

# Introduction to software development for behavior analysts

Carlos Rafael Fernandes Picanço  
Luiz Alexandre Barbosa de Freitas  
Hernando Borges Neves Filho  
(Editors)

**Edited by**

Carlos Rafael Fernandes Picanço  
Luiz Alexandre Barbosa de Freitas  
Hernando Borges Neves Filho

Introduction to software development for behavior analysts

Volume 2

1st Edition

© 2020 Associação Brasileira de Psicologia e Medicina Comportamental,  
Campinas, São Paulo, Brasil.

ISBN 978-65-87203-01-0



E-book for free online distribution.

**Associação Brasileira de Psicologia e Medicina Comportamental – ABPMC**  
Campinas, São Paulo, Brasil, 2020

—

**Administrative Board 2019-2020**

**Executive Board**

President: João Vicente Marçal  
Vice-president: Denise Lettieri  
First secretary: Gustavo Tozzi  
Second secretary: Elisa Sanabio Heck  
First treasurer: Flávio da Silva Borges  
Second treasurer: Cristiano Coelho

**Editora ABPMC's Editorial Board**

Angelo A. S. Sampaio  
César A. Alves da Rocha  
Diego Zilio  
Giovana Munhoz da Rocha  
Monalisa F. F. C. Leão

**About the book**

Editorial supervision: Editora ABPMC's Editorial Board  
Cover: Edmilson Pinto da Silva Junior  
Layout: Carlos Rafael Fernandes Picanço  
Support: Imagine Tecnologia Comportamental

Dados Internacionais de Catalogação na Publicação (CIP)  
(Câmara Brasileira do Livro, SP, Brasil)

Introduction to software development for behavior  
analysts [livro eletrônico] : volume 2 /  
Carlos Rafael Fernandes Picanço, Luiz Alexandre  
Barbosa de Freitas, Hernando Borges Neves Filho  
(editors) ; [tradução Carlos Rafael Fernandes  
Picanço, Luiz Alexandre Barbosa de Freitas,  
Hernando Borges Neves Filho]. -- São Paulo :  
ABPMC, 2020.  
PDF

Vários autores.

Título original: Introdução ao desenvolvimento de  
software para analistas do comportamento  
ISBN 978-65-87203-01-0

1. Análise comportamental 2. Linguagem de  
programação (Computadores) 3. Programação  
(Computadores) 4. Software - Desenvolvimento  
I. Picanço, Carlos Rafael Fernandes. II. Freitas,  
Luiz Alexandre Barbosa de. III. Neves Filho, Hernando  
Borges.

20-40694

CDD-004

Índices para catálogo sistemático:

1. Análise comportamental : Desenvolvimento de  
software : Ciência da computação 004

Cibele Maria Dias - Bibliotecária - CRB-8/9427

## **Authors**

### **Brent A. Kaplan**

Dr. Brent Kaplan received his PhD from the Applied Behavioral Economics Laboratory at the University of Kansas and completed his postdoctoral training at the Fralin Biomedical Research Institute at VTC. He currently works with Dr. Mikhail Koffarnus as a Research Assistant Professor in the Department of Family and Community Medicine at the University of Kentucky. His research interests focus on applying behavioral economic concepts to understand drug abuse and drug valuation. His other research interests include novel applications of behavioral economics and integrating contemporary technology into data analysis and dissemination.

### **Carlos Rafael Fernandes Picanço**

Dr. Rafael Picanço received his doctorate in Psychology from the graduate Program of Behavior Theory and Research at the Federal University of Pará. In the Experimental Analysis of Behavior, he conducted research on discriminative processes in the context of individualized teaching with capuchin monkeys (*Sapajus spp*) and typically developing adult humans. His main current research interest is the intersection between Computer Vision and Applied Behavior Analysis.

### **Christopher E. Bullock**

Christopher, BCBA-D, received his Ph.D. in Psychology from the University of Florida and postdoctoral training at the Johns Hopkins University School of Medicine's Kennedy Krieger Institute. His research has focused on making use of technological innovation and the application of findings from basic laboratory studies to enhance the effectiveness of behavioral interventions. In particular, his research has examined the principles governing the effectiveness of conditioned reinforcement, variables that influence choice responding, and extending the concepts of behavior economics to reinforcer assessment.

### **Hernando Borges Neves Filho**

Dr. Hernando Neves Filho is bachelor in Psychology and Psychologist (Federal University of Pará, UFPA), master in Behavior Theory and Research (UFPA) and Doctor in Experimental Psychology (University of São Paulo, USP). Was an invited researcher at the University of Auckland (New Zealand), and postdoctoral fellow at the Pontifícia Universidade Católica de Goiás and at UFPA. Currently work as senior researcher at Imagine Behavioral Technology.

**Jodie A. Waits**

Jodie Waits is a doctoral student at the Louisiana State University's School Psychology Department under Dr. Shawn Gilroy. She received her Bachelor of Science degree in psychology from the University of New Orleans. She is currently interested in developing communication interventions for bilingual children with autism.

**Julia Zanetti Rocca**

Professor at the Federal University of Mato Grosso. Master's degree in Philosophy and a Ph.D. in Psychology, both from the Federal University of São Carlos (UFSCar). Works in the area of educational psychology, with emphasis on learning to read and write processes. Has experience in computerized teaching in the program "Aprendendo a Ler e Escrever em Pequenos Passos [Learning to Read and Write in Small Steps]" (ALEPP) from the National Institute of Science and Technology on Behavior, Cognition and Teaching (INCT - ECCE) and was a consultant in the Computer Innovation Laboratory (LINCE - UFSCar) for the construction of teaching programs.

**Luiz Alexandre Barbosa de Freitas**

Bachelor in Psychology (Federal University of São João del Rei), Master in Behavior Analysis (State University of Londrina). Currently, is a doctoral student in Theory and Research of Behavior at Federal University of Pará, with international internship at Florida Tech and Texas Christian University. Since 2011, is a professor at the Federal University of Mato Grosso's Department of Psychology. Teaches Behavior Analysis courses since 2009. Has clinical and research experience with intervention for people with Autistic Spectrum Disorder. As an amateur programmer, writes his own scripts for research (far from elegant, but functional) in Python. Is an enthusiast of programming as a complimentary tool for any and all professions.

**Ricardo Fernandes Campos Junior**

Biology bachelor (Federal University of Mato Grosso) and Master in Genetics (University of São Paulo), worked with Approximate Bayesian Computation and evolutionary processes analysis using Coalescent Theory. Has 7+ years of experience in R programming and recently has been working with data mining and artificial intelligence with a technical training scholarship from National Institute of Science and Technology on Behavior, Cognition, and Teaching.

**Shawn P. Gilroy**

Gilroy, BCBA-D NCSP LBA, received his Ph.D. in School Psychology from Temple University. Certified as both an educational psychologist and behaviour analyst, Shawn completed his pre-doctoral training in Behavioural Pediatrics at the University of Nebraska Medical Center's Munroe-Meyer Institute and his postdoctoral training at the Johns Hopkins University School of Medicine's Kennedy Krieger Institute. His research

has focused on the development and evaluation of evidence-based protocols for the use of technology with exceptional populations. Additional projects have included behavioural decision-making, applied behaviour economics, and modelling of human decision-making.

### **Théo P. Robinson**

Théo Robinson, BCBA, is a behavioral researcher and computer programming hobbyist from Melbourne, Florida. He is currently enrolled as a student at the Florida Institute of Technology, where he is working to earn a doctoral degree in Applied Behavior Analysis. His interests include the study of relapse phenomena and finding ways to improve the quality and efficacy of translational research methodologies for studying human behavior.

### **Wiviny Araújo Lima**

Born in Anápolis/GO, Brazil, he graduated in Psychology at the Pontifical Catholic University of Goiás (PUC-GO). He was a scientific initiation fellow for three years, under Professor Lorismario Ernesto Simonassi, during which he developed various softwares for use by the Laboratory of Experimental Analysis of Behavior. He received the Magna Cum Laude Diploma of Academic Dignity for his undergraduate performance. In 2019, he joined as a master student in the Graduate Program in Psychology at PUC-GO as a CAPES fellow. Develops research on: Superstition, Remembering, Stroop Effect, Discrepant Rules and Culture.

## **Preface**

This second volume of the "Introduction to Software Development for Behavior Analysts" series, as the first one, aims to provide accessible tools for students, professionals, teachers and researchers interested in developing behavioral technology with the helping hand of computers. This book is designed to get you started on specific fields in contemporary Behavior Analysis that have been making efforts towards this end. Remember: programming is not just for professional programmers, with a specific formal instruction or working to large technology companies. This is not the only possible scenario. Of course, this does not mean that you should not be pursuing specific training for problems you aim to solve. We hope this book will reduce entry barriers for the use of computing solutions in our community, and will also encourage the leadership and autonomy in more behavior analysts interested in solving behavioral problems with the help of computing solutions.

C. R. F. P.

L. A. B. F.

H. B. N. F.

July, 2019.

## Foreword

Passing on information, skills, and passion for learning to the next generation of scientists, researchers, practitioners, clinicians, and students is the greatest pleasure of serving as a teacher. I have benefited greatly from mentorship and training from some of the finest teachers in my field, and it is extremely gratifying to have had the opportunity to read the chapters in this book and to provide some initial thoughts for the interested reader.

Textbooks can be so useful because they generally cull authors who have some sort of expertise in specific topics, and those related topics are discussed in one place. This makes it eminently easier for students – or anybody else – to read summaries and descriptions of scientific, clinical, and/or entrepreneurial endeavors in a cohesive and efficient way. In *Introduction to software development to behavior analysts – volume 2*, readers will experience an introduction to important questions and solutions about the use of technology in Behavior Analysis. Specifically, this book focuses on the development of software, and its ultimate application, for advancing clinical service, training, and research in Behavior Analysis.

Behavior Analysis has already come a long way with the introduction of technology for clinical service, training, and research. Basic researchers used the cumulative record as a way to easily and conveniently show the changes in behavior, and the changes in environmental conditions, in operant laboratories. Producing the cumulative record required a great deal of time and preparation unrelated to the experiment *per se*. Today, basic researchers incorporate computer technology to design and execute complex experiments in far more efficient manner. Applied researchers have used – and some still use – ‘paper and pencil’ methods to collect data. However, there are numerous technology-driven options to replace the ‘paper and pencil’ methods for more efficient data collection, treatment integrity and



reliability calculations, and graphing. When integrated into research and practice, these technological advancements open the door to greater efficiency at a minimum, and creative innovation as the ultimate goal.

In this book, readers will learn about a broad array of technological advancements in software development relevant to Behavior Analysis. Chapter 1 shows the history and relevance of technological advancements in the field. Beyond the examples of applications available for research and clinical use, authors discuss the importance of developing free and open-source software, and the need to train behavior analysts for software developing. Chapter 2 introduces readers to a specific software platform that is ideal for conducted basic and translational research via the Internet – a virtual laboratory. Chapter 3 discusses a novel way to register data using R programming language. Chapter 4 introduces a discussion on software development and research authorship: should a developer be co-author in an article he developed software for?

Michael Kelley

June, 2019.

## Contents

<b>Authors</b>	i
<b>Preface</b>	iv
<b>Foreword</b>	v
<b>Chapter 1</b>	
Current use and development of FOSS in Behavior Analysis: Modern Behavioral Engineering.	1
<i>Shawn P. Gilroy, Brent A. Kaplan, Christopher E. Bullock and Jodie A. Waits.</i>	
<b>Chapter 2</b>	
A step-by-step guide to develop experiments with Axure© RP.	21
<i>Théo P. Robinson.</i>	
<b>Chapter 3</b>	
Developing an application to register continuous responses with the shiny package in R environment	71
<i>Ricardo Fernandes Campos Júnior and Júlia Zanetti Rocca.</i>	
<b>Chapter 4</b>	
Is the programmer an author? Developing software for research	94
<i>Wivinnny Araújo Lima and Carlos Rafael Fernandes Picanço.</i>	

## Chapter 1

### Current use and development of FOSS in Behavior Analysis: Modern Behavioral Engineering<sup>1</sup>

Shawn P. Gilroy<sup>2</sup>

*Louisiana State University, LA, USA*

Brent A. Kaplan

*Virginia Tech Carilion Research Institute*

*Virginia Polytechnic Institute, VA, USA*

Christopher E. Bullock

*Francis Marion University, SC USA*

Jodie A. Waits

*Louisiana State University, LA, USA*

### Abstract

Technological development and engineering skills have long held a place in Experimental Behavior Analysis. This chapter presents a brief history of do-it-yourself culture in Behavior Analysis and suggests that this culture moved from the manufacture of electromechanical devices to computer programming. Additionally, it provides a panoramic view of recent advances in the field obtained with the help of computational solutions including contributions to research fields such as substance dependence, the provision of services in the field of atypical development and behavioral economics. It discusses the role of the development of free and open computer programs in the scientific field, suggesting that this model contributes to the teaching of engineering skills in academia and has the potential to increase the value of the work of behavior analysts in the market.

---

<sup>1</sup> Editors note: A pre-print version of this chapter received the editor's authorization for early distribution.

<sup>2</sup> Please send correspondence to Shawn Patrick Gilroy either to [sgilroy1@lsu.edu](mailto:sgilroy1@lsu.edu) or [shawnpgilroy@gmail.com](mailto:shawnpgilroy@gmail.com). You can find Shawn on GitHub at <https://github.com/miyamot0>.

### **Maker culture in Behavior Analysis: a brief history**

Technological development and engineering skills have long held a place in the Experimental Analysis of Behavior. In years prior to Burrhus Frederic Skinner (1904-1990) and his seminal works, behavioral psychologists regularly crafted the tools and technology necessary to perform controlled experiments. For example, scientists such as John Broadus Watson (1878-1958) frequently highlighted the measurement apparatuses (predominantly used with nonhuman animals at the time) used to investigate behavioral phenomena (Watson, 1916). Even 100 years ago, developing highly-specialized apparatuses was necessary when responses were difficult to perceive (e.g., by visual inspection), hard to measure reliably (e.g., due to the high rate of occurrence), or spanned great lengths of time (i.e., whole days, weeks). Skinner (1956) provides a thoughtful summary of the many tools created to support his early operant experiments.

In reviewing Watson and Skinner's discussions of apparatuses, it is made clear that the technology for measuring and recording behavior was not something that was commercially available at the time. In recollections provided in Catania (2002), the work of students in Skinner's Pidgeon Lab (1958-1962) regularly involved designing and constructing equipment using the mechanical components available at the time. For example, Catania (2002) describes the use of stepper motors and punched tape to improvise the repeating functionality (i.e., looping) necessary for time-controlled events (e.g., interval recording, reinforcer delivery). Dinsmoor (1990) also discussed similar experiences in this era, highlighting how many behavioral scientists at the time were performing the metal- and woodworking necessary for their experiments on their own. In the eras recalled by Catania and Dinsmoor (i.e., 1950-1960), behavioral psychologists were regularly designing and constructing the technology in their experiments using some combination of stepper motors, relays, switches, and timers (Escobar, 2014). As thoroughly and thoughtfully detailed in Escobar (2014), the days of relays, switches, and improvised apparatuses was eventually replaced by affordable computer

equipment that could be programmed using some form of state notation (e.g., Med State Notation™) or programming language (e.g., BASIC). In this period of increased computer usage, commercial products for running operant experiments (e.g., bought from Med Associates Inc) became increasingly available and affordable for researchers with the requisite funding. At this point, the focus became more on computer programming rather than developing apparatuses from individual components.

### **Recent behavior analytic research using technology**

Like the initial work done by Skinner using relays and timers, much of the early work using computer programming and computers focused on automating various aspects of experimental work (Chayer-Farrell, Freedman, & Computers, 1987; Emmett-Oglesby, Spencer, & Arnoult, 1982; Kaplan, 1985). For example, programmed instructions could be written to automate various aspects of research, such as generating variable schedules of reinforcement (Hantula, 1991). However, research using computers (e.g., smartphones, tablets) has since developed past simple schedules of reinforcement into a variety of systems designed to produce socially-significant behavior changes (Marsch, Lord, & Dallery, 2014).

### **Technology interventions against substance dependence**

Recent research in Behavior Analysis has leveraged the capabilities afforded by modern technology (i.e., cell phones, personal computers) and the internet to develop and evaluate novel forms of intervention. For instance, both Reynolds, Dallery, Shroff, Patak, Leraas, and Silverman (2008) and Dallery, Raiff, and Grabinski (2013) used personal computers (PCs) with an equipped web camera and carbon monoxide (CO) monitor to support a remote smoking abstinence program. That is, Reynolds et al. (2008) leveraged the capabilities of the internet to develop a remote method of contingency management whereby reinforcers (i.e., money) were delivered contingent on providing an acceptable CO sample (i.e., lower CO samples were suggestive of abstinence). However, this research was limited in the technological capabilities because study personnel had to manually email participants the results of the

CO samples and deliver the reinforcers (i.e., cash) at the end of each week.

In a more recent contingency management study by Koffarnus, Bickel, and Kablinger (2018), treatment-seeking, alcohol-dependent participants were provided internet-capable smartphones and breathalyzers and received reinforcers via a reloadable debit card. This study used a fully remotely-delivered contingency management protocol resulting in very high levels of adherence (over 95% of submitted breathalyzer samples) and abstinence (over 85% of participants) among those in the contingent group. Similarly, other reinforcement-based approaches using computer-aided forms of contingency management have been developed for cocaine- and opioid-dependent individuals (Bickel, Marsch, Buchhalter, & Badger, 2008). As an extension to Internet-based contingency management, Raiff, Fortugno, Scherlis, and Rapoza (2018) have also developed and evaluated a version of a smoking cessation program using a game-based approach. Rather than delivering monetary forms of reinforcement, this approach utilized virtual rewards in the form of in-game items and social support.

### **Technology and behavior analytic service delivery**

Aside from treatments for substance dependence and abuse, recent research has evaluated how video streaming software can be used to support remote behavioral consultation. Through video consultation, trained and credentialed behavior analysts can provide services to families, educators, and other professionals in areas where such services are not locally available (Tomlinson, Gore, & McGill, 2018). Recent research on this novel mode of service delivery has found that this approach yielded similar benefits with regard to the traditional method of in-person service delivery (Lindgren et al., 2016; Sutherland, Trembath, & Roberts, 2018).

A systematic review by Sutherland et al. (2018) found that remote service delivery has been successfully evaluated for a range of services beneficial to individuals with disabilities, such as behavioral analytic assessments and early intervention. Aside from similar effi-

cacy with regard to traditional modes of delivery, others have found that this novel approach was often more economical and sustainable (Hay-Hansson & Eldevik, 2013; Wacker et al., 2013) and could be made available at a lower cost to families (Tomlinson et al., 2018). Further, Lindgren et al. (2016) provide a compelling case that even highly-specialized procedures (i.e., experimental functional analyses) can be implemented remotely with families and the requisite technology.

### **High-tech treatments for individuals with disabilities**

Regarding direct interventions with service users (e.g., autism, intellectual disabilities, academic difficulties), several forms of intervention using modern technology have been developed. Under the umbrella of Computer-Assisted Instruction, the Headsprout<sup>TM</sup> reading program is an online reading program based on stimulus equivalence and verbal behavior. Using PCs and the internet, the Headsprout<sup>TM</sup> reading program has been found to be effective for children with reading difficulties (Cullen, Alber-Morgan, Schnell, & Wheaton, 2014) as well as for children diagnosed with autism (Plavnick et al., 2014; Whitcomb, Bass, & Luiselli, 2011). As highlighted in Cullen et al. (2014), programs such as Headsprout<sup>TM</sup> serve to support the use and dissemination of instructional curricula based on sound behavioral science.

In the area of social and communication impairments, behavior analysts have leveraged mobile technology (e.g., tablets, iPads) and commercially-available mobile applications (i.e., apps) to support individuals with effective speech. As found in a recent review by Gilroy, McCleery, and Leader (2017), over 50 peer-reviewed studies have used Speech-Generating Devices (SGDs) as a replacement for deficient vocal repertoires. Using mobile technology, several apps have been developed to provide functionality previously available only on dedicated devices (e.g., Tobii Dynavox<sup>TM</sup>) at a high cost (more than US \$ 8,000). Apps such as the Picture Exchange Communication System (PECS) Phase III app (Pyramid Educational Consultants, 2018) have been found to be effective supplements to function-based

communication training (Alzrayer, Banda, & Koul, 2014; Ganz, Hong & Goodwyn, 2013). Further, positive effects of these devices and intervention have also been demonstrated in larger, randomized control trials for children diagnosed with autism (An et al., 2017; Gilroy, McCleery, & Leader, 2018).

### **Behavior Analysis and Behavioral Economics**

Behavioral Economics, within the broader Behavior Analytic domain, has also been leveraging the capabilities of modern technology. Originally based on the framework of a virtual (i.e., simulated), experimental supermarket (Epstein, Dearing, Roba, & Finkelstein, 2010; Epstein et al., 2012), the Experimental Tobacco Marketplace (ETM) is a virtual storefront where participants may purchase, either hypothetically or experientially, from a range of tobacco products (Bickel et al., 2018; Heckman et al., 2017; Pope et al., 2018; Quisenberry, Koffarnus, Epstein, & Bickel, 2017; Quisenberry, Koffarnus, Hatz, Epstein, & Bickel, 2015). Indeed, the ETM framework serves as an analogue to the complex, real-world marketplace where a variety of research questions can be evaluated such as the effects of taxation/subsidization, flavor/product restrictions, and different product concentrations (Pope et al., 2018). When used in research, participants are provided a budget that approximates their typical tobacco product expenditures and their consumption of these goods is assessed as the price these products increases. Importantly, the ETM has been continuously refined to capitalize on more flexible frameworks.

In the initial forms of the ETM developed using WordPress<sup>TM</sup> and OpenCart<sup>TM</sup>, this approach presented with several limitations. For example, this approach required substantial time and effort from the research team and was not well-suited to large-scale data collection (e.g., use on Amazon's Mechanical Turk<sup>TM</sup>) or automation. To address these limitations, operant behavioral economists have since programmed alternatives to support more flexible and modular usage. For example, these teams have used Javascript (e.g., Qualtrics Research Platform<sup>TM</sup>) and Python (i.e., local Flask server) to develop novel meth-



ods of measuring individual preferences and consumption.

### **Behavior analysts and FOSS technology**

While many areas of behavior analytic research have effectively capitalized on the availability of modern technology, several behavior analysts have moved beyond *using* technology and instead towards *developing* their own. That is, rather than relying on commercially-available tools and devices, these behavior analysts have created specific technology to enhance their research and practice. In many areas of behavior analytic research and practice, such developments have been necessary because many commercially-available products may not provide functionality and features desired by behavior analysts.

The development and use of software that is both free *and* open is important for collaborative science, especially Behavior Analysis. For example, the ability to publicly inspect and re-use open computer software and scripts supports transparent and accessible sciences—regardless of individual specialty or focus. As noted in the third version of the General Public License (<https://www.gnu.org/licenses/gpl-3.0.en.html>), “When we speak of free software, we are referring to freedom, not price.” The term *free* here refers to the right to openly inspect software, and as useful, extend software to suit individual needs. This freedom is particularly salient for professionals working with exceptional populations, where nearly all elements of applied work require high levels of individualization. The availability of open software allows those with the requisite skills to truly individualize technology for individual users and particular populations and to do so in ways that support transparency and replicability.

**Electronic data collection tools**

Behavior analytic research has required specialized tools to support behavior analytic practices. For example, specialized software has been developed to support the measurement of behavior in assessments and interventions for individuals with developmental disabilities (Bullock, Fisher, & Hagopian, 2017; Gilroy, 2017). These applications have been particularly useful in alleviating the potentially large time demands placed on researchers and practitioners when collecting observation-based behavioral data.

Many of the earlier approaches for automating data collection for multiple topographies of problem behavior were either not commercially-available, prohibitively expensive, or were unacceptably invasive. As a result, most behavior analytic practices have involved methods in which an observer is equipped with a timekeeping device, and paper and pencil are used to record when and how often behavior occurred. The development of computerized, behavior analytic data collection software has provided a means to automate many aspects of data collection, analysis, and storage.

In the BDataPro data collection program (Bullock et al., 2017), this software can be used to systematically collect the information necessary to perform experimental analyses (i.e., functional analysis) as well as evaluate the effectiveness of on-going treatments. This software makes use of keyboard key presses as a means of recording the time of occurrence and frequency and duration of target behaviors. Automated data analyses occur following each session and include the response rate, latency, percent of intervals, and various other measures (that allow the inference of inter-observer reliability, for example). Figure 1 illustrates the interface of BDataPro.

Client: <b>Client Test</b>			Assessment: <b>Functional</b>			Session Time Limit: <b>600</b>		
Session Number: <b>1</b>			Condition: <b>Demand</b>			Session + Pause Time Limit: <b>N/A</b>		
Data Type: <b>Primary</b>								

Frequency Keys			Duration Keys					Permanent Keys	
Description	Key	Count	Description	Key	Bouts	Total time	Current Time	Description	Key
SIB	1	0	SR	r	0	0	0	Start Session	Tab
AGG	2	0						Toggle Session (ST) and Pause (PT) Time	Control-X
								Delete Last Entry	Backspace
								Replace Last Entry	Enter
								Press the Escape Key to End Session	Escape
								Total Time:	
								Session Time:	
								Pause Time:	

--	--	--	--	--	--	--	--	--	--	--	--

Figure 1. BDataPro Data Collection Software.

This program was written in Visual Basic 6.0, submitted to peer-review, and released under a free software license—the General Public License, Version 2.0 (GPLv2). It has been used and refined through extensive clinical use at many premier behavior analytic programs in the United States. While BDataPro was originally created for the Windows operating system, Gilroy (2017) designed a cross-platform alternative (DataTracker) that could be compiled for the Windows, macOS, and Linux operating systems. DataTracker was written in the C++ language and the GUI was constructed using the Qt Framework. The DataTracker software is currently in active development and released under a free software license—the General Public License, Version 3.0 (GPLv3).

