as shown in figure 5, all UI objects will send their stored values through outlets that are connected. If done properly, all of your variables will be sent to the .csd file immediately after the Csound performance begins, allowing your device to receive default parameters.

There are several ways to save and load presets, but by far the simplest option is shown in figure 5, using the live.menu and preset objects. This method allows the creator of a device to make a preset storage method quickly and cleanly, and allows the user to easily recall presets.

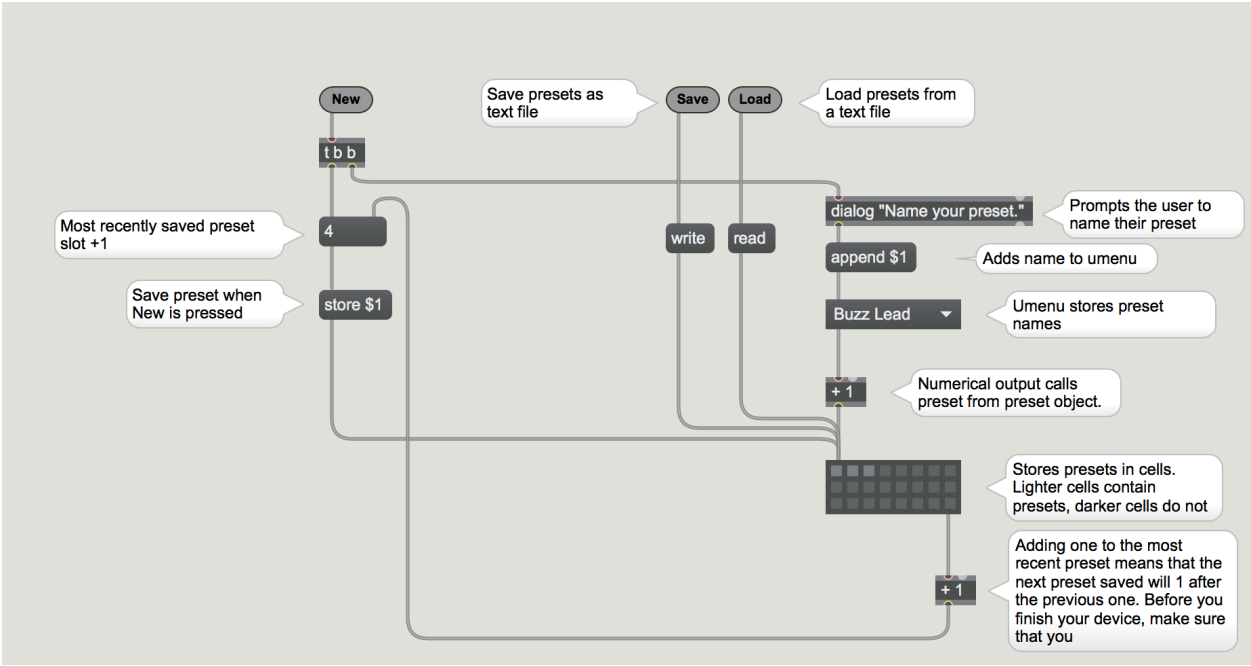

Figure 5 Saving and creating presets in Max

Conclusion

In this paper, we've examined the basic skills needed to create a Csound for Live device. We have learned how to run a unified Csound file that is compatible with multiple sample rates inside of csound~. We have learned how to send parameters from Max UI objects to Csound, via csound~, and we have learned several strategies for creating purpose-specific UI objects. We've also seen how to control parameters from within Live, store presets, and use parameters from elsewhere in Live to affect a device via the Live API. It is our hope that integrating Csound with Live will inspire new compositions, remixes, and music performances, and we hope that this tutorial has given you the insight and knowledge that you need to convert, adapt, and create new instruments that take advantage of this robust production environment.