### Speech generating devices

From beyond the data collection, behavior analysts have developed mobile applications designed to aid/supplement function-based treatments. For example, Gilroy, McCleery, et al. (2018) developed an open-source app for use in communication interventions for children diagnosed with autism. The FastTalker app was designed to function similar to communication interventions using exchanges of picture cards. The interface used in FastTalker is shown in Figure 2.

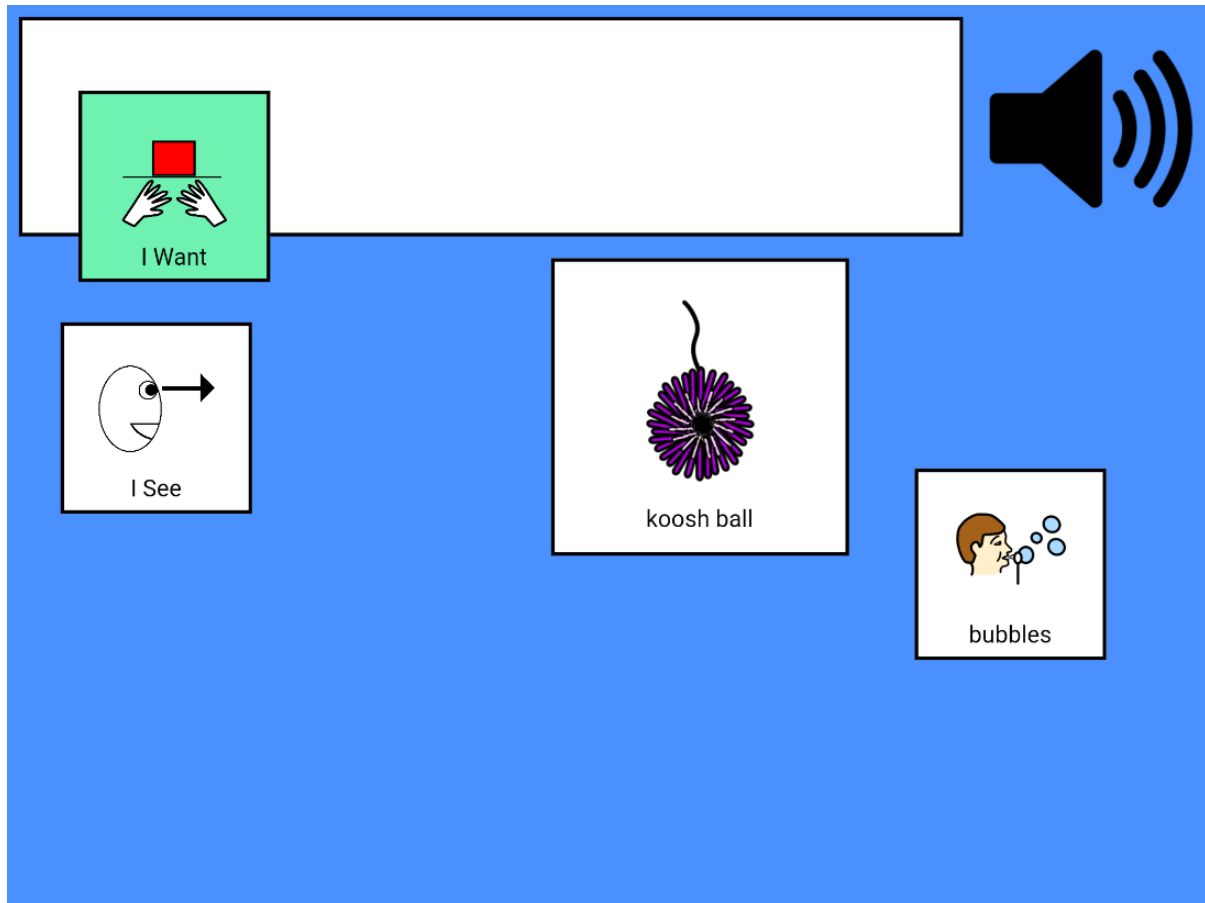
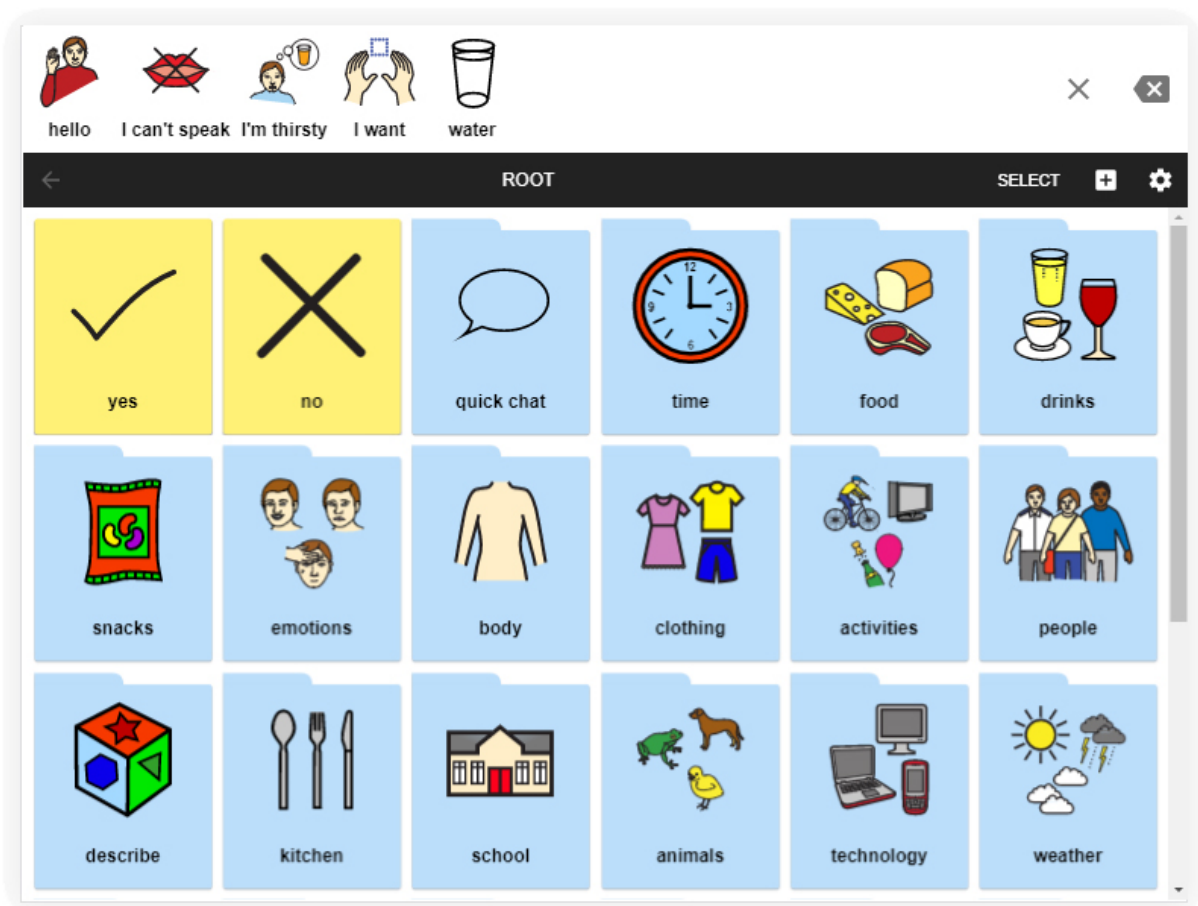


Figure 2. FastTalker Application.

Using a format consistent with earlier behavior analytic interventions, FastTalker was designed to enable a comparison of methods using high-tech (i.e., tablet) and low-tech (i.e., picture-exchange) communication devices. Specifically, FastTalker was used in a randomized control trial which found that high-tech approaches, such as FastTalker, provided benefits that were consistent with those from low-tech approaches (Gilroy, McCleery, et al., 2018). FastTalker was constructed using the C# language and the Xamarin.Forms framework to support Android and iOS platforms, with current efforts dedicated to porting FastTalker to Google's cross-platform Flutter framework (Dart). It has been licensed under an open source license—the permissive MIT license.

While not specific to function-based intervention and autism spectrum disorder, the Cboard application (CIREHA, 2018) is another open-source communication application designed to provide functionality similar to commercially-available applications, such as Proloquo2Go™ (AssistiveWare, 2018), but free-of-charge. The interface provided by Cboard is

displayed in Figure 3.



*Figure 3.* Cboard Application. The screenshot was retrieved from the GitHub repository associated with Cboard, at <https://github.com/cboard-org/cboard>.

Cboard is a browser-based application, written in JavaScript using Node.js and React.js, that functions across any device (e.g., tablet, phone, computer) with a modern internet browser. The application is currently in active development and released under a free software license—the GPLv3.

### **Operant Behavioral Economics**

Aside from assessment and intervention using technology, behavior analysts have also developed technology specific to statistical analyses. Statistical analyses, especially in the Experimental Analysis of Behavior, are increasingly observed in behavior analytic venues (Young, 2018). As noted in Gilroy, Franck, and Hantula (2017), few commercial statistical packages offer models and metrics specific to behavior analysts. That is, few statistical tools

have historically been available to perform analyses of operant demand and delay discounting (or probability discounting).

Studies of operant demand have been used broadly across disciplines, including substance dependence (Kaplan, Foster, et al., 2018), consumer purchasing behavior (Foxall, Olivera-Castro, Schrezenmaier & James, 2007; Foxall, Wells, Chang & Oliveira-Castro, 2010; Kaplan, Gelino & Reed, 2018), and assessments and treatments for individuals with disabilities (Gilroy, Kaplan & Leader, 2018). While demand analyses have been done using commercial software in the past, the Demand Curve Analyzer (DCA; Gilroy et al., 2018) was recently developed a completely free and open-source alternative for completing these analyses. The DCA was written in the C++ language and the graphical user interface was constructed using the Qt Framework. The program functions identically across many popular operating systems (e.g., macOS, Windows) with automated updates for users. Because the DCA was developed as a tool specialized for demand curve analyses, specific options are available for screening data, analyzing data, and generating a range of different outputs for use in analyses relevant to Behavior Analysis. The DCA and all its assets were released under a free software license (GPLv3).

Similarly, for researchers who primarily use syntax-based statistical tools such as the R Statistical Software (R Core Team, 2017), behavior analysts have developed an R package (i.e., a collection of specialized functions) for conducting demand curve analyses. Like the DCA, the R package, *beezdemand* (Kaplan, 2018), is specifically suited to conduct the common steps involved in demand curve analyses, which include data pre-processing, data screening, curve fitting, comparisons, and visualizations. Importantly, the utility of *beezdemand* is the ability to integrate seamlessly within a larger code-based workflow, meaning an entire analysis can be conducted within one program. The *beezdemand* R package was released under a free software license (GPLv3).

Aside from studies of operant demand, statistical tools have also been developed for

use in studies of delay (or probability) discounting. The Discounting Model Selector (DMS; Gilroy et al., 2017) was designed for behavior analysts to support the fitting and comparison of multiple possible delay discounting models. This software was designed for use by behavior analysts, who may or may not have the statistical training to accurately fit and compare competing models of discounting. Like the DCA, the DMS was written using the C++ language and the Qt framework. The DMS was made with a GUI that functions identically across many popular operating systems (e.g., macOS, Windows) and offers automated updates. The DMS, and all its associated assets, were released under a free software license—the GPLv3.

### **Future directions and next steps**

At the present time, many areas of Behavior Analysis continue to adapt to the increasingly powerful capabilities of technology. Alongside increasingly affordable hardware, a range of free and open-source development frameworks continue to mature. As a result of these increasingly accessible frameworks, even the novice developer can create software to enhance their research and practice. While this trend is likely to continue, the development of software for behavior analytic purposes would benefit from established standards and formal training in Computer Science.

### **Training in Computer Science and programming**

While several behavior analysts have taken the lead in developing technology, this number is relatively small at the time. Although using commercially-available products, with the requisite funding, has proven to be effective in behavior analytic research, few commercial products (i.e., software) have been developed explicitly for behavior analytic use and interpretation. Regardless of other constraints, the number of behavior analysts developing technology specific to Behavior Analysis is also constrained by the level of their training in Computer Science and modern programming languages. For example, students and researchers in Behavior Analysis are unlikely to be exposed to development using modern

languages such as Python, C, C++, C#, Go, Dart, Rust, Java, Kotlin, Objective-C, Swift and Object Pascal or version control systems, such as stand alone Git, GitHub or GitLab. As such, resources and formal training are likely to be necessary to support the development of technology consistent with modern programming conventions, languages, and platforms. Aside from benefit to the science of Behavior Analysis, training in software development would likely enhance the prospects for behavior analysts and increase their marketability. For example, behavior analysts trained in both the design of interventions and the technology to support them may have opportunities for work that extend much further than work with developmental disabilities and substance abuse (e.g., software development, design).

### **Free Software, Open Source and transparent practices in research**

In addition to the development of technology for behavior analytic research, the behavior analytic community should encourage the use of free and open-source software design and licensing. Purely from a clinical and research standpoint, the availability of source code supports both direct replication in research as well as collaboration with other professionals. Further, open development supports transparent practices and enhances the ability of future researchers and clinicians to understand and even extend released works. We encourage researchers and professionals in Behavior Analysis interested in accessible science and technology to become part of this movement and to familiarize themselves with free software and open source culture.

### **References**

- Alzrayer, N., Banda, D. R., & Koul, R. K. (2014). Use of iPad/iPods with Individuals with Autism and other Developmental Disabilities: A Meta-analysis of Communication Interventions. *Review Journal of Autism and Developmental Disorders*, 1(3), 179-191. <https://doi.org/10.1007/s40489-014-0018-5>
- An, S., Feng, X., Dai, Y., Bo, H., Wang, X., Li, M., . . . Wei, L. (2017). Development and evaluation of a speech-generating AAC mobile app for minimally verbal children



- with autism spectrum disorder in Mainland China. *Molecular Autism*, 8, 52.  
<https://doi.org/10.1186/s13229-017-0165-5>
- AssistiveWare. (2018). Proloquo2Go: AssistiveWare.
- Bickel, W. K., Marsch, L. A., Buchhalter, A. R., & Badger, G. J. (2008). Computerized behavior therapy for opioid-dependent outpatients: a randomized controlled trial. *Experimental and Clinical Psychopharmacology*, 16(2), 132-143.  
<https://doi.org/10.1037/1064-1297.16.2.132>
- Bickel, W. K., Pope, D. A., Kaplan, B. A., DeHart, W. B., Koffarnus, M. N., & Stein, J. S. (2018). Electronic cigarette substitution in the experimental tobacco marketplace: A review. *Preventive Medicine*, 117, 98-106.  
<https://doi.org/10.1016/j.ypmed.2018.04.026>
- Bullock, C. E., Fisher, W. W., & Hagopian, L. P. (2017). Description and Validation of a Computerized Behavioral Data Program: "BDataPro". *The Behavior Analyst*, 40(1), 275-285. <https://doi.org/10.1007/s40614-016-0079-0>
- Catania, A. C. (2002). The watershed years of 1958-1962 in the Harvard Pigeon Lab. *Journal of the Experimental Analysis of Behavior*, 77(3), 327-345.  
<https://doi.org/10.1901/jeab.2002.77-327>
- Chayer-farrell, L., & Freedman, N.L. (1987). *Behavior Research Methods, Instruments, & Computers*, 19(3), 319-326. <https://doi.org/10.3758/BF03202569>
- CIREHA. (2018). Cboard AAC Application. Retrieved from <https://www.cboard.io/>
- Cullen, J. M., Alber-Morgan, S. R., Schnell, S. T., & Wheaton, J. E. (2014). Improving Reading Skills of Students With Disabilities Using Headsprout Comprehension. *Remedial and Special Education*, 35(6), 356-365.  
<https://doi.org/10.1177/0741932514534075>
- Dallery, J., Raiff, B. R., & Grabinski, M. J. (2013). Internet-based contingency management to promote smoking cessation: A randomized controlled study. *Journal of Applied*

- Behavior Analysis*, 46(4), 750-764. <https://doi.org/10.1002/jaba.89>
- Dinsmoor, J. A. (1990). Academic roots: Columbia University, 1943–1951. *Journal of the Experimental Analysis of Behavior*, 54(2), 129-149. <https://doi.org/10.1901/jeab.1990.54-129>
- Emmett-Oglesby, M. W., Spencer, D. G., & Arnoult, D. E. (1982). A TRS-80-based system for the control of behavioral experiments. *Pharmacology Biochemistry and Behavior*, 17(3), 583-587. [https://doi.org/10.1016/0091-3057\(82\)90322-7](https://doi.org/10.1016/0091-3057(82)90322-7)
- Epstein, L. H., Dearing, K. K., Roba, L. G., & Finkelstein, E. (2010). The influence of taxes and subsidies on energy purchased in an experimental purchasing study. *Psychological Science*, 21(3), 406-414. <https://doi.org/10.1177/0956797610361446>
- Epstein, L. H., Jankowiak, N., Nederkoorn, C., Raynor, H. A., French, S. A., & Finkelstein, E. (2012). Experimental research on the relation between food price changes and food-purchasing patterns: a targeted review. *Am J Clin Nutr*, 95(4), 789-809. <https://doi.org/10.3945/ajcn.111.024380>
- Escobar, R. (2014). From relays to microcontrollers: The adoption of technology in operant research. *Revista Mexicana de Análisis de la Conducta*, 40(2), 127-153 <https://doi.org/10.5514/rmac.v40.i2.63673>
- Foxall, G. R., Olivera-Castro, J., Schrezenmaier, T., & James, V. (2007). *The Behavioral Economics of Brand Choice*: Palgrave Macmillan, London . <https://doi.org/10.1057/9780230596733>
- Foxall, G. R., Wells, V. K., Chang, S. W., & Oliveira-Castro, J. M. (2010). Substitutability and Independence: Matching Analyses of Brands and Products. *Journal of Organizational Behavior Management*, 30(2), 145-160. <https://doi.org/10.1080/01608061003756414>
- Ganz, J. B., Hong, E. R., & Goodwyn, F. D. (2013). Effectiveness of the PECS Phase III app and choice between the app and traditional PECS among preschoolers with ASD.

- Research in Autism Spectrum Disorders*, 7(8), 973-983.  
<https://doi.org/10.1016/j.rasd.2013.04.003>
- Gilroy, S. P. (2017). DataTracker: Cross-platform Electronic Data Collection Tool. Retrieved from <http://www.smallnstats.com/index.php?page=DataTracker>
- Gilroy, S. P., Franck, C. T., & Hantula, D. A. (2017). The discounting model selector: Statistical software for delay discounting applications. *Journal of the Experimental Analysis of Behavior*, 107(3), 388-401. <https://doi.org/10.1002/jeab.257>
- Gilroy, S. P., Kaplan, B. A., & Leader, G. (2018). A Systematic Review of Applied Behavioral Economics in Assessments and Treatments for Individuals with Developmental Disabilities. *Review Journal of Autism and Developmental Disorders*, 5(3), 247-259. <https://doi.org/10.1007/s40489-018-0136-6>
- Gilroy, S. P., Kaplan, B. A., Reed, D. D., Koffarnus, M. N., & Hantula, D. (2018). The Demand Curve Analyzer: Behavioral economic software for applied researchers. *Journal of the Experimental Analysis of Behavior*, 110(3), 553-568. <https://doi.org/10.1002/jeab>
- Gilroy, S. P., McCleery, J. P., & Leader, G. (2017). Systematic Review of Methods for Teaching Social and Communicative Behavior with High-Tech Augmentative and Alternative Communication Modalities. *Review Journal of Autism and Developmental Disorders*, 4(4), 307-320. <https://doi.org/10.1007/s40489-017-0115-3>
- Gilroy, S. P., McCleery, J. P., & Leader, G. (2018). A community-based randomized-controlled trial of Speech Generating Devices and the Picture Exchange Communication System for children diagnosed with autism spectrum disorder. *Autism Research*. <https://doi.org/10.1002/aur.2025>
- Hantula, D. A. (1991). A simple BASIC program to generate values for variable-interval schedules of reinforcement. *Journal of Applied Behavior Analysis*, 24(4), 799-801. <https://doi.org/10.1901/jaba.1991.24-799>

- Hay-Hansson, A. W., & Eldevik, S. (2013). Training discrete trials teaching skills using videoconference. *Research in Autism Spectrum Disorders*, 7(11), 1300-1309. <https://doi.org/10.1016/j.rasd.2013.07.022>
- Heckman, B. W., Cummings, K. M., Hirsch, A. A., Quisenberry, A. J., Borland, R., O'Connor, R. J., . . . Bickel, W. K. (2017). A Novel Method for Evaluating the Acceptability of Substitutes for Cigarettes: The Experimental Tobacco Marketplace. *Tobacco Regulatory Science*, 3(3), 266-279. <https://doi.org/10.18001/trs.3.3.3>
- Kaplan, B. A. (2018). beezdemand: R package containing functions to help aid in analyzing behavioral economic demand curve data. Retrived from <https://github.com/brentkaplan/beezdemand>
- Kaplan, B. A., Foster, R. N. S., Reed, D. D., Amlung, M., Murphy, J. G., & MacKillop, J. (2018). Understanding alcohol motivation using the alcohol purchase task: A methodological systematic review. *Drug and Alcohol Dependence*, 191, 117-140. <https://doi.org/10.1016/j.drugalcdep.2018.06.029>
- Kaplan, B. A., Gelino, B. W., & Reed, D. D. (2018). A Behavioral Economic Approach to Green Consumerism: Demand for Reusable Shopping Bags. *Behavior and Social Issues*, 27, 20-30. <https://doi.org/10.5210/bsi.v.27i0.8003>
- Kaplan, H. L. (1985). Design decisions in a Pascal-based operant conditioning system. *Behavior Research Methods. Instruments, & Computer*, 17(2), 307-318. <https://doi.org/10.3758/BF03214401>
- Koffarnus, M. N., Bickel, W. K., & Kablinger, A. S. (2018). Remote Alcohol Monitoring to Facilitate Incentive-Based Treatment for Alcohol Use Disorder: A Randomized Trial. *Alcoholism, Clinical and Experimental Research*. <https://doi.org/10.1111/acer.13891>
- Lindgren, S., Wacker, D., Suess, A., Schieltz, K., Pelzel, K., Kopelman, T., . . . Waldron, D. (2016). Telehealth and Autism: Treating Challenging Behavior at Lower Cost. *Pediatrics*, 137 Suppl 2, S167-175. <https://doi.org/10.1542/peds.2015-2851O>

- Marsch, L., Lord, S., & Dallery, J. (2014). *Behavioral healthcare and technology: Using science-based innovations to transform practice*. Oxford University Press.
- Plavnick, J. B., Mariage, T., Englert, C. S., Constantine, K., Morin, L., & Skibbe, L. (2014). Promoting Independence During Computer Assisted Reading Instruction for Children with Autism Spectrum Disorders. *Revista Mexicana de Análisis de la Conducta* , 40, 20. <https://doi.org/10.5514/rmac.v40.i2.63667>
- Pope, D. A., Poe, L., Stein, J. S., Kaplan, B. A., Heckman, B. W., Epstein, L. H., & Bickel, W. K. (2018). Experimental tobacco marketplace: substitutability of e-cigarette liquid for cigarettes as a function of nicotine strength. *Tobacco Control*. <https://doi.org/10.1136/tobaccocontrol-2017-054024>
- Pyramid Educational Consultants. (2018). PECS Phase III: Pyramid Group Management.
- Quisenberry, A. J., Koffarnus, M. N., Epstein, L. H., & Bickel, W. K. (2017). The Experimental Tobacco Marketplace II: Substitutability and sex effects in dual electronic cigarette and conventional cigarette users. *Drug and Alcohol Dependence*, 178, 551-555. <https://doi.org/10.1016/j.drugalcdep.2017.06.004>
- Quisenberry, A. J., Koffarnus, M. N., Hatz, L. E., Epstein, L. H., & Bickel, W. K. (2015). The experimental tobacco marketplace I: Substitutability as a function of the price of conventional cigarettes. *Nicotine & Tobacco Research*, 18(7), 1642-1648. <https://doi.org/10.1093/ntr/ntv230>
- R Core Team. (2017). R: A language and environment for statistical computing (Version 3.4.1): R Foundation for Statistical Computing.
- Raiff, B. R., Fortugno, N., Scherlis, D. R., & Rapoza, D. (2018). A Mobile Game to Support Smoking Cessation: Prototype Assessment. *JMIR Serious Games*, 6(2), e11. <https://doi.org/10.2196/games.9599>
- Reynolds, B., Dallery, J., Shroff, P., Patak, M., Leraas, K., & Silverman, K. (2008). A Web-Based Contingency Management Program With Adolescent Smokers. *Journal of*

- Applied Behavior Analysis*, 41(4), 597-601. <https://doi.org/10.1901/jaba.2008.41-597>
- Skinner, B. F. (1956). A case history in scientific method. *American Psychologist*, 11(5), 221.
- Sutherland, R., Trembath, D., & Roberts, J. (2018). Telehealth and autism: A systematic search and review of the literature. *International journal of speech-language pathology*, 20(3), 324-336. <https://doi.org/10.1080/17549507.2018.1465123>
- Tomlinson, S. R. L., Gore, N., & McGill, P. (2018). Training Individuals to Implement Applied Behavior Analytic Procedures via Telehealth: A Systematic Review of the Literature. *Journal of Behavioral Education*, 27(2), 172-222. <https://doi.org/10.1007/s10864-018-9292-0>
- Wacker, D. P., Lee, J. F., Dalmau, Y. C., Kopelman, T. G., Lindgren, S. D., Kuhle, J., . . . Waldron, D. B. (2013). Conducting functional analyses of problem behavior via telehealth. *Journal of Applied Behavior Analysis*, 46(1), 31-46. <https://doi.org/10.1002/jaba.29>
- Watson, J. B. (1916). The place of the conditioned-reflex in psychology. *Psychological Review*, 23(2), 89-116. <https://doi.org/10.1037/h0070003>
- Whitcomb, S. A., Bass, J. D., & Luiselli, J. K. (2011). Effects of a Computer-Based Early Reading Program (Headsprout®) on Word List and Text Reading Skills in a Student with Autism. *Journal of Developmental and Physical Disabilities*, 23(6), 491-499. <https://doi.org/10.1007/s10882-011-9240-6>
- Young, M. E. (2018). A place for statistics in behavior analysis. *Behavior Analysis: Research and Practice*, 18(2), 193-202. <https://doi.org/10.1037/bar0000099>

## Chapter 2

### **A step-by-step guide to develop experiments with Axure© RP**

Théo P. Robinson<sup>1</sup>

*Florida Institute of Technology, FL, USA*

#### **Abstract**

Programming an experiment with humans from start to finish is not always an easy task. Depending on the tool used, it may be something just for technology professionals. In this tutorial you will learn step-by-step how to develop a computer experiment in a very simple way using Axure© RP. Axure© RP was originally conceived so that people in the IT field could quickly develop a draft of the software or website that their customers need. To demonstrate how to use the platform to create experiments, first the Axure© RP interface will be presented and later we will build everything necessary for an experiment of choice with two competing options. In section I, the general arrangements of stimuli on the screen will be configured. In section II, we will add functionality to the items we created in the previous section. In section III, we will program button 1 to reinforce responses in an FR1 reinforcement schedule. In section IV, button 2 will be configured to reinforce responses in VR3. Section V will be used to add intervals between attempts to choose the participant. Finally, section VI will be for configuring the output of the data collected during the experiment. This chapter is only intended to initiate readers in the development of experiments using Axure© RP. It is certainly possible to create more complex screens, with more elaborate features than we learned here, including transporting the experiment to run on online platforms with Amazon's Mechanical Turk ("MTurk").

---

1 Please send correspondence to Théo P. Robinson to [theorobinson2012@gmail.com](mailto:theorobinson2012@gmail.com).

What you will need to follow this chapter:

- Axure© RP 8 (RP 9 is presently in beta testing)
- A personal computer
- An internet browser (Preferably Google© Chrome)

New researchers often aspire to conduct behavioral research with humans, but lack the prerequisite skills necessary to program an entire experiment. In this tutorial, we are going to take a new approach to the development of computer-based experiments. We will be doing this using Axure© RP, an application intended for software prototyping, or “wire-framing,” as it is called. Axure© RP is marketed to software developers who want to quickly develop a visual depiction of an application or website, before they—or companies who hire them—spend valuable resources developing the real application.

According to the developers of Axure© RP, Axure Software Solutions, Inc., the application was initially developed for three reasons. First, they wanted to be able to quickly construct prototypes of software applications. Second, the developers tried to create a platform for efficiently collecting feedback and re-designing prototypes. Third, they wanted the prototypes to act as guides, or outlines, for the overall development of the application. In this tutorial, we will re-purpose Axure© RP, with the goal of developing a behavioral experiment for humans.

The developers of Axure© RP offer free classroom licenses on their website. It is highly recommended that you (or a professor at your institution) apply for a free license because the cost of the application is quite high. This demo was created using Axure© RP9 which—at the time of writing this chapter—is still in beta testing. I designed this tutorial with the hope that the release of Axure© RP9 will closely coincide with the publication of this book. You can still complete this tutorial if you only have access to Axure© RP8, although some components of the user interface are different between versions. For example, the **Interactions tab** in Axure© RP9 (Figure 1) is called “Properties” in Axure© RP8.



Hopefully, the implications of Axure© RP for experimental design and implementation will come to light throughout this tutorial. First, we will develop a general protocol for a simple choice “experiment”. Next, we will learn how to collect data in Axure© RP. Finally, the experiment should present data in a way that can be easily copied and pasted into a Microsoft© Excel document.

### Brief presentation of the graphical interface

Next (Figure 1) you can see a brief description of the main controls of the graphic interface of the Axure © RP. Throughout the text, each of the major controls will be presented in bold.

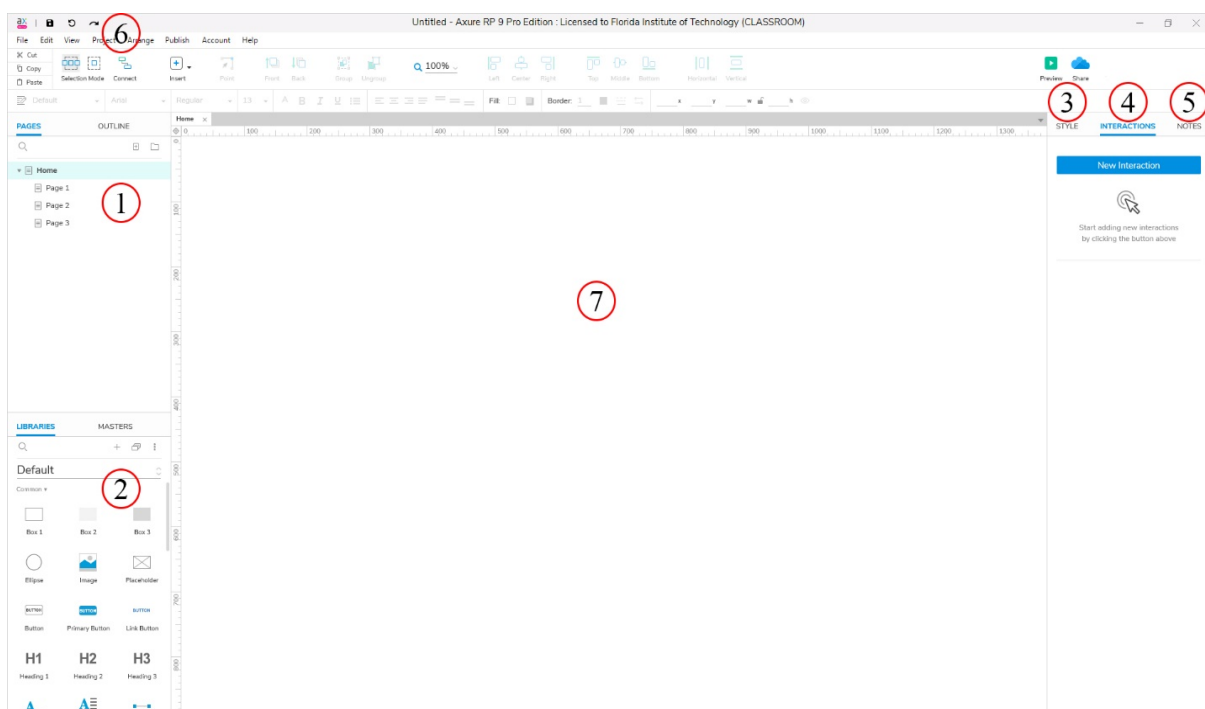


Figure 1. Main UI (user interface) controls of Axure© RP. Page pane (1), Widget pane (2), Style tab (3), Interactions tab (4), Notes tab (5), Menu bar (6), Work area (7).

#### Page pane (1)

Displays all of the pages involved with your project. *Home*, *Page 1*, *Page 2*, and *Page 3* are added to the project by default. This experiment will only use the *Home* page and *Page 1*.

**Widget pane (2)**

“Widgets” are the interactive objects that the user inserts into the project. These are added to pages by dragging and dropping them from the **Widget pane (2)** to the **Work area (7)**. Widgets include text boxes, buttons, labels, shapes, and many others.

**Style tab (3)**

Enables formatting the appearance of the page and the widgets on the page. Fill color, border color, text color, line width, and many other visual characteristics can be formatted by clicking on a widget and then clicking on the **Style tab (3)**.

**Interactions tab (4)**

This is where the magic happens. It permits the construction of various interactions between widgets. Interactions can also be assigned to the page itself by clicking in an empty part of the **Work area (7)** and then clicking on the **Interactions tab (4)**. So called, “actions,” can be created within interactions. To clarify, interactions are triggers assigned to widgets or directly to the page, and actions are the events that tell the Axure© RP engine what to do based on the trigger (interaction) of the widget or the page.

**Notes tab (5)**

Add notes to specific widgets or the page. This experiment will not use this tab, but it is useful for adding additional details to certain variables. For example, users might add the specifics of various schedules of reinforcement to certain widgets they are assigned to.

**Menu bar (6)**

Similar to most menu bars in other computer applications (e.g., Microsoft© Office, Microsoft© Visual Studio). Comprises *File, Edit, View, Project, Arrange, Publish, Account, Help*; the most important are *Project* and *Publish*. Access the Global Variables within the *Project* menu, and “Preview” or run the experiment in the *Publish* menu.

## Work area (7)

Visually displays the widgets that have been added to pages. Remember, this area can be formatted and assigned interactions in almost the same way as widgets. Additionally, remember that the **Style tab (3)**, **Interactions tab (4)**, and **Notes tab (5)** can be accessed by left-clicking in any empty (or “white”) area of the **Work area** and then navigating to the desired tab. Furthermore, right-clicking in an empty area will open a menu, wherein elements such as gridlines and guides can be displayed in the **Work area** (note that these gridlines and guides are not visible when running the experiment). Finally, right-clicking on a widget within the **Work area** brings up a menu with many options; this tutorial primarily uses the *Set Hidden* option of this menu.

### Introduction

To begin, we will be creating a choice experiment with two concurrently available options. These options will be presented in the form of clickable buttons—one blue and one green. The blue button will be placed under a FR1 schedule of reinforcement, and the green button will be placed under a VR3 schedule of reinforcement. The participants earn points by clicking on either button, and their goal is to earn as many points as possible within one minute. In the end, the experiment will look like Figure 2.

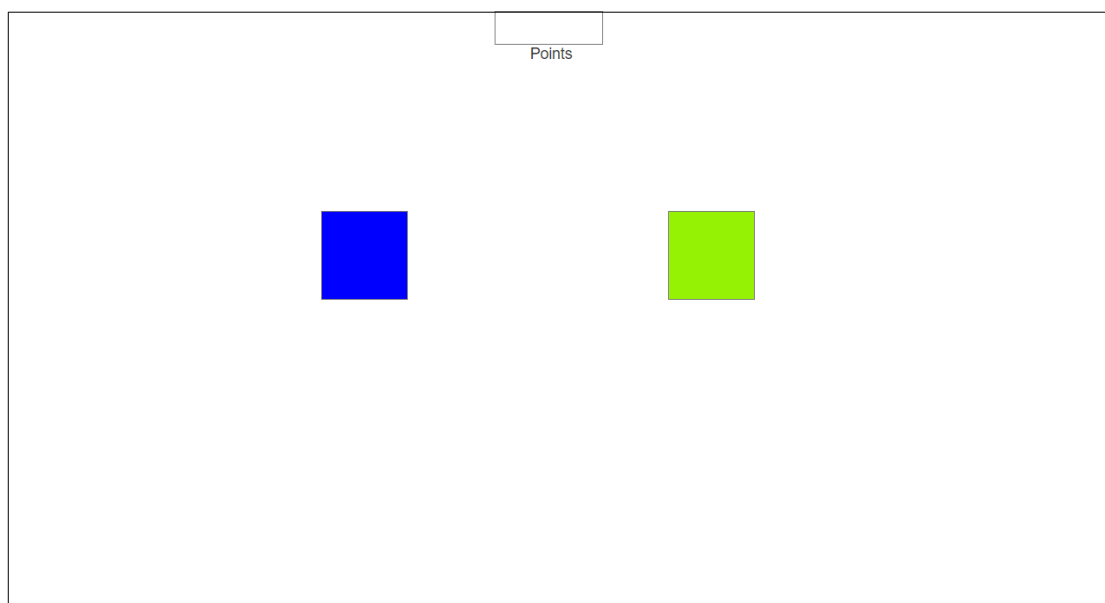


Figure 2. Appearance of our experiment at the end of our step-by-step guide.

## In-Text Color Legend

Global variables are displayed in PURPLE.

Local variables are displayed in GREEN.

Widget names are displayed in RED.

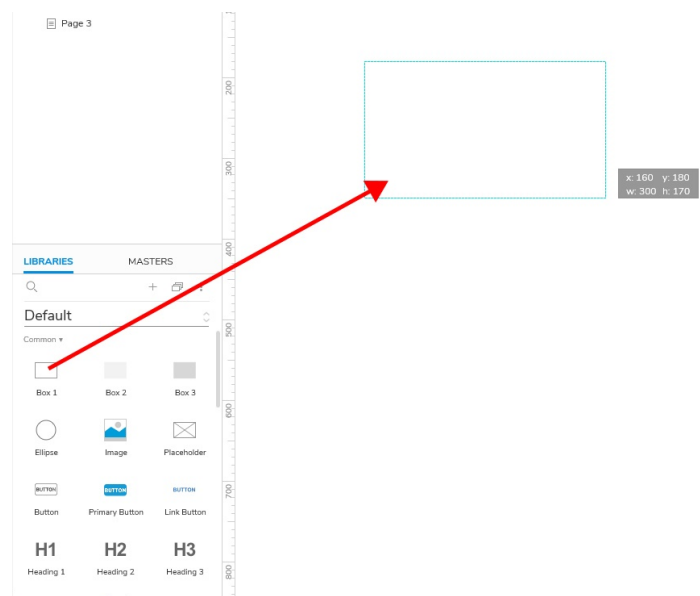
## Section I | An Experimental Blueprint

In this section, we will set up the general arrangement of the on-screen stimuli. We can work out the logistics of the arrangement later; for now, let's focus on creating the general layout of our experiment. The buttons will be the same size and shape, but they will differ in color. Finally, a points box will be added for the participant to keep track of his or her points.

Objectives:

- Create two boxes that are different colors (These boxes will later become buttons).
- Create a points box.

1. We will use two boxes as buttons in this experiment. Add your first button by dragging a white box widget from the **Widgets pane** to the **Work area**, as shown in Figure 3.



*Figura 3.* The red arrow illustrates the drag n' drop action (hold the left mouse button down with its cursor over a component and then move it from one control to another by releasing the mouse button at the target location).

2. Resize this box so the dimensions are 80px by 80px. To do this, look in the top-right of the **Menu bar** and type “80” for the **w** (width) and **h** (height) fields, as shown in Figure 4.

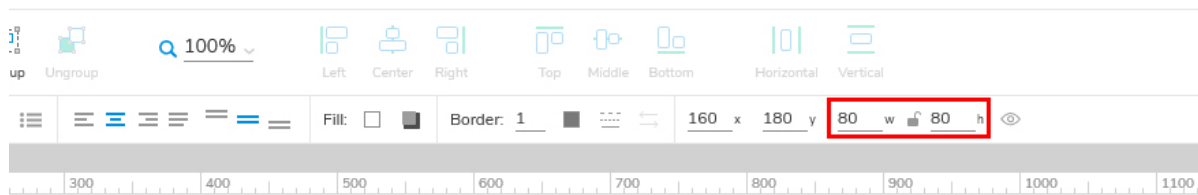


Figure 4. Fields **w** e **h** represent respectively width and height of selected boxes.

3. Right-click on the box and select “Copy.” Now, right click anywhere in the **Work area** and click paste. Arrange the boxes however you like—I made them horizontally aligned for the sake of aesthetics, as shown in Figure 5.

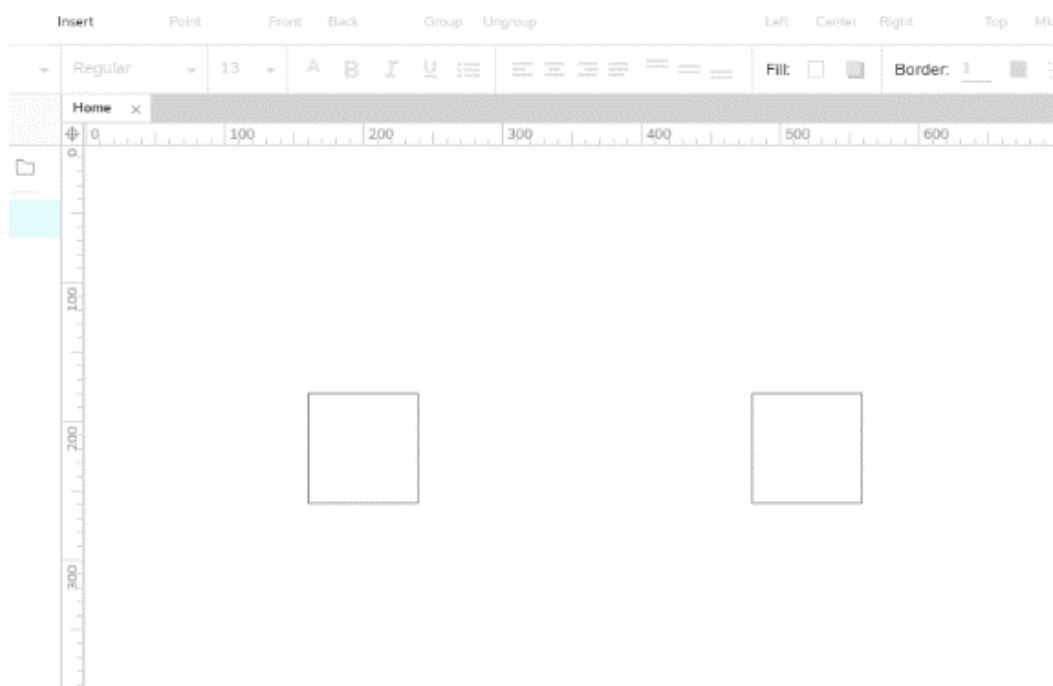


Figure 5. The boxes were horizontally aligned for aesthetic reasons only.

4. Now that we have our boxes, we should name them. Click on the first box we made and navigate to the **Interactions tab**. Replace the first box’s name, “(Rectangle Name)”, with “**aTarget**”. Name the second box “**bTarget**”, as shown in Figure 5.

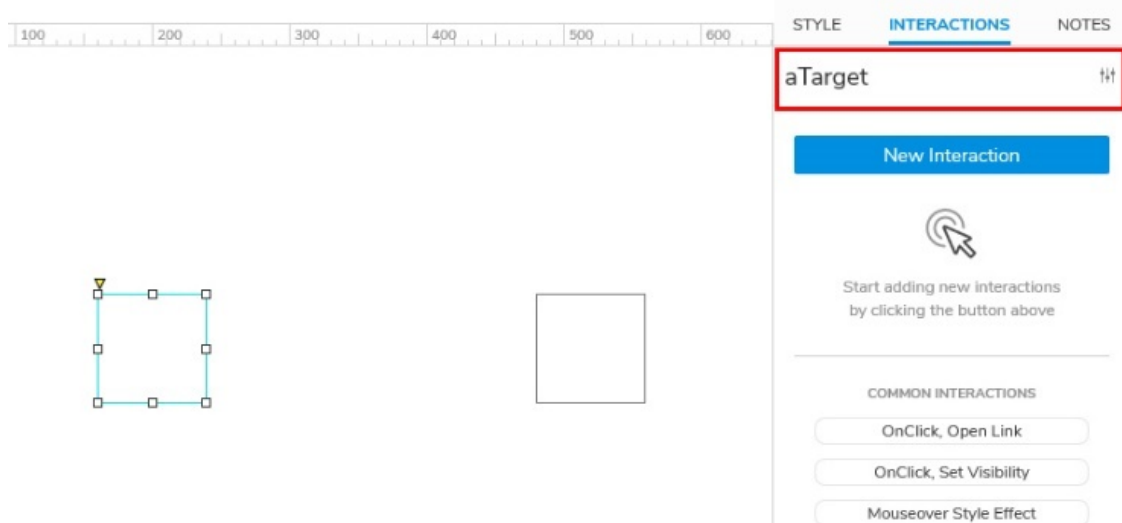


Figure 6. Boxes can be renamed through the (highlighted in red) field of the **Interactions** tab.

Naming widgets is important because it allows you to refer to these widgets directly in future steps. Specifically, interactions and actions (covered later in this chapter) will be added between different widgets; therefore, naming all widgets is key to not becoming confused and/or selecting the incorrect widget when setting up an action or interaction.

5. To change the color of the boxes, click on the **Style** tab. The style of all widgets can be edited within this tab. Change **aTarget** to blue and change **bTarget** to green, as shown in Figure 7.

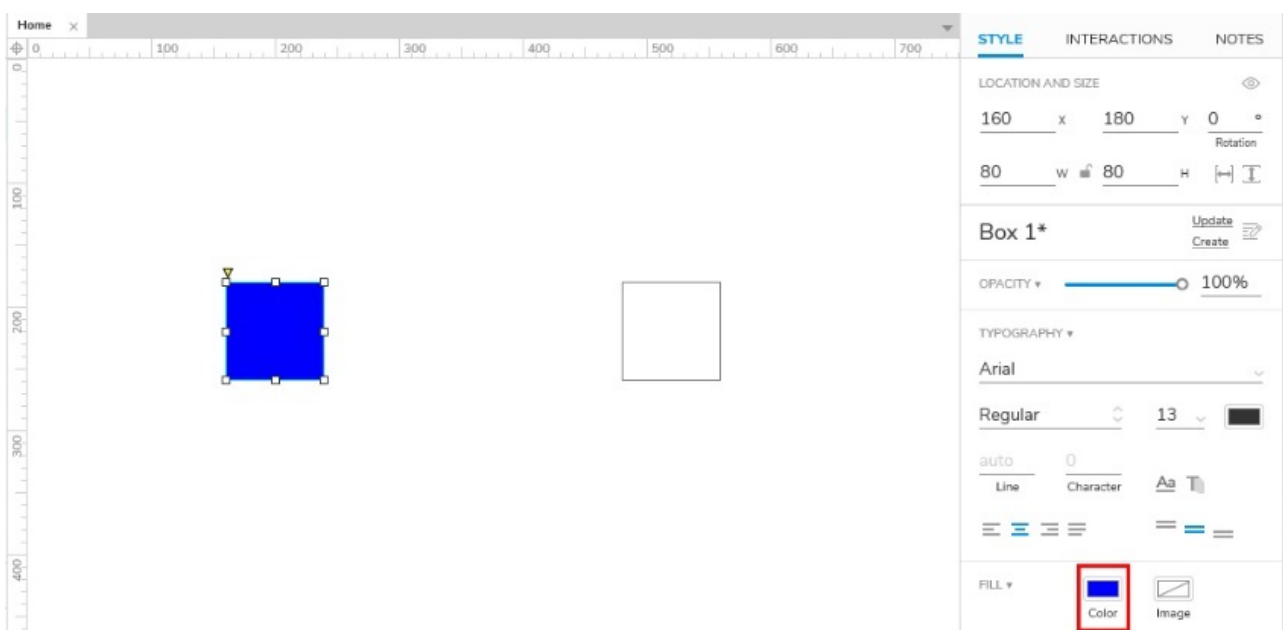


Figure 7. The color of each box can be edited using the Color tab in the Styles tab.

6. Next, we will add a points box at the top of the screen to track how many points the participant has earned. To do this, drag another white box widget to the top of the screen. Change the dimensions of this box to a rectangle, with the dimensions: 100px by 30px. Change the name of this rectangle to **pointsBox**, by following Step 4. Drag a label widget under the points box that we just added. Change the text in this label widget to “Points”, as shown in Figure 8. We will not name this label widget because it will not be used in future actions or interactions; it is simply a label designed to orient the participant to the area where his or her points will be displayed.

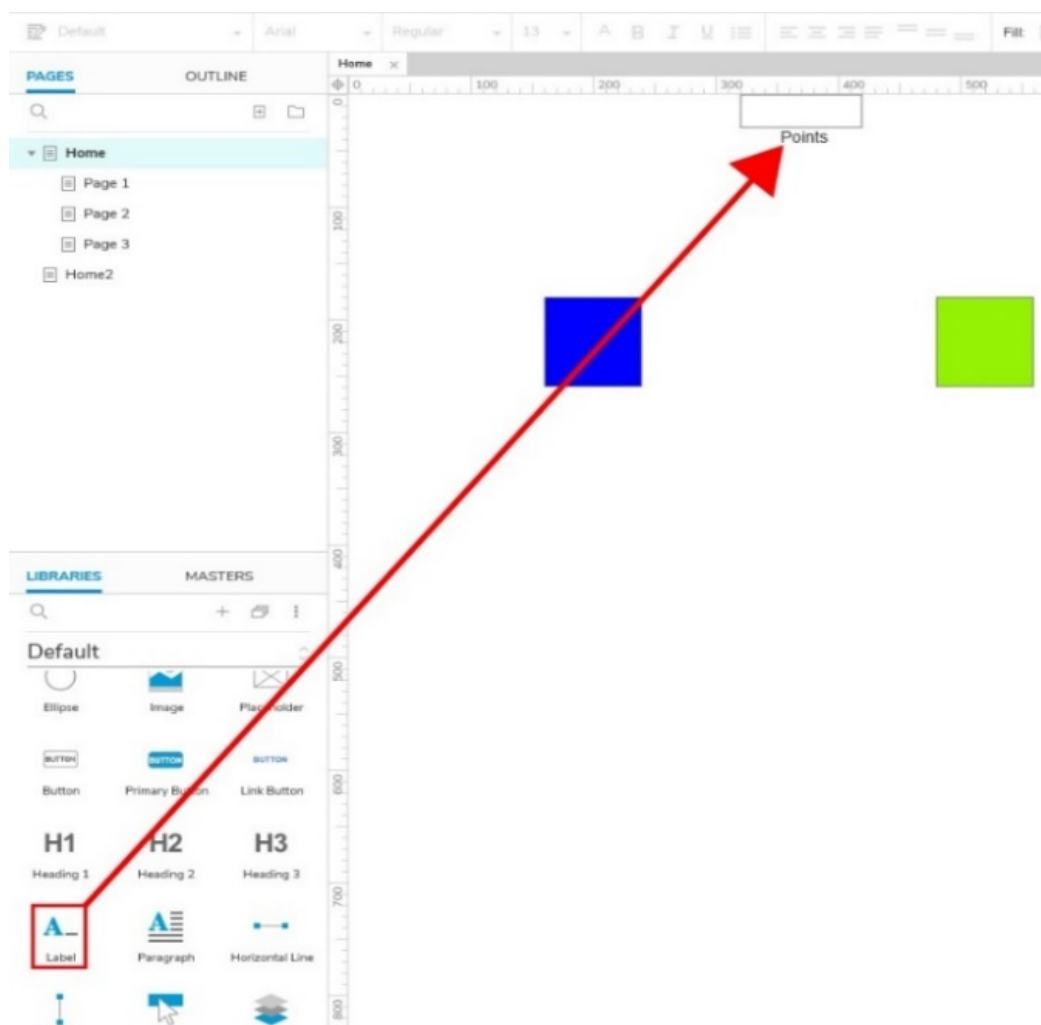


Figure 8. You can add a label to a visual control via the Label widget.

In their current state, the green and blue boxes lack functionality with respect to what we want them to do as buttons. Likewise, we have to add functionality to the points box as well. The

next section will go a little more in-depth—we'll remove the ambiguity of those boxes and unlock their true potential as buttons!

## Section II | Adding Functionality

Objectives:

- Create two global variables to store responses on each button
- Create one global variable to keep track of points earned
- Create interactions to keep track of the number of clicks on each button

Note: Throughout this section, and future sections, we will be using the **Interactions tab** of widgets to trigger certain events. It is very important that these interactions follow the same chronological order that they are presented in these instructions. The **Interactions tab** follows a chronological order throughout these next steps, so be sure to organize your actions and interactions correctly.

7. In the **Menu bar**, click on “Project,” and then click on “Global Variables”, as shown in Figure 9.

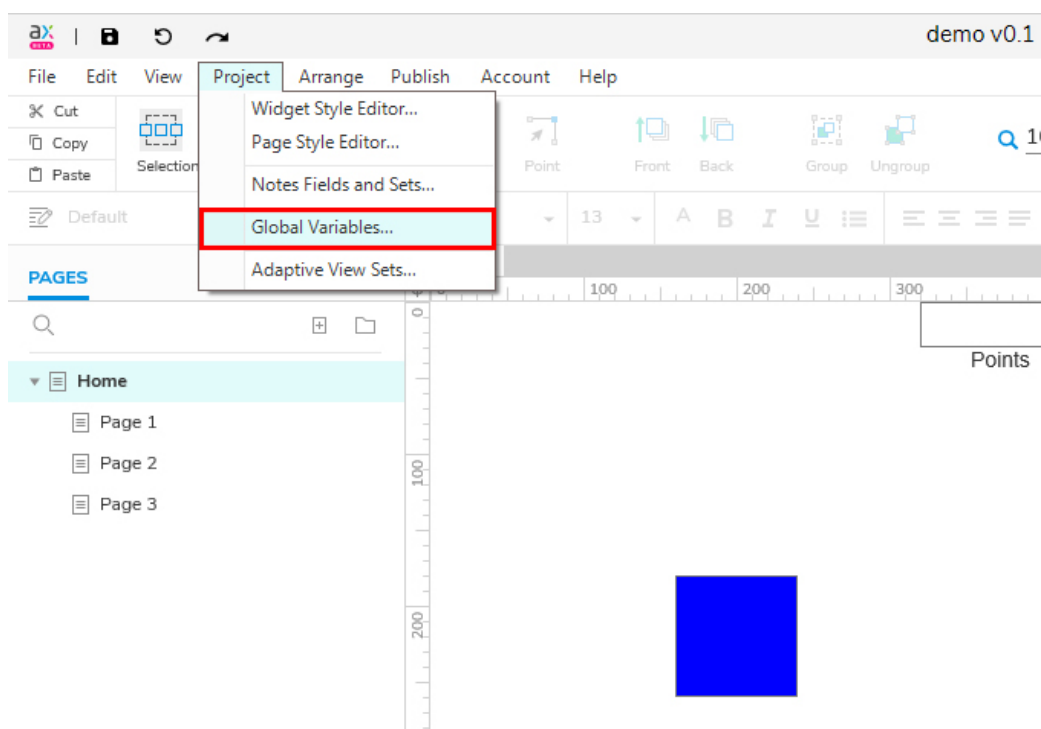


Figure 9. Global variables can be created from the Project option in the menu bar.



8. In the window that pops up, click the “Add” button. For now, we are only adding variables, so we do not need to pay attention to the “Default Value” column. We will need three variables in all: two variables will keep track of button presses (**aResponses** and **bResponses**) and one variable will keep track of the participant’s total points (**totalPoints**). You will need to click the “+ Add” button (highlighted with a red rectangle) for each new variable that you add. When all the variables have been added, your Global Variables window should look like Figure 10. If it does, click the “OK” button.

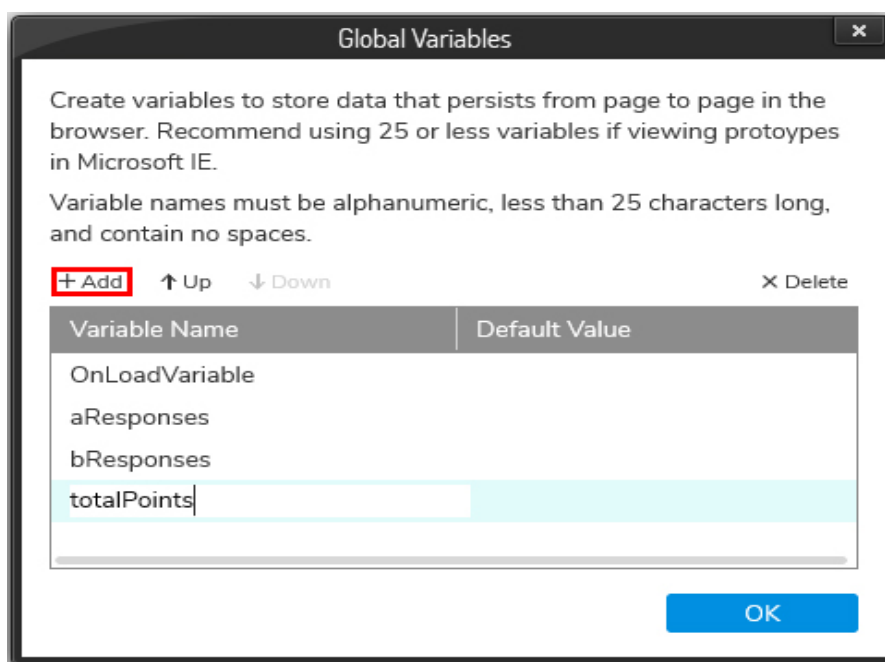


Figure 10. Click at the + Add button to add a new variable.

9. Next, click on the blue box, **aTarget**, and then navigate to the **Interactions** tab. Click the “New Interaction” button, and select “OnClick” from the drop-down (highlighted in red). If this option is not appearing, ensure that you have the blue box (**aTarget**) selected before you click the “New Interaction” button, as shown in Figure 11.

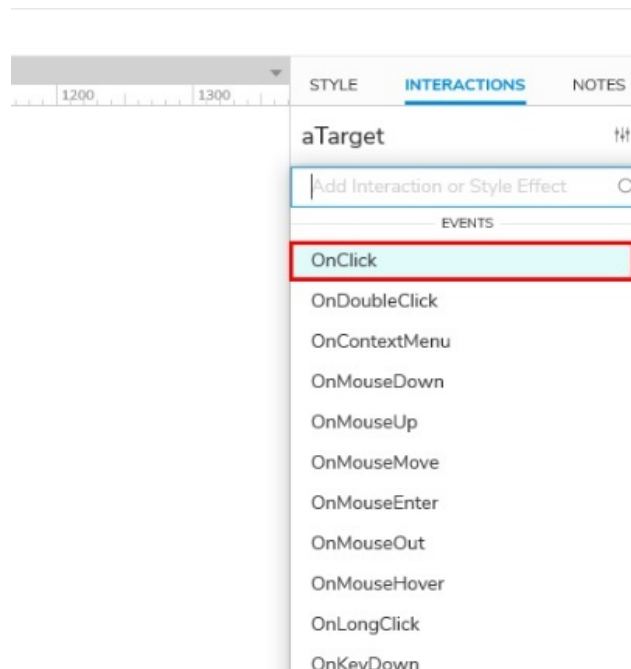


Figure 11. Select the "OnClick" option from the drop-down menu to add a click event to the box.

10. After selecting "OnClick," select "Set Variable Value", as shown in Figure 12. Basically, we are going to tell Axure© RP to set the variable, **aResponses** equal to the number of times the participant clicks on the button.

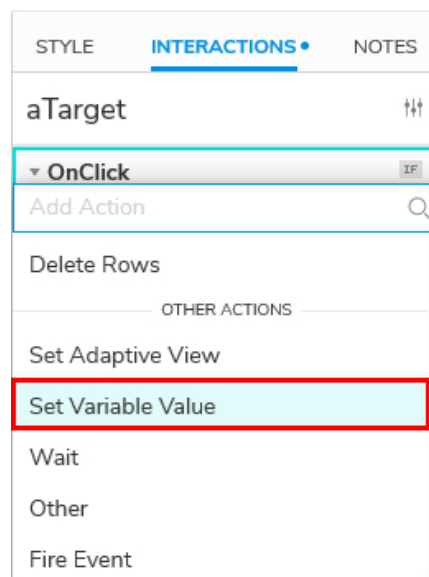


Figure 12. The Set Variable Value option makes the aResponses variable equal to the number of times the participant clicks the button.

11. Choose **aResponses** in the TARGET field, as shown in Figure 13.

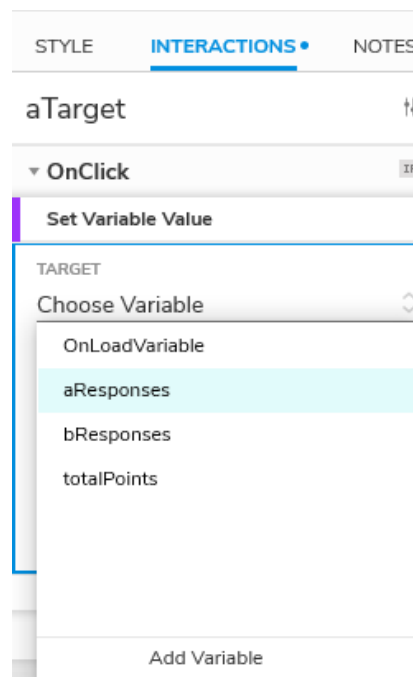


Figure 13. The aResponses option in the TARGET field is found in the Interaction tab.

12. The SET TO field should have the word “text” in it. Add the following statement into the VALUE box:

**[[aResponses + 1]]**

The statement tracks the number of responses made on **aTarget** by adding 1 to the value already held in the **aResponses** variable, when the OnClick event of the **aTarget** button is triggered (i.e., when the button is clicked). Then click the “Done” button, as shown in Figure 14.

STYLE	INTERACTIONS •	NOTES
aTarget		↑↑↑
▼ OnClick		IF
Set Variable Value		
aResponses to "[[aResponses + 1]]"		
TARGET	aResponses	↕
SET TO	text	↕
VALUE	[[aResponses + 1]]	$\int_x$
Delete		Done

Figure 14. Click the Done button to confirm editing the options.

13. Repeat Steps 9 through 12 for **bTarget**; therefore, you should replace any leading ‘a’s with ‘b’s for variables and widget names (see Figure 15). Additionally, remember that you should click on the **bTarget** widget and navigate to the **Interactions tab**. For example, we want the VALUE box to read:

[[bResponses + 1]]

Click the “Done” or “OK” button if your **Interactions tab** looks like Figure 15.

STYLE	INTERACTIONS •	NOTES
bTarget		↑↑↑
▼ OnClick		IF
Set Variable Value		
bResponses to "[[bResponses + 1]]"		
TARGET	bResponses	↕
SET TO	text	↕
VALUE	[[bResponses + 1]]	$\int_x$
Cancel		OK

Figure 15. Click the Done button to confirm editing the options.

In their current state, the buttons do little to tell us about the number of responses the participants made to each button because these data are stored in variables that we presently do not have access to. This issue will be placed on hold for a moment, and addressed in Section VI. The next section, Section III, will focus on the **totalPoints** variable and program a FR1 schedule under the blue button, **aTarget**.

### Section III | Programming an FR1 Schedule

In this example, the blue button (**aTarget**) will be under a FR1 schedule, and the green button (**bTarget**) will be under a VR3 schedule of reinforcement.

Objectives:

- Program a FR1 schedule of reinforcement for responses made on **aTarget**
- Increment the value in the **pointsBox** widget by 1 whenever the schedule requirement

is met

14. Reinforcement for **aTarget** will be easy to construct because it is under a FR1 schedule. Open the **Interactions** tab for **aTarget**. Add a second “Set Variable Value” action by clicking “+ Add Variable” (or the “+ Add Target” button), as shown in Figure 16.

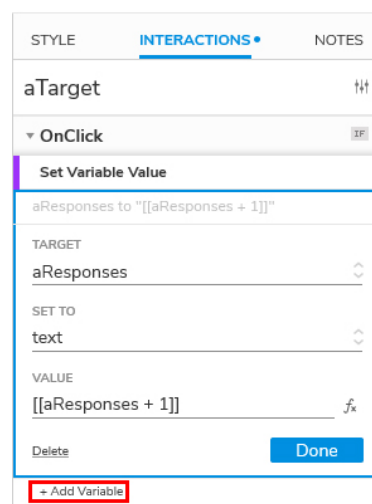


Figure 16. Add an action by clicking on the "+ Add Variable" button (or the "+ Add Target" button). At the time of writing this, there seems to be a bug in Axure© RP9 in which either "+ Add Variable"

or “+ Add Target” is shown. Either way, you are simply adding another target variable under the Set Variable Value action.

15. Select **totalPoints** in the TARGET column, as shown in Figure 17, keep “text” in the SET TO field, and type the following into the VALUE field:

**[[totalPoints + 1]]**

STYLE	INTERACTIONS •	NOTES
aTarget		↑↑
▼ <b>OnClick</b>		IF
<b>Set Variable Value</b>		
aResponses to "[[aResponses + 1]]"		
totalPoints to "[[totalPoints + 1]]"		
TARGET	totalPoints	↕
SET TO	text	↕
VALUE	[[totalPoints + 1]]	f*
Delete		Done
+ Add Variable		

Figure 17. Select the TARGET column from the Interactions tab.

16. Again, navigate to the **Interactions tab** for **aTarget**. Click on the purple addition sign (+) under your most recently added action, as shown in Figure 18. This will add a new action to be carried out when the OnClick event is triggered.

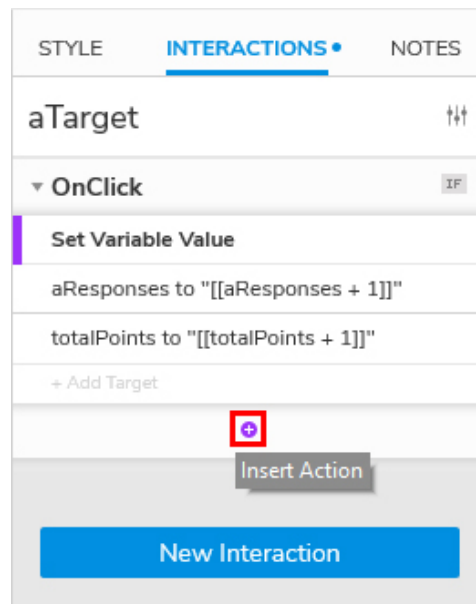


Figure 18. The purple button (highlighted by the red square) allows you to add a new action.

17. Select the “Set Text” option from the drop-down, as shown in Figure 19.

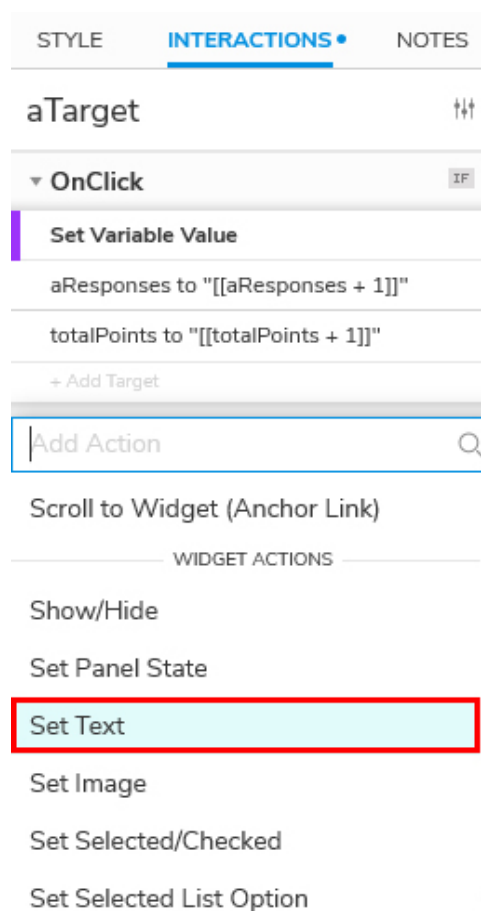


Figure 19. "Set Text" option, highlighted by the red square.

18. In the TARGET box, select your **pointsBox** widget. Keep “text” in the SET TO box. Finally, double-click on the “fx” (function) button to the right of the VALUE box, as shown in Figure 20.

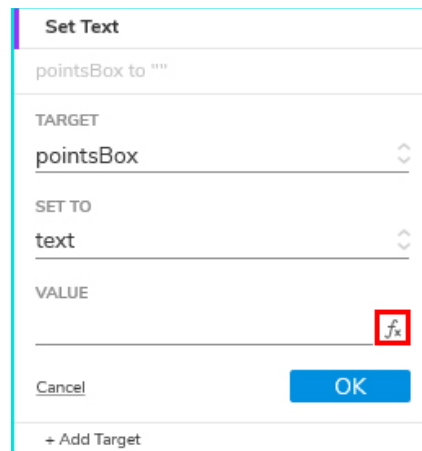


Figure 20. "fx" button, highlighted by the red square.

19. A new window will pop up (see Figure 21); this window has two text areas. The top area is where you enter the function that you would like Axure© RP to carry out whenever the action event is triggered. In this case, you want Axure© RP to add 1 to **totalPoints** whenever the OnClick action is triggered for **aTarget**, **and** you want the output of this function/calculation to be displayed in your **pointsBox** widget.

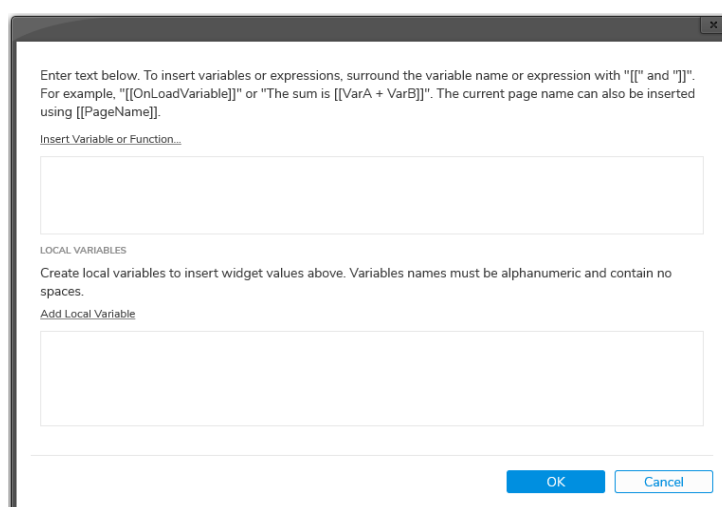


Figure 21. The bottom area of this window is where you assign local variables which are **discrete** to this specific action, and these variables **are not** stored as global variables.



20. To begin, click “Add Local Variable” in the bottom area of the window. This will add **LVAR1** as a local variable. Keep the name as **LVAR1** (you can change this to a different name if you want, but make sure to keep it consistent), and keep “text on widget” entered into the middle drop-down. In the last drop-down, select the widget **pointsBox**. With that, you just declared **LVAR1** as a local variable, which holds the text on the **pointsBox** widget as its value, as shown in Figure 22.

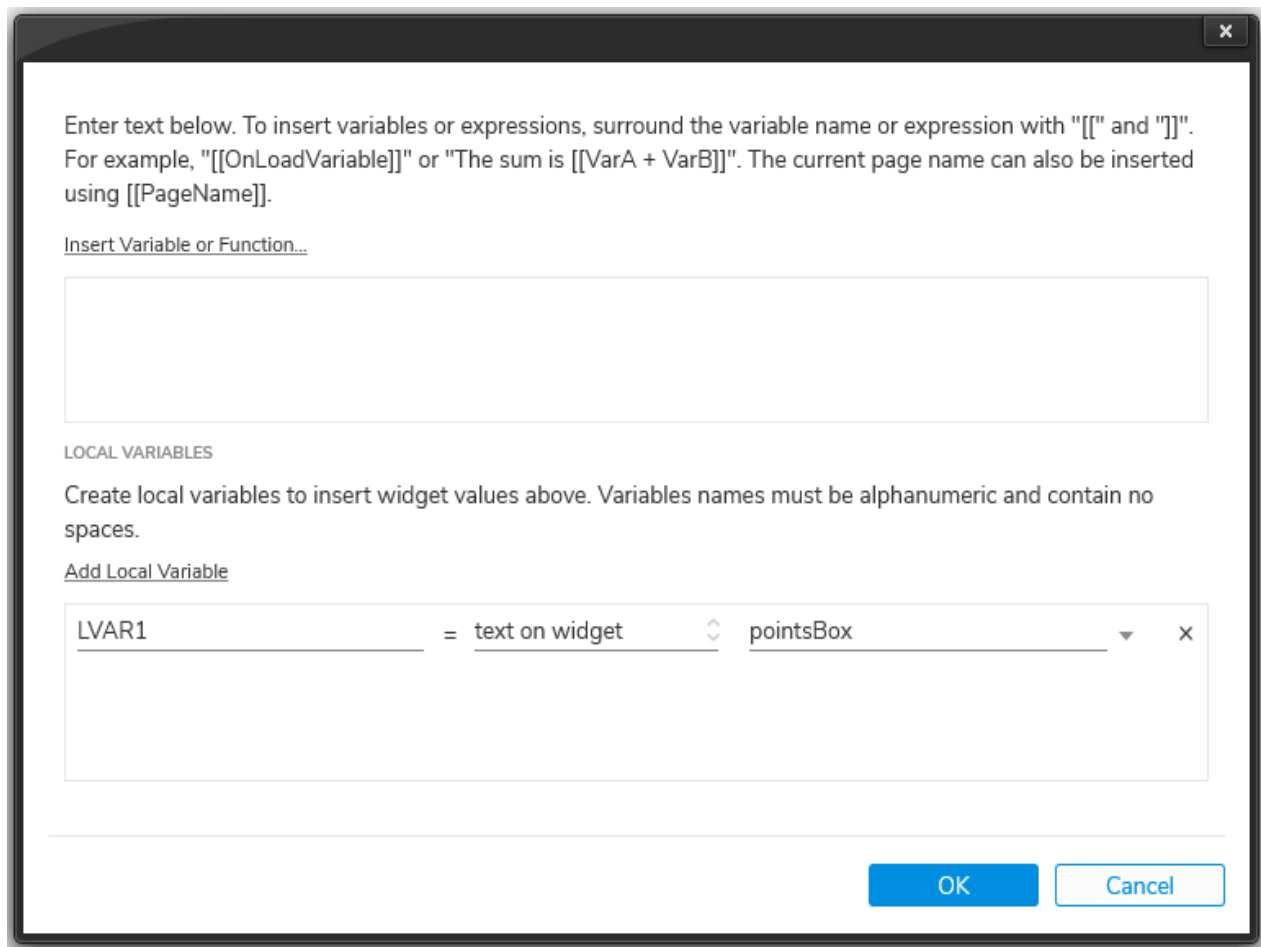


Figure 22. LVAR1 declared as a local variable will keep its value equal to the text of the pointsBox widget.

21. The next step is to program the experiment to increment the participant’s points by 1, whenever the **aTarget** widget is clicked. You have already defined **LVAR1** as the text on **pointsBox**, so you simply need to type the following into the top area of the window (see Figure 23):

**[[LVAR1 + 1]]**

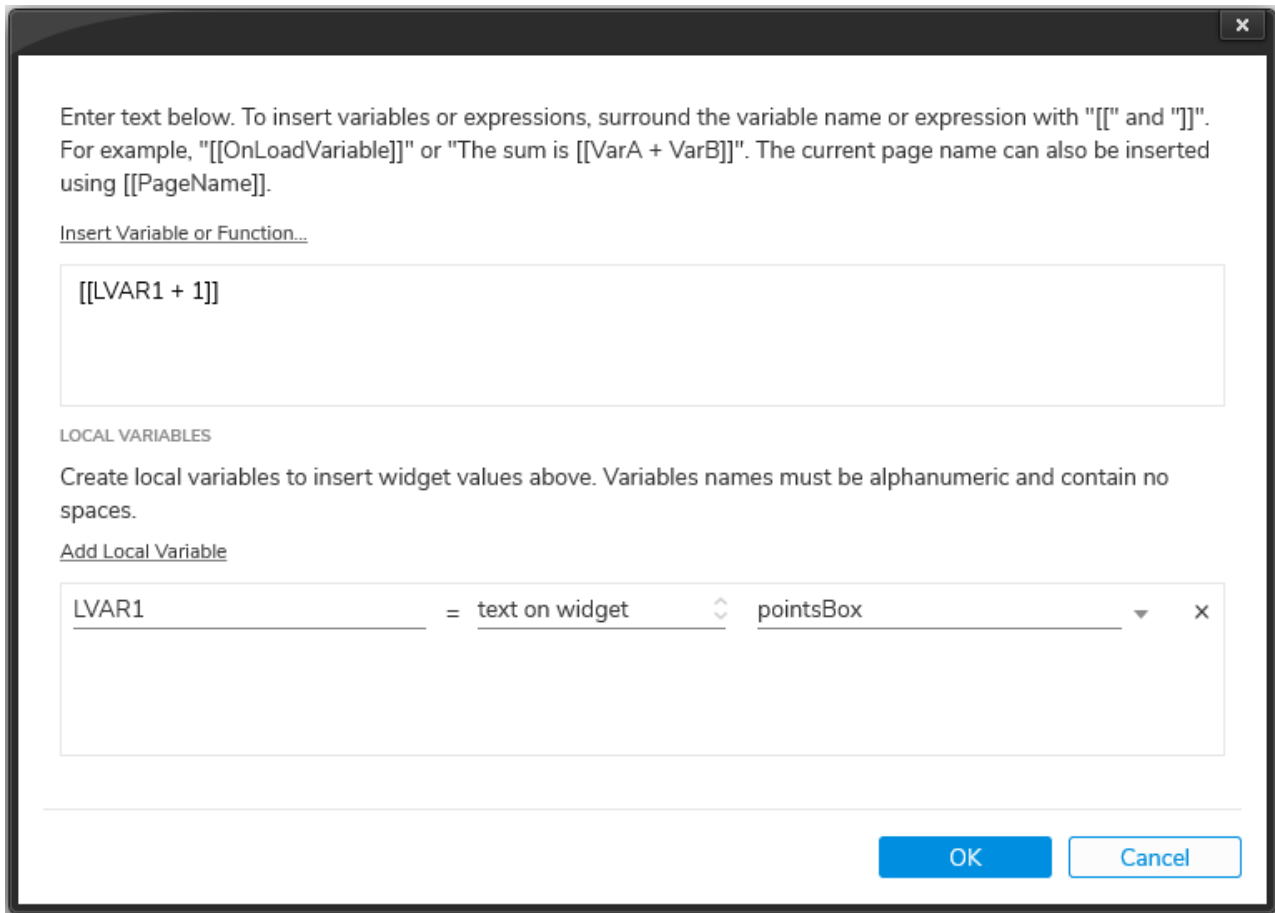


Figure 23. The functions allow you to program the points of the participant.

This will tell Axure© RP to add 1 to whatever value is already in **pointsBox** every time the participant earns a point.

In this section, we placed responses on **aTarget** under a FR1 schedule of reinforcement. In the next section, we will place **bTarget** under a VR3 schedule of reinforcement. Play close attention to this next section, because it includes steps that are more complex than what we have done so far.

#### Section IV | Programming a VR3 Schedule

Here, we will construct a process that randomly generates a value of either 0, 1, or 2 whenever the participant responds to **bTarget**; therein, the participant's response will *only* be reinforced if the value generated is 0.

Note: Before I decided to use this technique to generate VR schedules, I ran five sessions of 20 trials, in which each reinforcer delivery signaled the end of the trial. The number of responses (clicks) made within each trial was recorded. The number of responses was averaged at the end of each 20-trial session, to generate the average rate of reinforcement delivery. The results of this test are displayed in Table 1. These values do not comprise a rectangular distribution. For an alternative way to generate VR response requirements, see Bancroft and Bourret (2008).

*Table 1.* Number of responses per attempt over sessions with response requirements under VR3.

Trial	Session				
	1	2	3	4	5
	Number of Responses				
1	2	2	1	1	1
2	2	1	5	2	2
3	1	8	3	1	7
4	1	2	1	6	2
5	1	3	4	2	4
6	1	2	1	4	1
7	4	3	2	4	5
8	6	3	1	2	1
9	8	1	3	5	2
10	3	3	1	1	1
11	6	2	2	1	4
12	1	4	2	4	12
13	4	3	4	3	6
14	1	8	1	6	1
15	9	1	2	2	3
16	1	13	1	2	1
17	2	3	12	1	1
18	7	2	2	1	1
19	3	1	1	1	3
20	3	2	7	1	2
<b>In-Session Average</b>	3.30	3.35	2.80	2.50	3.00
<b>All sessions Average</b>	2.99				

Objectives:

- Program in a VR3 schedule of reinforcement for **bTarget**
- Increment the value in the **pointsBox** widget by 1 whenever the schedule requirement

is met

22. Begin by creating another global variable, and name it **sch**. For a reminder of how to create new variables, see Step 8 (Project -> Global Variables). Enter the value, “3” (without quotes) into the “Default Value” column of the window. Basically, this variable represents the average response requirement for your VR schedule—in this case, it is 3. For example, if you want assign a VR6 schedule of reinforcement under the **bTarget** button, you would enter a “6” into the Default Value column. When your Global Variables window matches Figure 24, click OK.

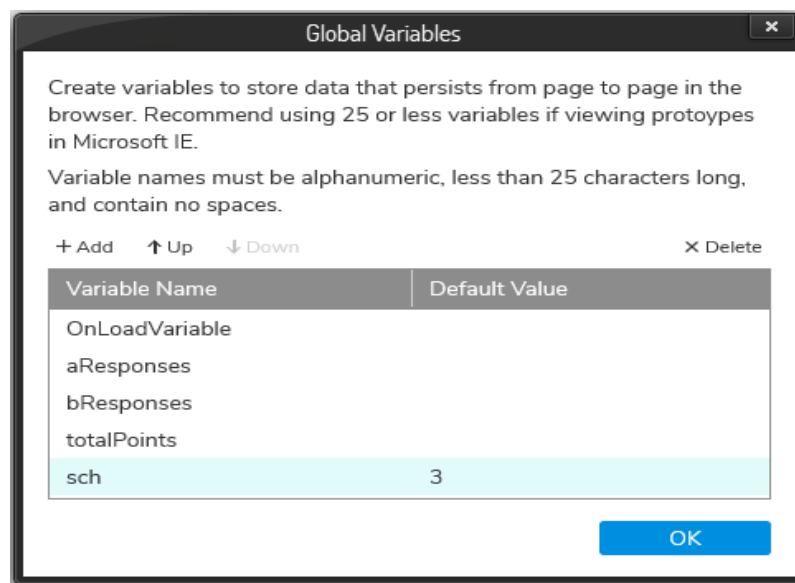


Figure 24. Creating and configuring a global variable.

23. We need to create a widget for “deciding” whether or not to provide reinforcement. When the participant clicks on the **bTarget** button, it will generate a random number (either 0, 1, or 2). Then, the deciding widget will determine if the response should be reinforced based on the randomly generated value. Specifically, reinforcement is delivered only if “0” is the randomly generated value whenever the participant clicks **bTarget**.

Create one box widget, name it **bDecide**, and make the dimensions 30px by 30px. Place the widget in the top-left corner of the screen. Right click on **bDecide** and select *Set Hidden* to hide it from the participant, as shown in Figure 25. Even though the widget is hidden, it will still serve the

function that we are going to assign it—the widget will simply not be visible or useable by the participant.

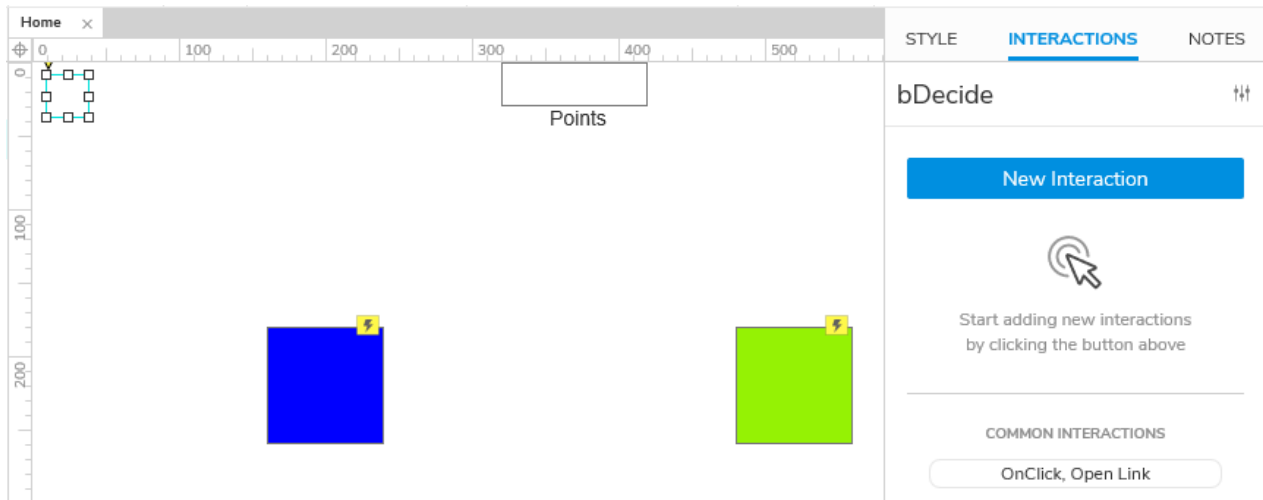


Figure 25. The widget in the upper left corner will not appear for the participant because it was hidden. Once you have hidden the **bDecide** widget, it will appear yellow on the Axure© RP **Work area**, but it will not appear in the browser for the participant.

24. Click on the **bTarget** button and navigate to the **Interactions tab**. Insert a “Set Text” action (using the purple “+” sign) and set the widget, **bDecide**, as the target. Enter this formula into the VALUE box:

$$[[\text{Math.floor}(\text{Math.random()} * (\text{sch}))]]$$

This formula (see Figure 26 also) will set the text on the widget **bDecide** to a random number between 0 and the value assigned to **sch**, and round it to the nearest whole number (whenever **bTarget** is clicked).

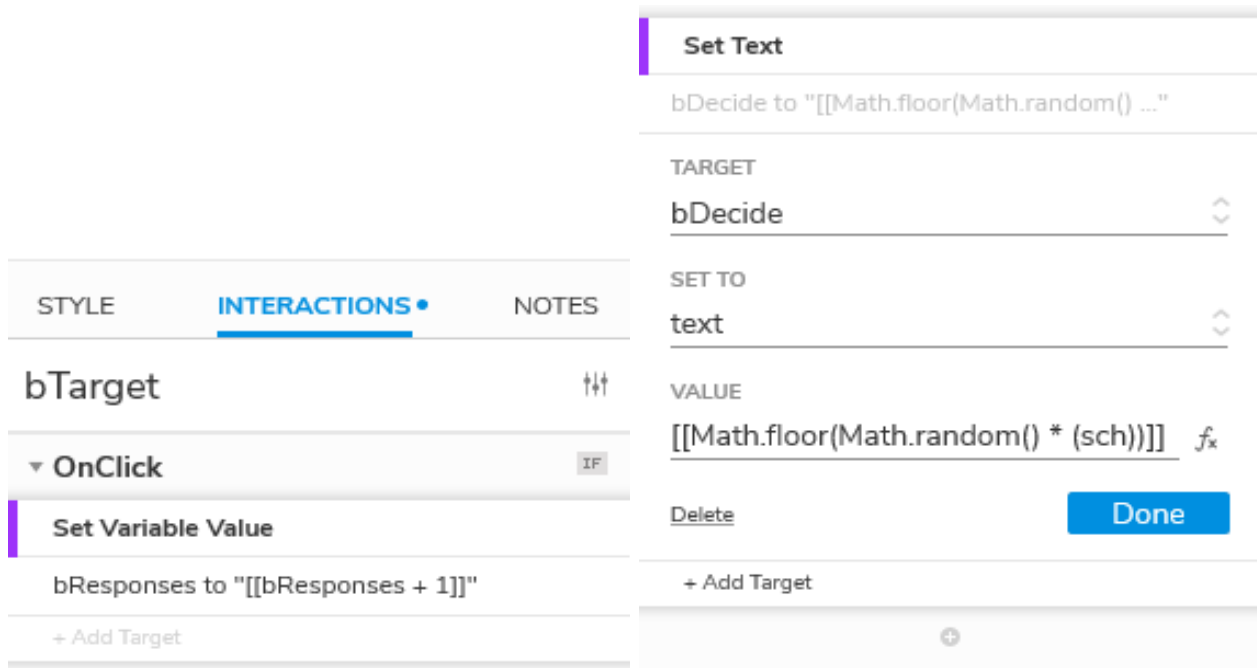


Figure 26. The *Math.random* function generates a random value between 0 and 1, then the *Math.floor* function rounds this value to the nearest whole number.

25. Now, we will add a triggering event to **bTarget**, which will activate the “decision” process for the randomly generated value on **bDecide**. To do this, add a “Fire Event” action to **bDecide**; select **bDecide** as the TARGET, and OnClick as the EVENT, as shown in Figure 27.

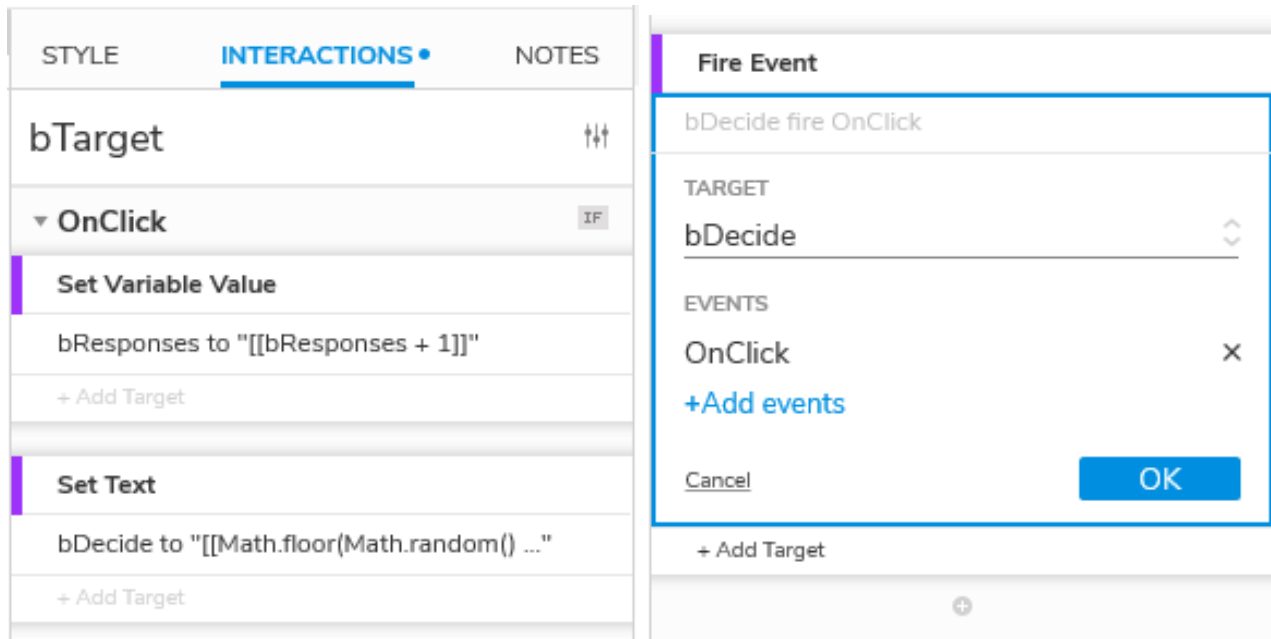


Figure 27. Ensure that your **Interactions** tab for **bTarget** matches the image. It is important that these actions follow the exact chronological order as the ones in the image.

26. In this step, we will add a conditional statement which adds 1 point to the **totalPoints** variable, as well as insert this new **totalPoints** value into the **pointsBox** widget, *only when the randomly generated value on **bDecide** equals 0*.

Open the **Interactions** tab for the **bDecide** widget. First, create a new OnClick interaction with a Set Variable Value action. The TARGET variable is **totalPoints**; keep “text” in the SET TO field; and enter the following into the VALUE field:

[[**totalPoints** + 1]]

This action (see Figure 28) adds 1 point to the **totalPoints** variable. The next step will display the current points in the **pointsBox** widget.

The screenshot shows the 'INTERACTIONS' tab in Axure RP. The widget 'bDecide' is selected. Under the 'OnClick' event, a 'Set Variable Value' action is configured. The 'TARGET' is 'totalPoints', the 'SET TO' is 'text', and the 'VALUE' is the formula '[[totalPoints + 1]]'. The formula is entered in a field with a function icon (fx) to its right. Below the formula field are 'Cancel' and 'OK' buttons. At the bottom of the panel is a '+ Add Variable' button.

STYLE	INTERACTIONS •	NOTES
bDecide		↑↑↑
▼ OnClick		IF
Set Variable Value		
totalPoints to "[[totalPoints + 1]]"		
TARGET	totalPoints	⌵
SET TO	text	⌵
VALUE	[[totalPoints + 1]] fx	
Cancel		OK
+ Add Variable		

Figure 28. The VALUE column has a formula that adds 1 point to the variable "totalPoints" in the TARGET column.

27. Add a Set Text action to **bDecide**, after the Set Variable Value action. The TARGET is the **pointsBox** widget; change the SET TO drop-down field to "value of variable"; and select **totalPoints** as the VARIABLE. This will update the **pointsBox** widget to display the participant's current points, whenever the participant earns a point under the VR3 schedule, as shown in Figure 29.



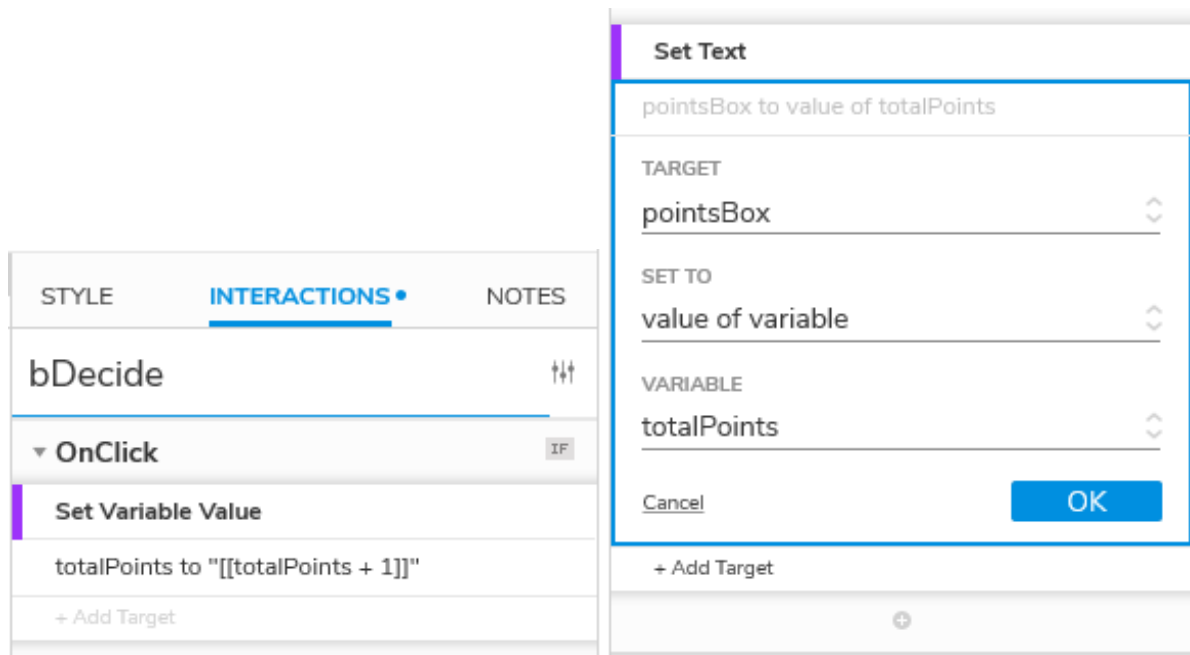


Figure 29. The settings highlighted by the blue square allow you to show the participant's current points each time he wins points.

28. Now, we will add the conditional statement (called a “case” in Axure© RP) to the OnClick interaction of **bDecide**. Right click the OnClick action and select “Add Case”, as shown in Figure 30. Name the case, **decideCase**, press enter, and then double-click on the interaction to open the Condition Builder window. The Condition Builder window can also be opened by clicking on the small, gray “IF” button to the right of the OnClick interaction.

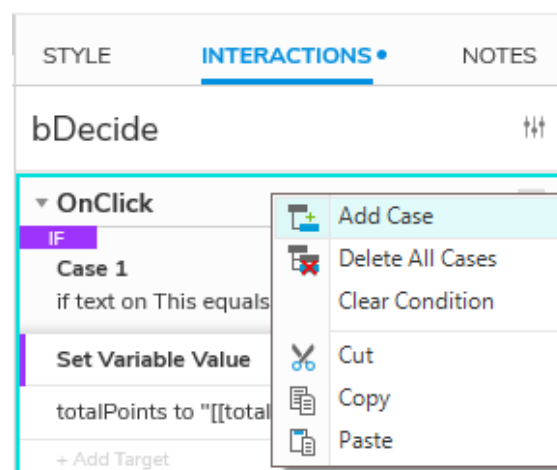


Figure 30. Context menu with the "Add case" option appears when you right-click.

29. In the Condition Builder window, we want the interaction to be carried out if the text on **bDecide** (i.e., “This” widget) equals 0; thus, click on the “+ Add Logic” button and keep “text on widget” in the first field, and “This” in the second field. The second field designates which widget will be focused on in the case—selecting “This” in this field is the same as selecting **bDecide** in this field because they both refer to the same widget. Next, keep “equals” in the third field. The fourth field should contain the select “text”. Finally, simply enter “0” in the last text field of the Condition Builder window. Your Condition Builder window should look like Figure 31. Click OK.

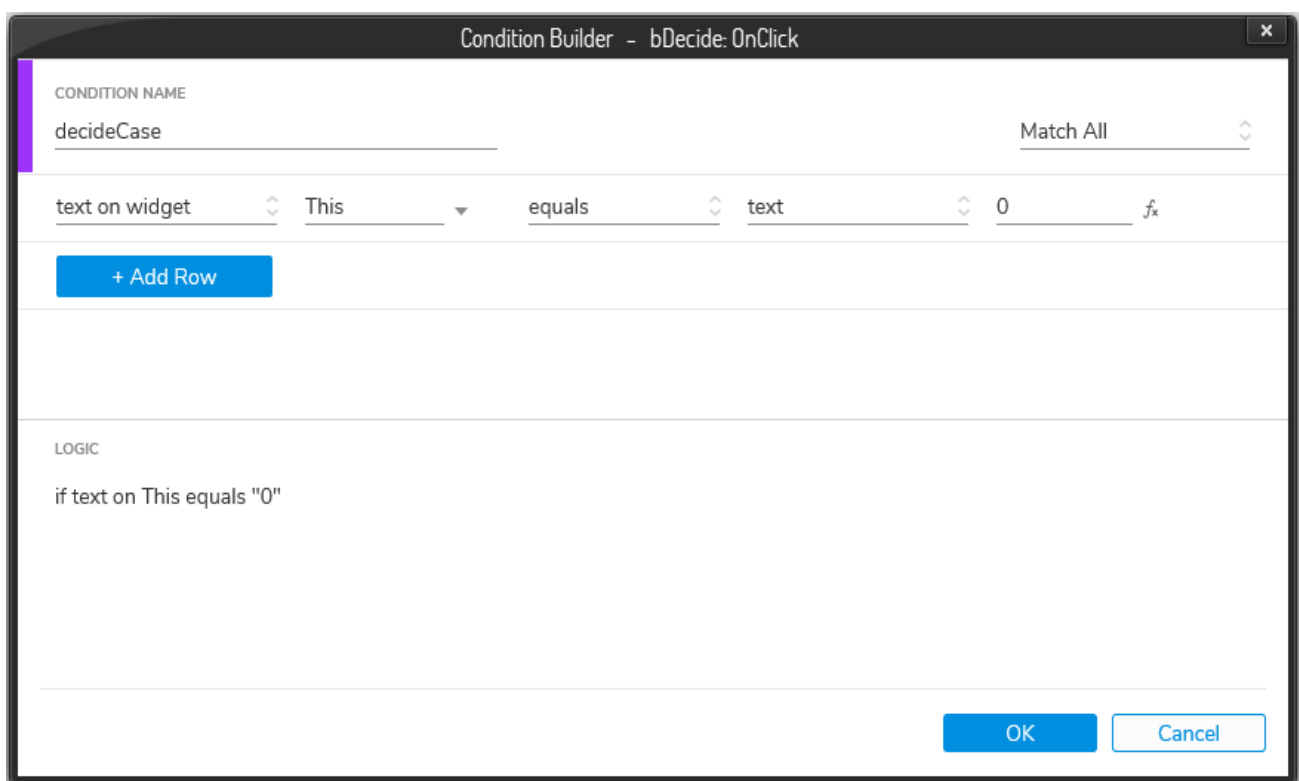


Figure 31. Configuring the condition so that the interaction is performed if the text in bDecide equals 0.

In the previous two sections, we added functionality to **aTarget** and **bTarget** by assigning schedules of reinforcement to both stimuli. Additionally, we used widget actions to trigger deciding events for our variable ratio schedule.

## Section V | Adding Intervals

There will be three, 10 s intervals in this demo experiment. These intervals were kept brief for the purposes of this demonstration—real experimental arrangements would include many more intervals. The participant’s total points will be displayed throughout the session, but only the experimenter will have access to the data for responding throughout each of the intervals.

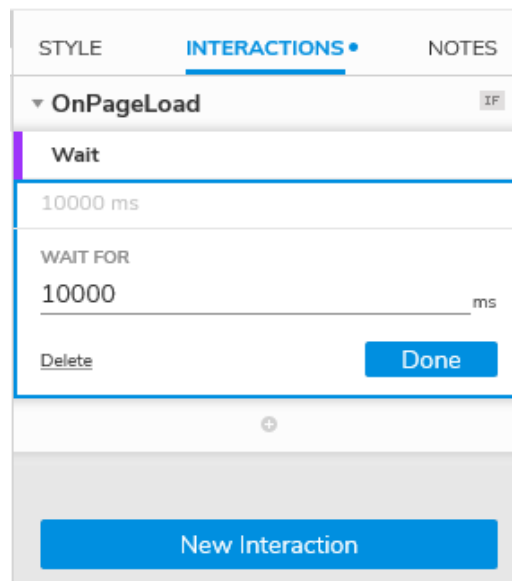
Objectives:

- Add three, 10 s intervals to the experiment
- Add six new global variables
- Store the number of responses in one of these new variables at the end of each interval (To do this, we will take a quick “snapshot” of our other global variables, **aResponses** and **bResponses**, by saving the values of each of these variables in one of the six new variables at the end of each interval)
- Reset the value of **aResponses** and **bResponses** to accurately reflect responding during each interval (If **aResponses** and **bResponses** are not reset, the six variables responsible for storing the number of responses will not accurately represent the number of occurrences per trial. It would not be necessary to reset these variable values if we were interested in analyzing these data using a cumulative record.)

30. To begin, we will have to add six new variables to store the data from each response at the end of each 10 s interval. Add the following global variables, and keep their default values blank: **a1**, **a2**, **a3**, **b1**, **b2**, and **b3**.

31. So far, we have been building interactions between widgets. Now, we will build interactions between the page and widgets. Click in any part of the white space in the **Work area** to deselect any widgets you might have selected. Navigate to the **Interactions tab** and add a new OnPageLoad interaction. The OnPageLoad interaction is triggered when the page loads. Select “Wait” as the

action for this interaction and enter “10000” in the WAIT FOR field, as shown in Figure 32. Click Done.



The screenshot shows the 'INTERACTIONS' tab in the Axure RP software. Under the 'OnPageLoad' interaction, a 'Wait' action is configured. The 'WAIT FOR' field is set to '10000 ms'. There are 'Delete' and 'Done' buttons at the bottom of the configuration area. A 'New Interaction' button is visible at the bottom of the panel.

Figure 32. The OnPageLoad interaction fires when the page loads.

The first 10 s interval has been set. Next, we need to store the number of responses, during each interval, in two of our six new global variables.

32. At the end of each 10 s interval, the data will be stored in two of the variables we created in Step 29. When the first interval ends, the number of responses on **aTarget** and **bTarget** will be stored in the variables **a1** and **b1**, respectively. Additionally, the second interval will store the number of responses on the buttons in the **a2** and **b2** variables. Finally, the third interval will use the variables **a3** and **b3**.

If you are not already on the **Interaction tab** for the page, begin by navigating to the **Interaction tab** for the page. Click the purple “Insert Action” button (“+”) under the same OnPageLoad interaction, and insert a Set Variable Value action. The TARGET will be **a1**. In the SET TO field, select “value of variable”. Select **aResponses** in the VARIABLE field, as shown in Figure 33. Click “OK”.

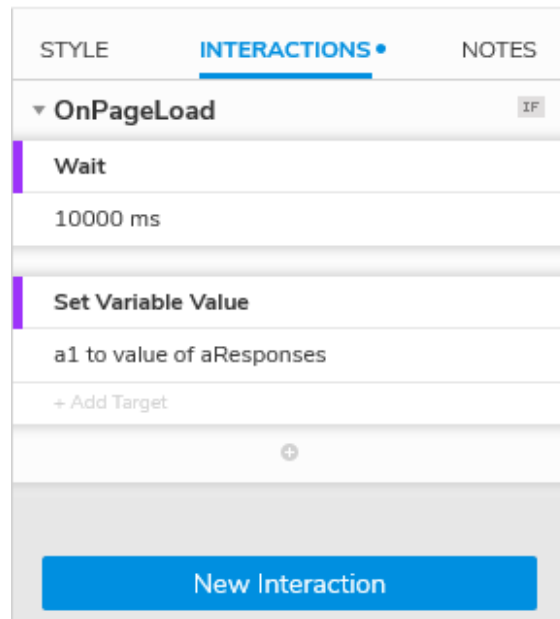


Figure 33. SET TO field allows selection of the "value of variable" option. Then the VARIABLE field allows the variable aResponses.

33. Next, we will repeat Step 32 for the **b1** variable. First, click the "+ Add Target" button under the Set Variable Value we just created. Select **b1** in the TARGET field; select "value of variable" in the SET TO field; finally, select **bResponses** in the VARIABLE field. Click "OK".

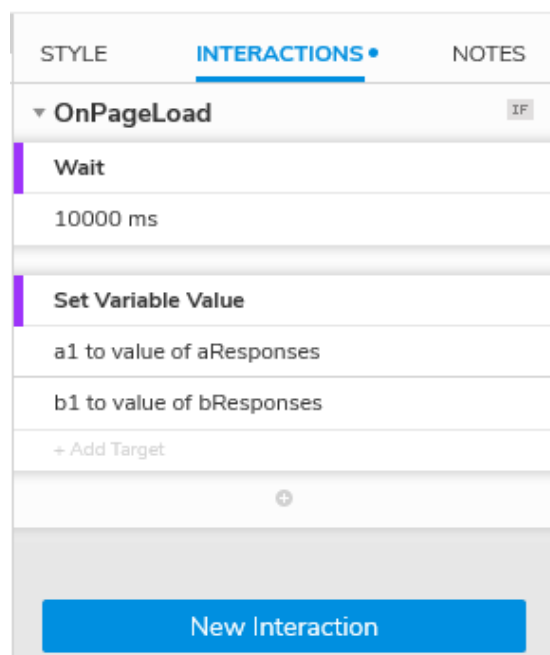


Figure 34. After repeating Step 32, the column "Set Variable Value" should look like this.

34. We need to reset the value of **aResponses** and **bResponses**. If we do not reset these variable values, the values from the second interval will include those responses in the first interval and the second interval, and so on for the third interval.

Add a new target under the column Set Variable Value actions we just created, by clicking the “+ Add Target” button. Select **aResponses** in the TARGET field; keep “text” in the SET TO field; finally, enter “0” into the VALUE field, as shown in Figure 35. Click the “OK” button.

The screenshot displays the 'INTERACTIONS' tab in Axure RP. Under the 'OnPageLoad' event, there is a 'Wait' action set to 10000 ms, followed by a 'Set Variable Value' action. This action is configured with two targets: 'a1 to value of aResponses' and 'b1 to value of aResponses'. A modal dialog is open for editing the first target, 'aResponses'. The dialog has three fields: 'TARGET' set to 'aResponses', 'SET TO' set to 'text', and 'VALUE' set to '0'. There are 'Cancel' and 'OK' buttons at the bottom of the dialog. Below the dialog, there is a '+ Add Variable' button and a 'New Interaction' button at the bottom of the panel.

Figure 35. Settings from Step 34.

35. Add an additional target. Repeat Step 34 but select **bResponses** in the target field.

The screenshot shows the 'INTERACTIONS' tab in the Axure RP software. Under the 'OnPageLoad' interaction, there are two actions listed: 'Wait' and 'Set Variable Value'. The 'Wait' action is set to '10000 ms'. The 'Set Variable Value' action has four targets: 'a1 to value of aResponses', 'b1 to value of bResponses', 'aResponses to "0"', and 'bResponses to "0"'. A '+ Add Target' button is visible below the targets. At the bottom of the panel is a 'New Interaction' button.

STYLE	INTERACTIONS •	NOTES
▼ OnPageLoad		IF
Wait		
10000 ms		
Set Variable Value		
a1 to value of aResponses		
b1 to value of bResponses		
aResponses to "0"		
bResponses to "0"		
+ Add Target		
New Interaction		

Figure 36. Settings after repeating Step 34.

36. Now, we will focus on the second interval. Under the same OnPageLoad interaction, add a new Wait action by clicking the purple “+” symbol. Enter “10000” into the WAIT FOR field, as shown in Figure 37, and click “OK”.

This screenshot is identical to Figure 36, showing the 'INTERACTIONS' tab with the 'OnPageLoad' interaction. It lists the 'Wait' action (10000 ms) and the 'Set Variable Value' action with its four targets. The '+ Add Target' button and the 'New Interaction' button are also present.

STYLE	INTERACTIONS •	NOTES
▼ OnPageLoad		IF
Wait		
10000 ms		
Set Variable Value		
a1 to value of aResponses		
b1 to value of bResponses		
aResponses to "0"		
bResponses to "0"		
+ Add Target		
New Interaction		

Figure 37. Settings after Step 36.

37. Insert another Set Variable Value action under the OnPageLoad interaction, by clicking the purple “+” symbol. Select **a2** in the TARGET field; “value of variable” in the SET TO field; and **aResponses** in the VARIABLE field, as shown in Figure 38. Click “OK”.

The screenshot displays the 'INTERACTIONS' tab in the Axure RP software. Under the 'OnPageLoad' interaction, there are three actions listed: 'Wait' (10000 ms), 'Set Variable Value', and another 'Wait' (10000 ms). The third 'Set Variable Value' action is selected, and its configuration is shown in a detailed view below the list. This view includes fields for 'TARGET' (set to 'a2'), 'SET TO' (set to 'value of variable'), and 'VARIABLE' (set to 'aResponses'). At the bottom of this configuration view are 'Cancel' and 'OK' buttons. Below the configuration view is a '+ Add Variable' button. At the very bottom of the interactions panel is a large blue button labeled 'New Interaction'.

STYLE	INTERACTIONS •	NOTES
▼ OnPageLoad IF		
Wait	10000 ms	
Set Variable Value	a1 to value of aResponses b1 to value of bResponses aResponses to "0" bResponses to "0" + Add Target	
Wait	10000 ms	
Set Variable Value	a2 to value of aResponses TARGET a2 SET TO value of variable VARIABLE aResponses Cancel OK + Add Variable	
New Interaction		

Figure 38. Settings from Step 37.



38. Repeat Step 37 for the **b2** variable by adding another target. Select **b2** in the TARGET field; “value of variable” in the SET TO field; and **bResponses** in the VARIABLE field. Click “OK”.
39. Next, repeat Steps 34 and 35 to reset the **aResponses** and **bResponses** before the next interval. So far, your page **Interactions tab** should look like Figure 39.

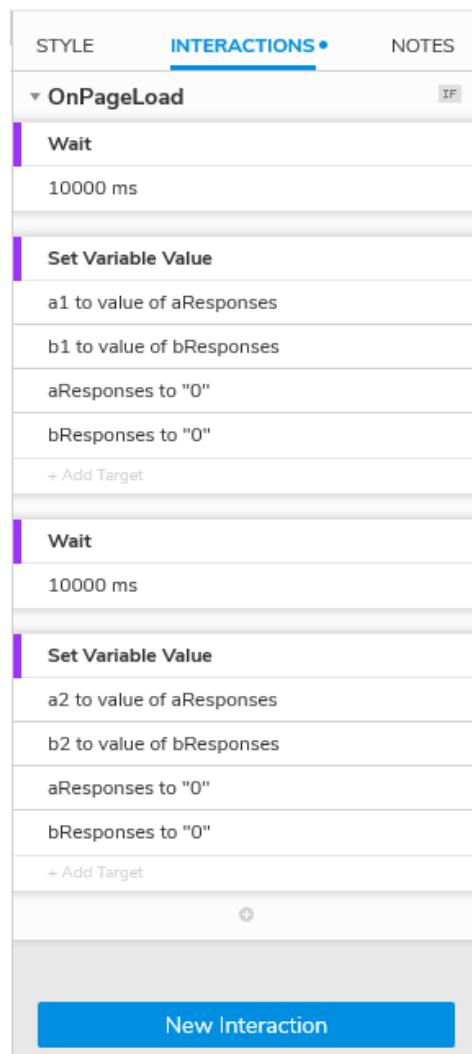


Figure 39. Settings after repeating Step 34 and Step 35.

40. Lastly, we will construct the components of the third interval. Begin by adding a Wait action (purple “+” symbol) under the same OnPageLoad interaction we have been using. Enter “10000” into the WAIT FOR field and click “OK”.

41. Add a Set Variable Value action with **a3** as the TARGET. Select “value of variable” in the SET TO field, and **aResponses** in the VARIABLE field. Click “OK”.
42. Click “+ Add Target” under this Set Variable Value action, and select **b3** as the TARGET; select “value of variable” in the SET TO field; and **bResponses** in the VARIABLE field. Click “OK”. It is not necessary to reset the **aResponses** and **bResponses** variables, because this is our last interval.

When this section is completed, your **Interactions tab** for the page should look like Figure 40.

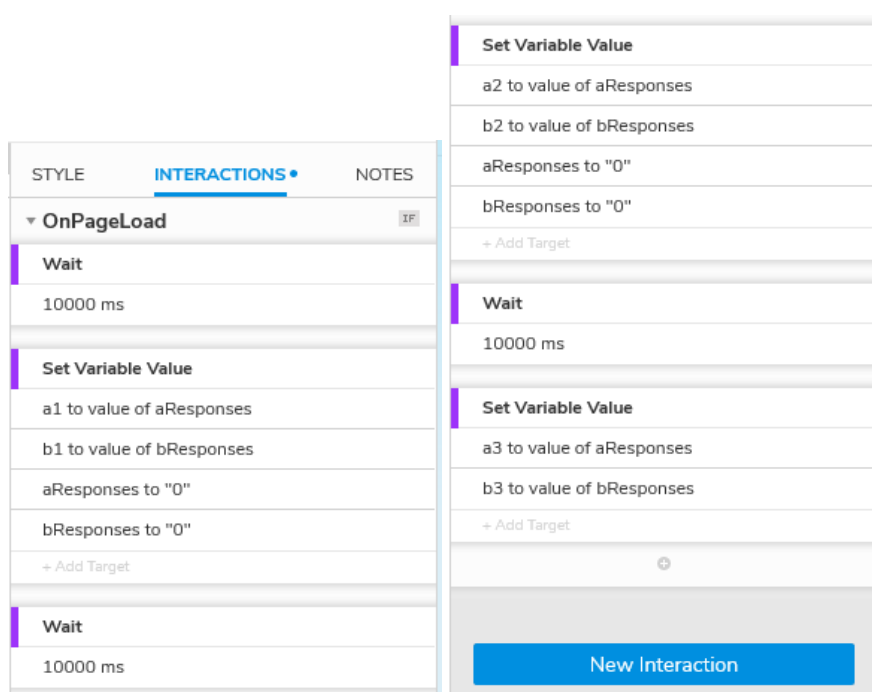


Figure 40. Settings after Step 40.

In this section, we created three, 10 s intervals. We also stored this interval values in Global Variables that we created. In the next section, you will learn how to output the data generated by participants during this demo experiment.

## Section VI | Data Output

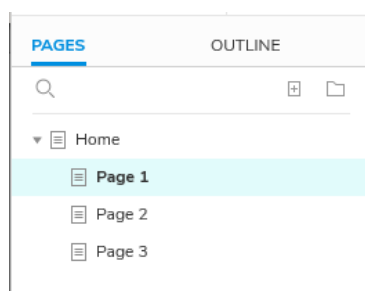
In this section, we will organize the data from all three intervals into separate text fields, which can be copied and pasted into Microsoft© Excel. Outputting the data to a table is not the

most ideal method for displaying the data if you have many variables because the table generated by Axure© RP cannot be pasted into Microsoft© Excel very efficiently. Using the method you are about to learn might still require you to organize the data after pasting it into Microsoft© Excel. Outputting the data as a string in a text box (with commas as delimiters) is the most ideal method for data output. This method allows you to copy and paste the string into Microsoft© Excel and then use Microsoft© Excel's built-in "Text-to-Columns" wizard to organize the variable values into individual columns.

Objectives:

- Create two text field widgets which will contain the variable values for each interval
- Create a button for displaying the variable values

43. Begin by navigating to the **Page pane** (1 in the guide). Double-click *Page 1* under the *Home* page (see Figure 41) and ensure that there are no widgets on this page (IMPORTANT: Make sure that you have *Page 1* selected before you delete any widgets!). If there are already widgets on *Page 1*, delete them all.



*Figure 41.* Double click on Page 1 below the Home page. If Page 1 is not listed, create another page by clicking the white square with a plus sign, to the right of the search (magnifying glass) button. Name this new page, "Page 1".

44. Next, go back to the *Home* page and navigate to the **Interactions tab** for the page (reminder: click on any white space in the **Work area** to deselect any widgets, and then go to the

**Interactions tab**). Create another action under the OnPageLoad interaction. Select the Open Link action. Select *Page 1* in the LINK TO field, as shown in Figure 42. Click “Done”.

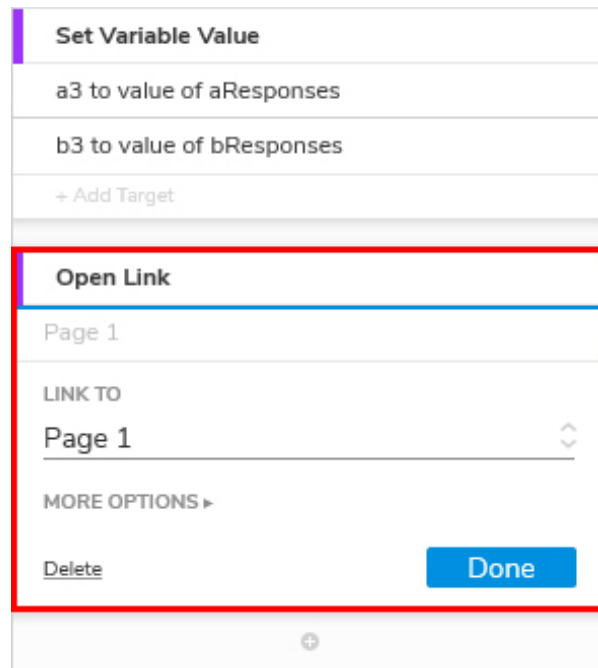


Figure 42. Step 44 settings highlighted by the red square.

45. Open *Page 1* again. Drag and drop two Text Field widgets into the **Work area**, as shown in Figure 43. Place these Text Fields in a visible location on the page, and ensure that one Text Field is above the other (see Figure 43). Name the top Text Field, **aResults**; and the bottom Text Field, **bResults**.

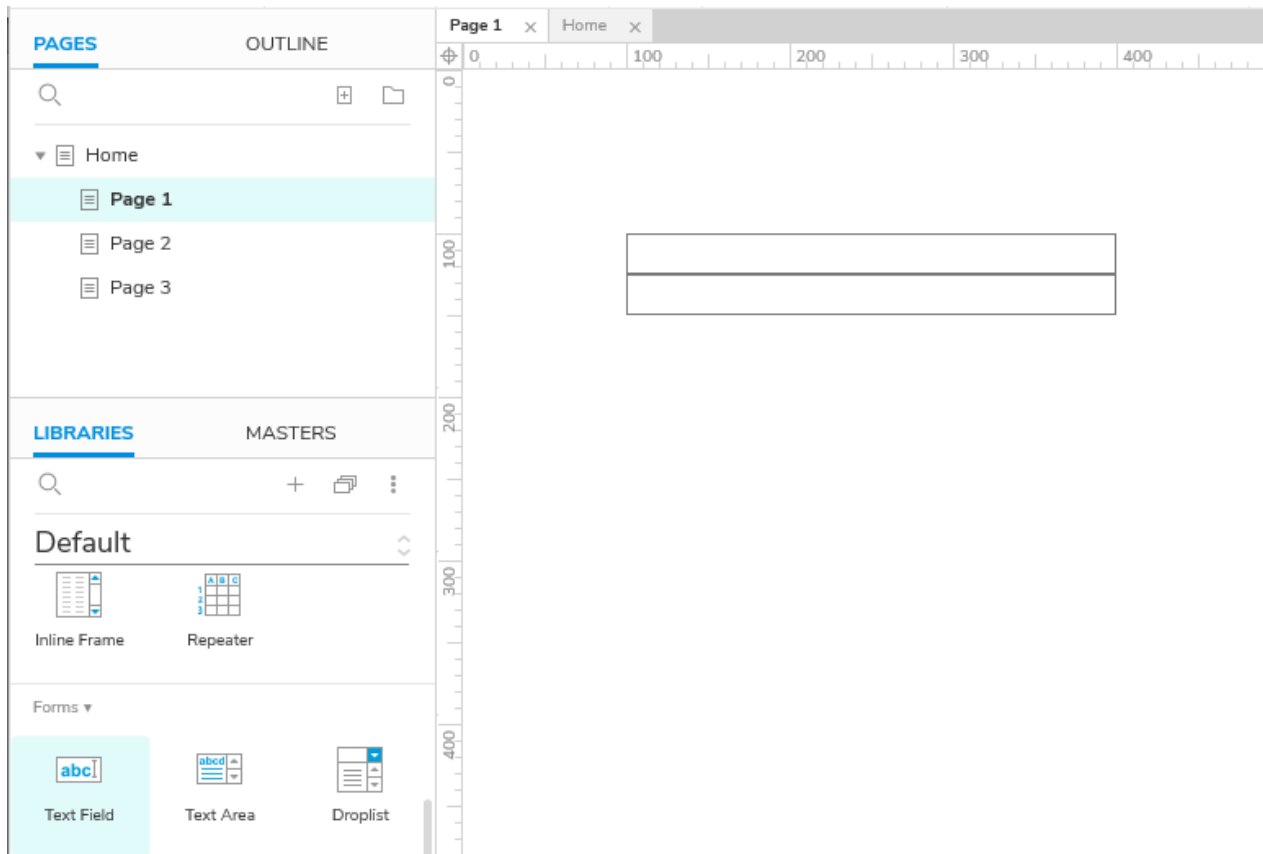


Figure 43. Two text fields (lower left corner) have been added to the Work area (right).

46. Add two label widgets to the immediate left of each of these Text Fields. Replace the text in these widgets with “a” and “b” to specify which target values they represent. Specifically, place the “a” label to the left of the **aResults** widget, and place the “b” label to the left of the **bResults** widget, as shown in Figure 44.

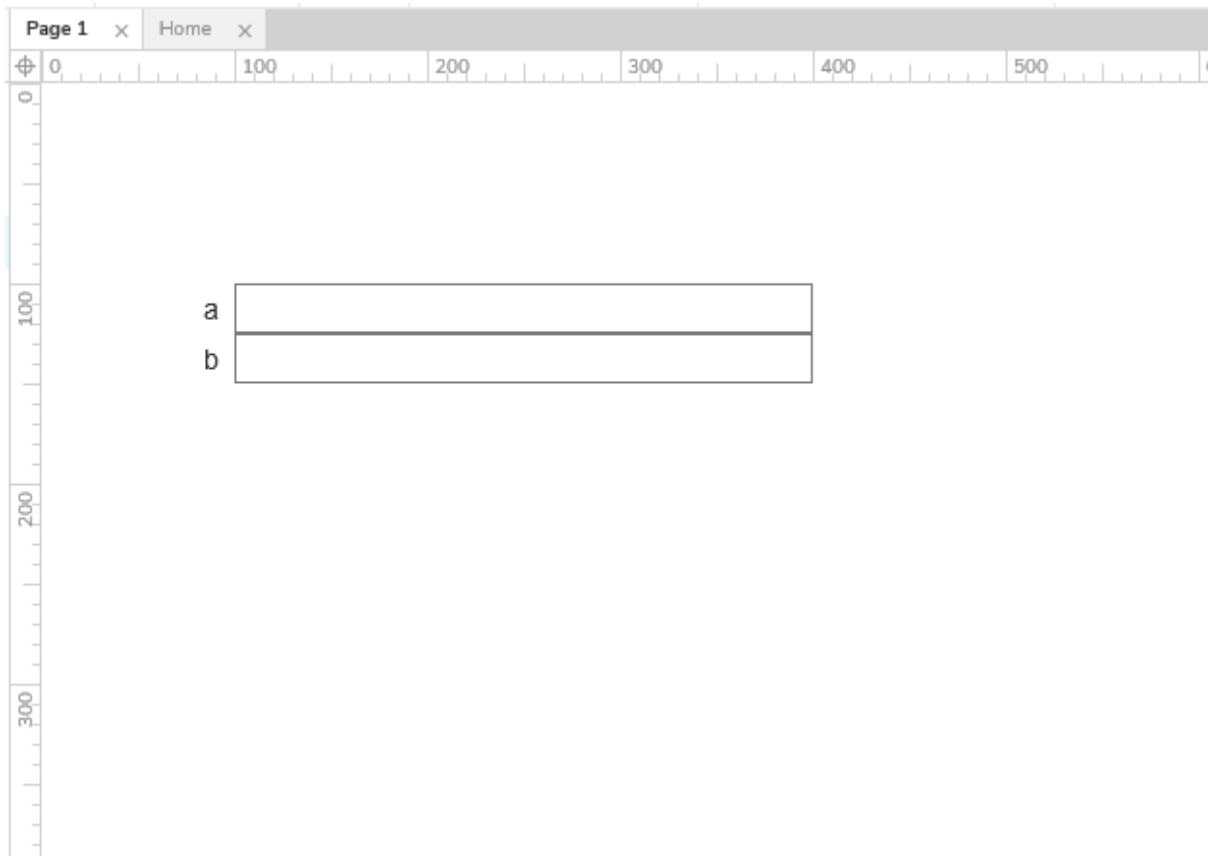


Figure 44. The label widgets do not need to be named, they are only added to help guide the experimenter to the relevant data.

47. Add a regular, white button widget to the page. Replace the text on the button with “Display Results”. Name this button widget, **resultsButton**. See Step 4 for a reminder of how to set widget names, as shown in Figure 45.

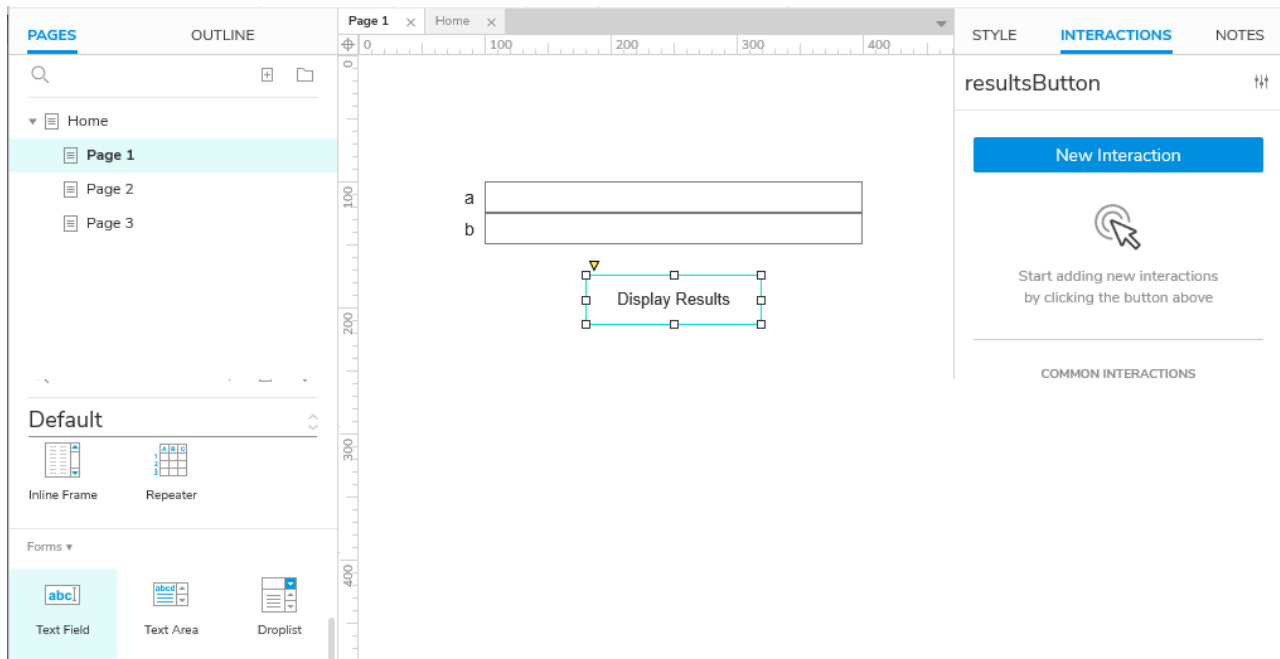


Figure 45. "Display Results" button added to the desktop.

48. With the **resultsButton** selected, navigate to the **Interactions** tab. Add a new OnClick interaction and choose the Set Text action. Select **aResults** as the TARGET widget; keep the SET TO field as "text"; and enter the following into the VALUE field:

[[a1]], [[a2]], [[a3]]

When using variables, place them in double brackets (i.e., [[variableName]]). The same applies for formulas and functions (e.g., "[[1 + 2]]"). What we have done, with the above bit of code, is tell Axure© RP to list the variables with commas and a space between each variable when the **resultsButton** has been clicked (see Figure 46). Click "OK" after you have entered this text.

The screenshot shows the 'Interactions' tab for a widget named 'resultsButton'. Under the 'OnClick' event, a 'Set Text' action is configured. The action is highlighted with a blue square. The configuration details are as follows:

Field	Value
Text	aResults to "[[a1]], [[a2]], [[a3]]"
TARGET	aResults
SET TO	text
VALUE	[[a1]], [[a2]], [[a3]]

Buttons for 'Cancel' and 'OK' are visible at the bottom of the configuration panel. Below the configuration panel is a '+ Add Target' button and a 'New Interaction' button at the very bottom.

Figure 46. Step 48 settings highlighted by the blue square.

49. With the **Interactions** tab still open for **resultsButton**, click the “+ Add Target” button under the Set Text action. Select **bResults** as the TARGET widget; keep “text” in the SET TO field; and enter the following into the VALUE field:

[[b1]], [[b2]], [[b3]]

Once you have entered this, click “OK”, as shown in Figure 47.



The screenshot shows the 'Interactions' panel in Axure RP. The 'resultsButton' is selected. Under the 'OnClick' event, the 'Set Text' action is highlighted with a blue square. The configuration for this action is as follows:

TARGET	SET TO	VALUE
bResults	text	[[b1]], [[b2]], [[b3]]

At the bottom of the panel, there is a '+ Add Target' button and a 'New Interaction' button.

Figure 47. Step 49 settings highlighted by the blue square.

The data from **aResults** and **bResults** can be easily copied into Microsoft© Excel. Microsoft© Excel comes with a wizard for converting text to columns. Ensure that you specify commas as delimiters within the Text-to-Columns wizard in Microsoft© Excel.

### Conclusion

In this tutorial, we created a simple choice experiment which consisted of two buttons and a points box. One button was under an FR1 schedule of reinforcement and the other was under a VR3 schedule. We also learned how to output participants' data in a way that makes it easy to paste into Microsoft© Excel.

One limitation of this specific approach is that researchers planning to implement a study made in Axure© RP will have to be physically present throughout the participant's engagement

(although, remote participation is discussed in the next paragraph). Additionally, Axure© RP global variables are stored in the URL field of the browser. This limitation affects almost all browsers, and most web developers stick to a limit of 2000 URL characters. Given all of this, Axure© RP's global variables must be used somewhat conservatively throughout the development stage—a limitation that researchers will probably not have to worry about, yet should still keep in mind.

Altogether, Axure© RP has far-reaching implications. Specifically, it has the capacity to facilitate the construction of scientific experiments, comprised of multiple web-based programming languages (e.g., JavaScript, CSS). The fact that these experiments consist of web-based code enables researchers to upload these experiments to personal or institutional websites and collect data from participants from almost anywhere in the world. For example, at the time of writing this chapter, I am recruiting participants and running my dissertation through a combination of an experiment I created in Axure© RP8, and Amazon's Mechanical Turk ("MTurk"). This combination made it somewhat easy for me to collect over 250 reliable data sets, in what took me a total of about 125 h of re-design and re-implementation. Unfortunately, the methodology for applying experiments made in Axure© RP to MTurk are too extensive to include in this chapter; although, I hope to write a second guide for that method soon.

### References

- Bancroft, S. L., & Bourret, J. C. (2008). Generating variable and random schedules of reinforcement using Microsoft Excel macros. *Journal of Applied Behavior Analysis*, 41(2), 227-235.

## Chapter 3

### Developing an application to register continuous responses with shiny package in R environment

Ricardo Fernandes Campos Júnior<sup>1</sup>  
*Universidade de São Paulo, SP, Brasil*

Julia Zanetti Rocca<sup>2</sup>  
*Universidade Federal de Mato Grosso, MT, Brasil*  
*Instituto Nacional de Ciência e Tecnologia sobre Comportamento, Cognição e Ensino*

**Translators<sup>3</sup>**  
Luiz Alexandre Freitas  
*Universidade Federal de Mato Grosso, MT, Brasil*

Théo P. Robinson  
*Florida Institute of Technology, FL, USA*

#### Abstract

To register a response directly is a frequent and essential part of the behavior analysts' job. There are computer programs to assist with this task already, but they are generally inflexible and require payment. Our objective in this chapter is to guide the reader in developing a basic application to measure behaviors and building cumulative frequency records during data post processing. The application will be created using the R programming language, through RStudio IDE with shiny, lubridate and ggplot2 extension packages.

---

1 Contact Ricardo Fernandes Campos Junior at [ricardofc@gmail.com](mailto:ricardofc@gmail.com). You will find Ricardo on GitHub at <https://github.com/RicardoFCJ>.

2 Contact Julia Zanetti Rocca at [profjuliarocca@gmail.com](mailto:profjuliarocca@gmail.com).

3 All filenames and names of variables in the example programming codes were translated to English to help readers. However, in the original code some names are in Brazilian Portuguese as seen in the first author's GitHub project for this chapter.

“Measurement (applying quantitative labels to describe and differentiate natural events) provides the basis for all scientific discoveries and for the development and successful application of technologies derived from those discoveries. Direct and frequent measurement provides the foundation for applied behavior analysis. Applied behavior analysts use measurement to detect and compare the effects of various environmental arrangements on the acquisition, maintenance, and generalization of socially significant behaviors.” (Cooper, Heron, & Heward, 2014, p. 93).

According to Cooper, Heron, and Heward (2014), direct and frequent measuring of the organism’s responses is a fundamental part of behavior analysts’ jobs, whether in academic or professional settings. In fact, the strategies for defining and identifying classes of behaviors to be measured, as well as the description of their topographies, have been continuously investigated by researchers in the field (Springer, Brown, & Duncan, 1981) and require specific and continuous training for practitioners.

However, besides the theoretical difficulties associated with measuring, to register responses accurately during intervention sessions is a challenge for practitioners (LeBlanc et al., 2016). Generally, practitioners and experimenters should manipulate contingencies and, simultaneously, keep records of frequency, duration and/or intensity of target behaviors, to be able to follow the development of the individuals’ repertoires. In order to do that, in some cases, sessions are video recorded and responses are registered later. This is an effective solution, but it considerably increases the amount of time and resources required to keep the analysis up to date.

Several practitioners and scholars have begun using software applications with the ability to record behavioral data through smartphones, tablets, or laptops (Mudford, Locke, & Jeffrey, 2011). These programs are advantageous because they facilitate faster and more effective registering. Importantly, they give practitioners the ability to construct graphs or tables in real-time, or immediately after the session. Many of these applications are available on the market, however, they usually require payment or have low flexibility with respect to the specific needs of each client, practitioner, and/or context.

With this in mind, the purpose of this chapter is to guide readers in the development of a basic application to register the occurrence of behaviors of participants, experimental subjects, or clients and, subsequently, plot a cumulative frequency graph for the session.

### 1. Prerequisites

Before we start, it is important that the reader has basic knowledge of R programming language and RStudio programming environment. Also, you should already have these programs installed on your computer, as well as some other packages, including: *shiny* (Chang et al., 2018), *lubridate* (Wickham & Grolemund, 2011) and *ggplot2* (Wickham, 2016). For a basic introduction to R see Campos Junior and Rocca (2018).

To be brief and go direct to the point, this chapter will not extensively define the features of each employed function. Therefore, it is important that the reader always checks the *help()* of those functions presented in this chapter.

### 2. Initial steps

*Shiny* applications are created directly in R. While they work, an R session is active, processing all actions inside the application. A *shiny* application has two main programming blocks: *ui* and *server*. All elements with which the user will interact are programmed inside *ui* (user interface). Inside *server*, all commands and actions made by the user in *ui* are evaluated and processed. In R, *server* and *ui* are interdependent, so it is impossible to run one without the existence of the other. However, in the programming phase, it is possible to build *ui* with an empty *server*, as is shown below. In this case, the components of the first will not work, but you will be able to see what your application looks like.

To get started, it is necessary to load a *shiny* package that has all basic functions for the application to run. Following that, *ui* and *server* variables should be declared. There are several ways to structure these variables, although to facilitate the understanding of the general operation of *shiny* applications, they will be structured in the most basic form, as in Example 1.

**Example 1**

```
library(shiny)
ui <- fluidPage()
server <- function(input, output, session) {}
shinyApp(ui, server)
```

See that the function `server` has some arguments (`input`, `output` and `session`), such arguments define variables that will be used inside the same function. These arguments are necessary so that the objects in `ui` can be used and manipulated inside the `server`.

In the programming code within this chapter, we will use a style of formatting named indentation. This style consists of aligning the code or some of its parts farther to the right or to the left in order to create a hierarchical structure. Indentation is used to better clarify the hierarchy of the code, and the algorithm structure. The lines below show one form of indentation to facilitate readers' understanding:

**Example 2**

```
1. Title
  1.1. Subtitle
  1.2. Other Subtitle
    1.2.1. Part of the previous subtitle
  1.3. One more subtitle
2. Etc
```

**3. User Interface (`ui`)**

In this phase, all `ui` elements of the application presented in this chapter will be programmed. All elements that the final user will interact with are programmed inside the user interface. Further, the structure of those elements will be programmed, such that they will be visually organized and displayed in a functional manner. An important feature of the `ui` is that all concatenated elements in it should be separated with single commas, just like the values in a vector in basic R coding. For this reason, the following examples that demonstrate various elements together will always be separated with single commas.

### 3.1. Layout

The proposed layout for the application that will be built here will follow the model shown in Figure 1. On the top, there will be a field for the experimenter to put his/her name and the name of the participant, as well as a clock that will be used to record the time of session, and buttons to start and pause the clock. The central part includes the buttons on which specific behaviors will be recorded during session. On the bottom, there will be buttons to export recorded data in csv format (datasheet) and to plot a cumulative frequency graph for the session.

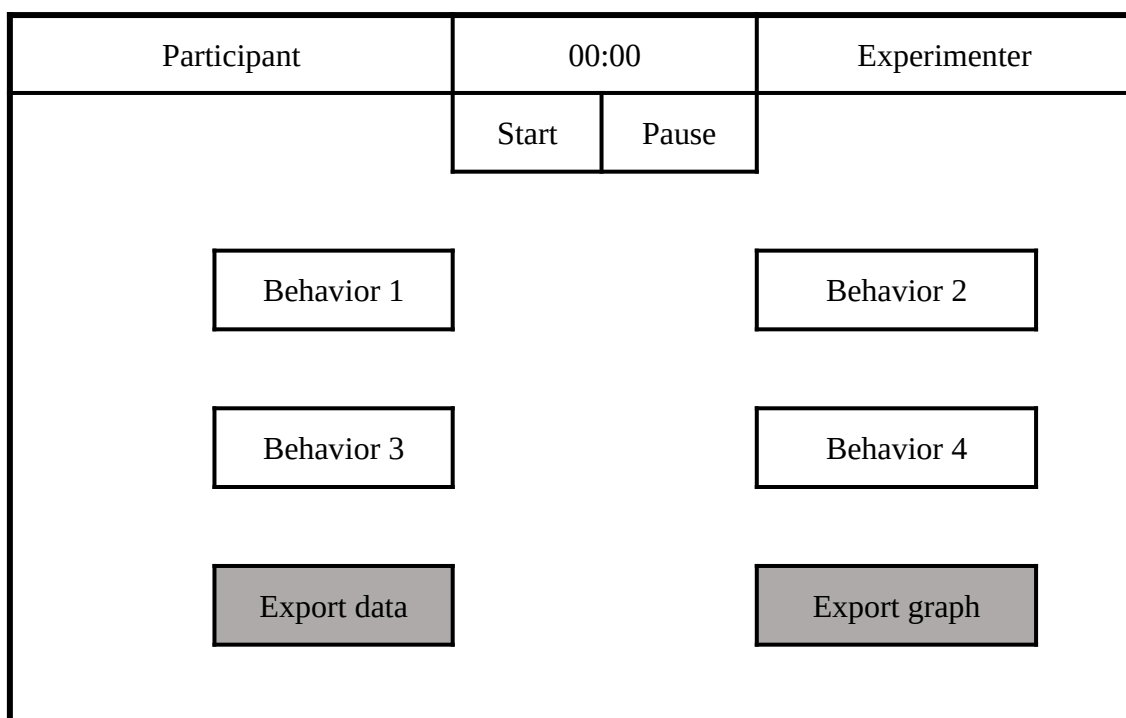


Figure 1. Initial draft of the application.

#### 3.1.1. Fluid rows (`fluidRow()` e `column()`)

The use of fluid rows is the simplest way to organize `ui` elements, and this is what we are doing in this chapter. It is built with the function `fluidRow()`. Setting a layout through this method is made by creating rows, which aim to ensure that elements of interface will be visually aligned. On the other hand, these rows are split in 12 columns, which are used to ensure the elements fill the right horizontal space in the layout. Thus, the layout created with

this method may have its basic structure conceived as an Excel spreadsheet, with 12 columns of horizontal space and as many rows as necessary.

One of the advantages of using `fluidRow()` to build layouts is that columns and rows always self-adjust to fill all spaces in the browser. Additionally, the function `column()` allows us to merge columns in one element so that it will fill the space corresponding to several columns. Also, this function can be used to subdivide one or many “cells” on the layout in its own set of rows and columns. The first argument of the function `column()` will always be the columns to be filled, followed by the elements that compose it. Another argument that could be used in this function is `align`, which serves to align the elements in specific positions inside the columns, taking the values *left*, *center*, and *right*. Figure 2 will help readers visualize the operation of these functions.

### Example 3

```
fluidRow(
  column(12,align="center",element1(),element2(),element3()),
  column(12,
    column(3,align="center",element1(),element2()),
    column(9,align="center",element3())),
  column(12,
    column(2,align="center",element1()),
    column(8,align="center",element2()),
    column(2,align="center",element3())),
)
```

Functions named as `elementX()` at Example 3 are for illustrative purposes only.

#### 3.1.2. Interactive elements

Figure 1 shows a draft on which the development of the application will be based. There are some textual elements in it, such as: the text box, where we insert names of participant and experimenter, and the line that displays session time. The other elements are buttons on which users will click in order to: start or pause the clock, record occurrence of behaviors, export session data, and export the graph. Next, we will go over these interactive elements, explaining how to create each one of them.



	1	2	3	4	5	6	7	8	9	10	11	12
1	ELEMENTO1				ELEMENTO2				ELEMENTO3			
2	ELEMENTO1    ELEMENTO2			ELEMENTO3								
3	ELEMENTO1		ELEMENTO2							ELEMENTO3		

Figure 2. Example of positions of elements using the functions *fluidrow* and *column* from Example 3. Numbers in the figure are only to illustrate each row and column position according to the programming in the example. Gray lines that separate rows and columns are also for illustrative purposes only.

### 3.1.2.1. Text input (*textInput()*)

In Figure 1, the type of interactive element used to make the input of the names of participant and experimenter will require the use of the function *textInput*. This function is often used when the user needs to insert some textual information freely – in this case, the names of the participant and the experimenter. This function has three main arguments, *inputId*, *label* and *value*. The argument *inputId* is used to create a unique identification for the element, such identification will be used to get the data and manipulate the elements inside the *server*. For this reason, that argument is used in the majority of the *ui* elements of *shiny*. The argument *label* is used to give the interactive element a name, so users can easily identify what it is for. In the example we used in the draft (Example 3), the argument *label* is empty. Finally, the other argument is *value*, and it is used to tell the element its initial value. In the example on Figure 1, the value of *value* is “Participant”. Note that if the word “Participant” had been used in the argument *label* instead of *value*, this word would be on top of the textbox, not inside. In Figure 3, we filled *label* in the element of participant, and *value* is empty (“”). Differently, we filled *value* of experimenter with “Experimenter” and *label* is empty, as an example for readers.

**Example 4**

```
textInput(inputId="participant",label="Participant",value=""),
textInput(inputId="experimenter",label="",value="Experimenter")
```

The result of the elements programmed with the codes in Example 4 can be viewed in Figure 3. However, some programming codes were omitted in Example 4 because they were either previously explained or will be explained later in this chapter.

Participant	00:00	Experimenter
<input type="text"/>		<input type="text"/>

Figure 3. Result of the text boxes programmed in Example 4.

**3.1.2.2. Text output (*textOutput()*)**

The main argument for *textOutput* is *inputId*. The reason for that is because size, color, and content will be manipulated inside the *server* of the application. In Figure 1, the only text output is the clock, which is initiated with the value “00:00” and modified after the button “Start” is clicked. Figure 3 shows the clock in its initial value, a value that is generated inside the server. Example 5 shows the code that will generate the text output that will generate the clock. In the same way, a warning is displayed when one of the behavior buttons is pushed, indicating that a behavior was recorded. The text output for both, and how they relate to the server, will be explained further in the chapter.

**Example 5**

```
textOutput(inputId="clock")
```

The result of the code in Example 5 is displayed in Figure 4, but some programming codes were omitted because they have been previously explained or will be explained later. The code that controls the clock and generated the numbers 0:16 will also be explained below.

Participant	00:16	Experimenter
<input type="text"/>		<input type="text"/>

Figure 4. The central part of the figure displays the output for the element programmed in Example 5.

### 3.1.2.3. Action buttons (`actionButton()`)

Considering only the initial outline proposed for the application, the sole interactive element of the *ui* that we need to cover is action buttons. These trigger actions in the server that will make the clock start or pause, record target behaviors, export the spreadsheet, or plot the graph. `ActionButton` main arguments are *inputId* and *label*. They have the same function as the previously discussed arguments – they create a unique identification and denote what the interactive element is for. However, unlike `textInput`, the *label* of `actionButton` will always be inside the created button. Also, we are able to modify some basic features of style using CSS programming code inside the `actionButton`. In Example 6, the argument *style* will be used to modify the size of the buttons.

#### Example 6

```
actionButton("start","Start"),actionButton("pause","Pause"),
actionButton("behav1","Behavior 1", style="padding:15px 35px; fontsize:100%"),
actionButton("behav2","Behavior 2", style="padding:15px 35px; fontsize:100%"),
actionButton("behav3","Behavior 3", style="padding:15px 35px; fontsize:100%"),
actionButton("behav4","Behavior 4", style="padding:15px 35px; fontsize:100%"),
actionButton("expdata","Export data"),
actionButton("expgraph","Export graph")
```

The result of the element programmed with the code in Example 6 is displayed in Figure 5.

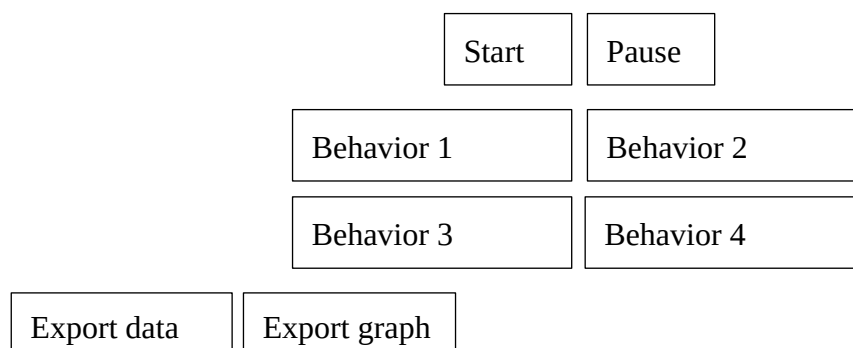


Figure 5. Preview of the outcome of the programming code that created buttons, as shown in Example 6.

### 3.2. Final interface

Now that the basic elements of the *ui* and how to organize them in this project have been presented, it is possible to put this all together to get the desired result. The code below shows the final code of the *ui*, and Figure 6 displays the final result. Note that, despite that the clock in the picture is functional, it is still necessary to program the elements in the *server* of the application to make it operational.

```
ui <- fluidPage(
  fluidRow(
    column(12, column(4, textInput("participant", "", value="Participant")),
      column(4, align="center", h2(textOutput("clock"))),

    column(4, textInput("experimenter", "", value="Experimenter")),
      column(12, column(4,
        column(4, align="center", actionButton("start", "Start"),

        actionButton("pause", "Pause")),
        column(4)),
        column(12, align="center", actionButton("behav1", "Behavior 1",
style='padding:15px 35px; font-size:100%'),
        actionButton("behav2", "Behavior 2", style='padding:15px
35px;
font-size:100%')),
        column(12, align="center", actionButton("behav3", " Behavior 3",
style='padding:15px 35px; font-size:100%'),
        actionButton("behav4", " Behavior 4", style='padding:15px
35px;
font-size:100%')),
        column(12, align="center", textOutput("warning")),
        column(12, actionButton("expdata", "Export data"),
          actionButton("expgraph", "Export graph"))
      )
    )
)
```

### 4. Server

At this step, the elements that respond to the interactions of the user inside the *ui* will be programmed. In the *server*, programming blocks will be built the same way a script in R is written. However, the blocks are read selectively, depending on the interaction of the user or the intention of the developer. Some parts will run only once, others will run dozens of times.

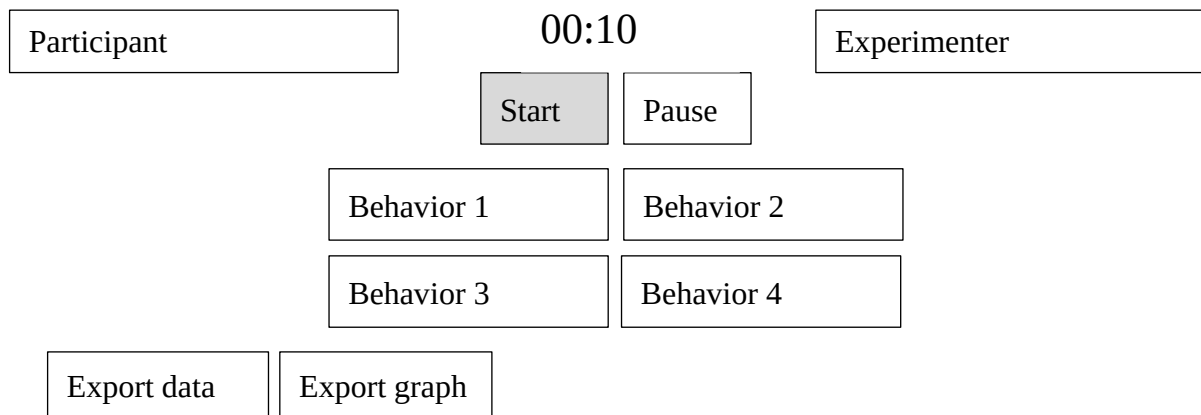


Figure 6. Final interface for the application. The button “Start” was clicked for the clock to appear.

The general operation of the server of the application shown here will occur as follows:

- 1) By starting the application, the clock is put in its initial state, that is “00:00”. Additionally, a table will be created to record the name of experimenter, participant, and what behavior occurred, and when that happened.
- 2) By clicking the “Start” button, the clock will start running and, every 1 s, the clock will be updated to show the current duration of the session.
- 3) By clicking on any of the behavior buttons, a warning message will be sent to the user interface confirming that a behavior was recorded at that moment. In addition to that, an instance of the behavior and the moment it occurred is recorded on the table.
- 4) By clicking “Pause”, the end of a session is recorded (so that the graph will be created with the correct limit) and the clock goes back to “00:00”.
- 5) By clicking “Export data”, a “.csv” file is created with the name “Experimenter-Participant.csv” that will be saved in the same folder as the application. This file contains all recorded behaviors and the time they occurred.
- 6) By clicking “Export graph”, one or several “.png” files with the name “Behavior-N.png” will be created. N is the number of the behavior (1, 2, 3, or 4), and it will be saved in the same folder as the application. This file is a cumulative frequency record containing each behavior in that session.
- 7) Finally, in case the application is closed and there were unsaved recorded behaviors, the application will save (preventively) by itself a file named “Experimenter-ParticipantAutoSaved.csv”.

Next, we will present the server elements that will be used in the application.

#### 4.1 Arguments of the Server

In section 2, the function `server` was presented, and it had three arguments: *input*, *output* and *session*. These arguments correspond to objects inside the server, which can be used to access other specific objects in it. These arguments have the list type structure,

because inside them there might be several objects of different types. So, when we want to access server objects by their arguments, it is necessary that we use the argument with the same name, separated by the symbol \$.

#### 4.1.1 Session

In coding, functions such as *callback* are triggered after a specific event. The argument *session* in server has some functions that allow *callbacks*, for instance, when an application is initiated or when it is terminated. Example 7 shows how to do a task when the application is terminated.

##### Example 7

```
session$onSessionEnded(function(), {})
```

#### 4.1.2. Input

In user interface, we have used several elements that allow the user to insert information – name of participant and name of experimenter, for example. That information has to be accessed in the server to be used. Still, inside *ui*, we created unique names for interactive elements. Those names will be used here to grant access to the elements and retrieve necessary information from them. The access is made with the argument *input*. Example 8 shows how we access the name of the participant and designate a variable to store it. Because the only way to have access to the values of interactive elements is from inside the reactive expressions (discussed later), the following example shows how to access an interactive object, but it will return an error if you run it by itself.

##### Example 8

```
part = input$participant
```

#### 4.1.3. Output

If *input* gets information from users, *output* gives it to them. Inside the application we are building in this chapter, there are two types of output information: the clock and the warning message indicating that the behavior was recorded. The output is made through the element *textOutput()* at user interface. To return the information from this element to the user,

we use the function `renderText()`. Example 9 shows how this information is returned. For now, we are returning only the first characters of the clock. That is, the line in Example 8 could be part of the function in Example 7 to set the clock to its initial state.

### Example 9

```
output$clock=renderText({paste("00:00")})
```

## 4.2. Reactive expressions

In R scripting, codes run linearly from top to bottom in the algorithm – conditionally to cycles. In the server of a *shiny* application, it is necessary that codes are selectively executed in response to specific events, for instance, by clicking a button or changing a text input. Additionally, it is required that scripts executed in response to these events are capable of retrieving updated values of data provided by users. That way, for the actions to be executed in the right order, reactive expressions are used. A reactive expression is a block of code that responds to specific changes, so that the programmer can control what changes will trigger the execution or re-execution of a script. In the application presented in this chapter, we use four reactive expressions: (1) one that initiates the clock with the start button; (2) one that stops the clock with the pause button; (3) one that exports data with the button “Export data” and; (4) one that builds a graph when “Export graph” is clicked. All these expressions use only one form of triggering the re-execution of programming blocks.

Example 10 uses the function `observeEvent()`. The first argument in this function tells what change triggers the execution or re-execution of the block of code. In this case, we use the start button as an example, but we could use any variable as an argument, such as text inputs.

### Example 10

```
observeEvent(input$start, {})
```

In `observeEvent()`, no object is directly returned to be used or changed later. Although, if it is necessary to receive an object, the functions `reactive()` and `eventReactive()`

are available. The former does not require arguments, it is often used to access information from interactive objects in non-reactive blocks. The latter works as `observeEvent()`, but returns an object. For instance, if we want to display the text “Participant’s name is Joe” in a `textOutput()` called “name” every time it is modified, the following three examples (Examples 11, 12, 13) show how to do the task using each one of the reactive expression functions.

#### Example 11

```
observeEvent(input$participant,{
  output$name=renderText({
    paste0("Participant's name is ", input$participant, ".")
  })
})
```

#### Example 12

```
participant = reactive({
  paste0("Participant's name is ", input$participant, ".")
})
output$name=renderText({ participant() })
```

#### Example 13

```
participant = eventReactive(input$participant,{
  paste0("Participant's name is ", input$participant, ".")
})
output$name=renderText({ participant() })
```

### 4.3 Global variables

For this application to work as desired, some variables will be needed to help control its operation. For the existence of these variables to make sense for readers, although they are not part of server (or user interface), they will be presented here. They are the following:

1. A *boolean* type variable (true or false) that indicates if the clock is running. This indicator will help us to decide what will be shown on the clock: either the elapsed time, or its initial state ("00:00").
2. A “time” type variable that indicates the time when the clock was initiated. It will be useful for calculating how much time has passed, and for showing that time on the clock.
3. A *data frame* with four columns to be filled with the occurrence of behaviors.
4. A function to calculate elapsed time since the “Start” button was clicked and format this time in “minutes:seconds”.

Variables 1 (start), 2 (stTime) and 3 (data) are declared as a list of reactive values **outside** of the server and user interface, as in Example 14.



**Example 14**

```
vars=reactiveValues(
  start=F,
  stTime=0,
  data=as.data.frame(matrix(ncol=4, nrow=0,
                           dimnames= list(NULL, c("Experimenter",
                                                    "Participant",
                                                    "Behavior",
                                                    "Occurrence")
                           ), stringsAsFactors = F)
)
ui = fluidrow({ ... })
server = function(input,output,session){ ... }
```

Although the application we are developing is intended to be used by a single user, in theory it could be used simultaneously by multiple users. Because we are using reactive values, instead of a common type list variable, the program allows multiple users to have access to the same clock, and use the same table to record behaviors. If our purpose was to create a table with its own clock for each user, we would have declared those variables in a list **inside** the server, and replaced *reactiveValues* for *list* as well.

Also, Variable 4 must be declared outside of the server and user interface, as in Example 15.

**Example 15**

```
format.timediff <- function(start_time) {
  diff = as.numeric(difftime(Sys.time(), start_time, units="mins"))
  hr <- diff/%60
  min <- floor(diff - hr * 60)
  min=ifelse(nchar(min)<2,paste0(0,min),min)
  sec <- round(diff%1 * 60)
  sec=ifelse(nchar(sec)<2,paste0(0,sec),sec)
  return(paste(min,sec,sep=':'))
}
ui = fluidrow({ ... })
server = function(input,output,session){ ... }
```

Because at the beginning of this chapter we assumed readers had basic knowledge of R, this function will not be detailed.

**4.4. Server blocks**

In section 4. (*Server*) we introduced the tasks that will be executed separately by the application. From now on the blocks of code for each task will be explained.

#### 4.4.1. States of the clock (Items 1 and 2)

When the application is started, the clock should have the state “00:00”. After a click on the “Start” button, its state should change every second to imply the passage of time. Since the block of code that tells in what state the clock is is the same, either stopped or running, we will introduce first the block of code activated by clicking “Start”.

##### Example 16

```
1. observeEvent(input$start,{
2.     vars$stTime=Sys.time()
3.     vars$start=T
4. })
```

1. The click on the button “Start”, which *inputID* is “Start”, triggers the execution of *observeEvent()* in line 1.
2. Line 2 records the exact time when the button was clicked in the global variable *vars\$stTime* by requesting date and time with the function *Sys.time()*.
3. Line 3 records that the clock is running in the global variable *vars\$start*. Using this information, the *script* below (Example 17) can tell the clock what state should be presented in the block below.

##### Example 17

```
1. output$clock=renderText({
2.     invalidateLater(1000,session)
3.     if(vars$start){
4.         paste(format.timediff(vars$stTime))
5.     }else{
6.         paste("00:00")
7.     }
8. })
```

1. Every 1 s, the text output for the clock is updated. This task is executed with the function *invalidateLater()* in line 2. This function “invalidates” the block of code every x ms. In *shiny*, when a block is invalidated, it is signaled to be re-executed.
2. In line 3, the function *if()* checks if the clock is running in the global variable *vars\$start*.
3. If the clock is running, line 4 sets the difference between the time since the clock was initiated and the current time, and records it in the variable *vars\$stTime*.
4. If the clock is not running, line 6 will set the time in the clock as “00:00”.

#### 4.4.2. Recording behaviors occurrence (Item 3)

When one of the behavior buttons is clicked, a record should be added to the record table, and a warning should be displayed telling the user it happened. These tasks are executed by the clock in Example 18.

**Example 18**

```

1. observeEvent(input$behav1,{
2.   occurrence=format.timediff(vars$stTime)
3.   output$warn=renderText({paste("Behavior 1 recorded in", occurrence)})
4.   vars$data[nrow(vars$data)+1,]=c(input$experimenter, input$participant,
5.   occurrence)
5. })

```

1. Clicking the button “Behavior 1”, which *inputID* is “behav1”, triggers the execution of *observeEvent()* in line 1.
2. Line 2 records the time of the click.
3. Line 3 sends information that a click was registered to the user and the time of the record.
4. Line 4 adds a record to the table, with information of the experimenter, participant, what behavior was clicked, and the time of occurrence.

Note that all behavior buttons work exactly the same way as button 1, the differences are the name of the button in the argument of *observeEvent()*, the text displayed to the user with the number of behaviors, and the record telling which behavior was clicked in line 4. For this reason, the code for the other buttons will only be shown in the section presenting the final result for the server.

**4.4.3. End of session (Item 4)**

When the pause button is clicked, the application has to stop the running clock and record the end of the session in the record table. The block below (Example 19) introduces how this task is executed.

**Example 19**

```

1. observeEvent(input$pause,{
2.   vars$start=F
3.   end=format.timediff(vars$stTime)
4.   vars$data[nrow(vars$data)+1,]=c(input$experimenter,
5.   input$participant, 0, fim)
5. })

```

1. Clicking the button “Pause”, which *inputID* is “pause”, triggers the execution of *observeEvent()* in line 1.
2. Line 2 changes the value of the global variable *vars\$start* indicating that the clock should stop and reset to “00:00”.
3. Line 3 records the final time of session in a variable.
4. Line 4 records the end of session in the record table *vars\$data* using the “behavior” 0 as a mark.

#### 4.4.4. Export data (Item 5)

At the end of the session, the experimenter has to export the data recorded by the application. This task is initiated with a click on the button “Export data” and executed by the block below (Example 20).

##### Example 20

```
1. observeEvent(input$expdata,{
2.   write.csv(vars$data, paste0(paste(input$experimenter,
                                     input$participant, sep="-"), ".csv"))
3. })
```

1. Clicking the button “Export data”, which *inputID* is “expdata”, triggers the execution of *observeEvent()* in line 1.
2. Line 2 saves the data with the name of experimenter and participant separated by “-“, and the extension “.csv”, in the same folder as the application.

It is important to remember that the experimenter should move this file to another folder to prevent the overwriting of this file with data from a new session.

#### 4.4.5. Export graph (Item 5)

At the end of the session, the experimenter can export a graph for each recorded behavior. This task is initiated with a click on the button “Export graph”, and the execution of the block below. Figure 7 displays an example of graph created after this action.

##### Example 21

```
1. observeEvent(input$expgraph,{
2.   datum=vars$data
3.   end=datum$Occurrence[datum$Behavior==0]
4.   datum=datum[datum$Behavior!=0,]
5.   for(i in unique(datum$Behavior)){
6.     datum=rbind(c(input$experimenter,
                     input$participant,
                     i,
                     "00:00"),
                   datum)
7.     ggplot(datum [datum $Behavior==i,], aes(x=ms(Occurrence),
        y=cumsum(grepl(i, Behavior))-1)) +
8.     geom_line() + geom_point()+xlim(0,ms(end))+ ylab("Frequency")
        +xlab("Session time")
9.     ggsave(paste0("Behavior -",i,".png"))
10.   }
11.   output$warn=renderText({paste("Graphs were saved in the folder.")})
12. }
```

1. Clicking on the button “Export graph”, which *inputID* is “expgraph”, result in the execution of *observeEvent()* in line 1.
2. Line 2 saves the data from a global variable in a local variable.

3. Line 3 records the final state of the clock in a local variable, for this information to be excluded from the data that will be plotted, but used as a limit on the graph.
4. Line 4 removes the registry of the final state of the clock from the data to be plotted.
5. Line 5 builds a cycle that will create and save a graph for each recorded behavior.
6. Line 6 adds a line on the top of the table and records the beginning of the session as “00:00”, for the initial line on the graph to start on this point.
7. Line 7 creates the initial parameters of the graph, indicating what behavior will be used in the present cycle. It converts the column “Occurrence” in elapsed seconds and uses the column “Behavior” to create a cumulative summation variable, in order to produce the cumulative frequency graph of the recorded behaviors.
8. Line 8 adds the other graphic parameters, such as: the type of graph, x axis limits, and labels of both axes.
9. Line 9 saves the graph in a file containing the label of the behavior saved in that cycle.
10. Line 11 sends a warning to the user indicating that the file (or files) is saved.

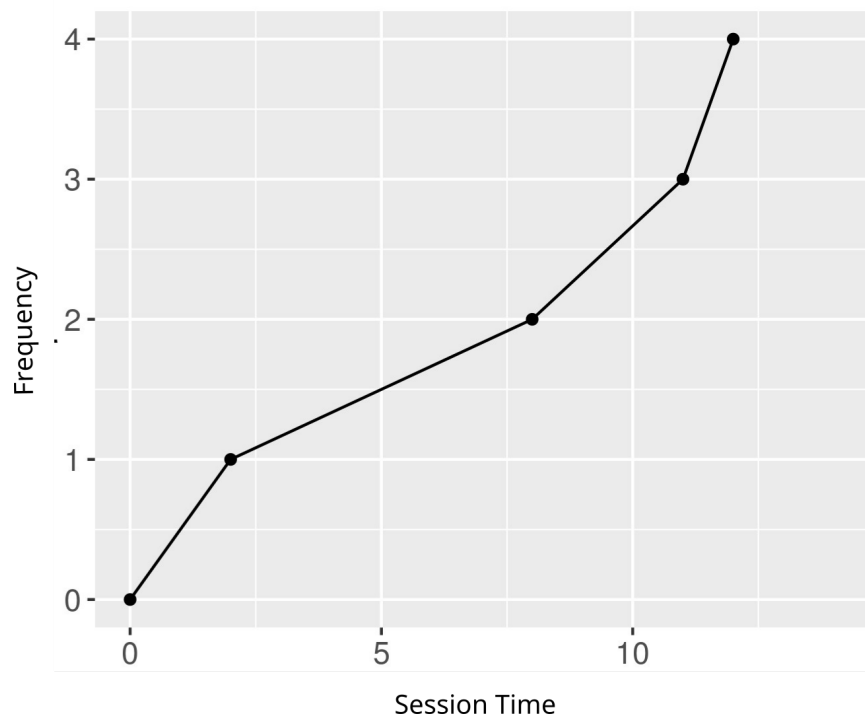


Figure 7. Cumulative frequency graph generated by clicking the button “Export graph”.

#### 4.4.6. Automatically saving the data (Item 6)

It could happen accidentally that the experimenter registers a session, shuts the application down, and forgets to save the data. To prevent the loss of data, the application will check for a file saved in the application folder under the name of experimenter and participant. In case there is not, the application will automatically save the data recorded until that moment. This task is done with the code in Example 22.

**Example 22**

```

1.   session$onSessionEnded(function(){
2.     experimenter=isolate(input$experimenter)
3.     participant=isolate(input$participant)
4.     filename=paste0(paste(experimenter,participant,sep="-"), ".csv")
5.     autoSave.name=paste0(paste(experimenter,
                                participant, sep="-"),
                           "AutoSave.csv")
6.     if(dim(isolate(vars$data))[1]>0){
7.       if(!file.exists(filename)){
8.         write.csv(isolate(vars$data), autoSave.name)
9.       }
10.    }
11.  })

```

1. Shutting down the application makes it run the function `onSessionEnded()` in line 1.
2. Lines 2 and 3 save the names of experimenter and participant in different local variables each. Because these variables are reactive, that is, they may be modified, they must be isolated with the function `isolate()`. Isolating these variables prevents them from being re-evaluated during the execution of `onSessionEnded()` and allows reactive variables to be used in nonreactive contexts, such as `onSessionEnded()`.
3. Lines 4 and 5 save a filename to be checked for its existence in the folder and a filename to be created in case none exists in the folder, respectively.
4. Line 6 checks if there is any information recorded in the data table. If true, the existence of a saved file with this information is verified.
5. Line 7 checks for the existence of a file named under the names of experimenter and participant. In case there is not, the next lines will do the work.
6. Line 8 saves the data under the name “Experimenter-ParticipantAutoSave.csv”.

**4.5. Server**

All blocks of the server were presented and detailed separately. With this information, it is possible to build our final server, which will put together the code for all interface parts to work according the application initially proposed.

**Example 23**

```

server <- function(input, output, session) {
  observeEvent(input$start,{
    vars$stTime=Sys.time()
    vars$start=T
  })
  observeEvent(input$pause,{
    vars$start=F
    end=format.timediff(vars$stTime)
    vars$data[nrow(vars$data)+1,]=c(input$experimenter,input$participant,0,end)
  })

  output$clock=renderText({
    invalidateLater(1000,session)
    if(vars$start){
      paste(format.timediff(vars$stTime))
    }else{
      paste("00:00")
    }
  })
}

```

```

observeEvent(input$behav1,{
  occurrence=format.timediff(vars$stTime)
  output$warn=renderText({paste("Behavior 1 recorded in", occurrence)})
  vars$data[nrow(vars$data)+1,]=c(input$experimenter,input$participant,1,
  occurrence)
})

observeEvent(input$behav2,{
  occurrence=format.timediff(vars$stTime)
  output$warn=renderText({paste("Behavior 2 recorded in ", occurrence)})
  vars$data[nrow(vars$data)+1,]=c(input$experimenter,input$participant,2,
  occurrence)
})

observeEvent(input$behav3,{
  occurrence=format.timediff(vars$stTime)
  output$warn=renderText({paste("Behavior 3 recorded in ", occurrence)})
  vars$data[nrow(vars$data)+1,]=c(input$experimenter,input$participant,3,
  occurrence)
})

  observeEvent(input$behav4,{
    occurrence=format.timediff(vars$stTime)
    output$warn=renderText({paste("Behavior 4 recorded in ", occurrence)})
    vars$data[nrow(vars$data)+1,]=c(input$experimenter,input$participant,4,
    occurrence)
  })

  observeEvent(input$expdata,{

write.csv(vars$data,paste0(paste(input$experimenter,input$participant,sep="-"),
".csv"))
  output$warn=renderText({paste("Data saved in the folder.")})
})

observeEvent(input$expgraph,{
  datum=vars$data
  end=datum$Occurrence[datum$Behavior==0]
  datum=datum[datum$Behavior!=0,]
  for(i in 1:unique(datum$Behavior)){
    datum=rbind(c(input$experimenter,input$participant,i,"00:00"),datum)
    ggplot(datum[datum$Behavior ==i,], aes(x=ms(Occurrence),
y=cumsum(grepl(i,Behavior))-1)) +
      geom_line() + geom_point()+xlim(0,ms(end))+ylab("Response frequency")
+ xlab("Occurrence per second")
    ggsave(paste0("Behavior-",i,".png"))
  }
  output$warn=renderText({paste("Graphs were saved in the folder.")})
})

session$onSessionEnded(function(){
  experimenter=isolate(input$experimenter)
  participant=isolate(input$participant)
  filename=paste0(paste(experimenter,participant,sep="-"),".csv")

filename.autoSave=paste0(paste(experimenter,participant,sep="-"),"AutoSave.csv"
)
  if(dim(isolate(vars$data))[1]>0){
    if(!file.exists(filename)){
      write.csv(isolate(vars$data),filename.autoSave)
    }
  }
})
}

```

## 5. Application

Now that all of the elements for the application are constructed, that is, global variables, the user interface, and the server, we can put all this together and make it run. All parts of the application can be stored in a single file or saved separately and loaded using the function `source()`. To make it easier, Example 24 shows all elements depicted as if they were in a single file, and summarizes the elements in lines using suspension points.

### Example 24

```
# Loading required packages
library(shiny)
library(lubridate)
library(ggplot2)

# Global variables
vars=reactiveValues( ... )
format.timediff = function(start_time) { ... }

# User interface
ui = fluidPage ( ... )

# Server
server = function(input, output, session) { ... }

# Running application
shinyApp(ui, server)
```

It is possible to run *shiny* applications hosted in collaborative platforms, such as GitHub®. For the reader to have access to the application in this chapter, it is available at GitHub® in the profile “ricardofcj”, “capitulo-shiny” repository. With that information, if readers have RStudio and Shiny package installed in a computer, they will be able to run it using the command `runGitHub(“ricardofcj”, “capitulo-shiny”)`. It is important to say that the ability to save files will not be operational, since the application was programmed to run locally. However, it is possible to make a copy of the application directly from GitHub® in the link <https://github.com/RicardoFCJ/capitulo-shiny/blob/master/app.R>.

## References

- Chang, W., Cheng, J., Allaire, J. J., Xie, Y., & McPherson, J. (2018). shiny: Web Application Framework for R. R package version 1.1.0. <https://CRAN.R-project.org/package=shiny>
- Cooper, J. O., Heron, T. E., & Heward, W.L. (2007). *Applied behavior analysis* (2nd ed.) Upper Saddle River, NJ: Pearson.



- Grolemund, G., & Wickham, H. (2011). Dates and Time Made easy with lubridate. *Journal of statistical software*, 40 (3), 1-25. <https://doi.org/10.18637/jss.v040.i03>
- LeBlanc, L. A., Raetz, P. B., Sellers, T. P., & Carr, J. E. (2016). A proposed model for selecting measurement procedures for the assessment and treatment of problem behavior. *Behavior Analysis Practice*, 9, 77-83. <https://doi.org/10.1007/s40617-015-0063-2>
- Campos Junior, R. F., & Rocca, J. Z. (2018). Introdução à linguagem de programação R aplicada à pesquisa e intervenção comportamental. In Hernando Borges Neves Filho, Luiz Alexanfre Barbosa de Freitas and Nicolau Chaud de Castro Quinta (Eds), *Introdução ao desenvolvimento de softwares para analistas do comportamento*, v. 1 (pp. 101-123).
- Mudford, O. C., Locke, J. M., & Jeffrey, K. (2011). Rates of responding measured by continuous recording in applied behavioral research. *Behavioral Interventions*, 26, 41-49. <https://doi.org/10.1002/bin.323>
- Springer, B., Brown, T., & Duncan, P. K. (1981). Current measurement in Applied Behavior Analysis. *The Behavior Analyst*, 4, 19-31. <https://doi.org/10.1007/BF03391849>
- Wickham, H. (2016). *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York. <https://doi.org/10.1007/978-0-387-98141-3>

## Chapter 4

### Is the programmer an author? Developing software for research<sup>1</sup>

Wivinnny Ferreira Lima  
*Universidade Federal de Minas Gerais*

Carlos Rafael Fernandes Picanço  
*Imagine Tecnologia Comportamental*

#### Abstract

Whether or not to involve software in experimental studies is a recurring issue in the research environment. Methodology planning involves decisions such as whether to use existing software, software that needs to be adapted, or to develop novel software. The present essay discusses criteria for authorship and co-authorship of research including such decisions. The discussion was organized in three topics: (a) issues related to the scientific relevance of the authorship of a software; (b) issues related to the practicality of developing software in a research environment; (c) ethical issues that arise when researchers collaborate with others either from industry (e.g., professional programmers) or not (e.g., research students that also develop software). Although only the authorship and co-authorship of software research is considered, the present work can serve as a basis for discussing the authorship of research that requires other specialized technical services (e.g., construction of mechanical or electromechanical devices). We hope that this discussion will proactively guide decision-making in the context of research planning, increasing the chances of success of relevant and injustice-free scientific endeavors.

---

<sup>1</sup> We thank the valuable commentaries of Dr. Shawn Gilroy on early versions of the present chapter.

Whether or not to involve software in experimental studies is a recurring question in the research environment. The planning of the methodology involves decisions as to whether to use existing software, which will need to be adapted before being used, or even novel software that will need to be developed. The purpose of this essay is to discuss the criteria of authorship and co-authorship of research with such decisions.

This discussion was organized in three topics: (a) issues related to the scientific relevance of the authorship of a software; (b) issues related to the practicality of developing software in a research environment (e.g., funding, testing, licensing); and (c) ethical issues that arise when researchers collaborate with others either from industry (e.g., professional programmers) or not (e.g., research students that also develop software). Besides discussing the issue of authorship and co-authorship of research involving software, the present text can also serve as a basis for discussing the authorship of research that requires other specialized technical services (e.g., construction of mechanical or electromechanical devices). We hope that this discussion will proactively guide decision-making in the context of research planning in scientific endeavors, increasing the chances of successful, relevant research, as well as recognizing the contributions of the entire research team (e.g., giving the right credit for the work of principal investigators, research assistants performing research, research students developing software, and so on).

### **On the scientific relevance of software authorship**

Next, several criteria are proposed to classify whether software has relevance (i.e., it is recognized as a valuable contribution) in the context of scientific authorship. There are times when software is directly relevant to replication or extension. It is accessible to interested researchers and the consequences of its use are part of the literature in such a way that it supports the control over intervening variables, dependent variables, or independent variables. That is, software is known to support experimental control and researchers may further include and extend it to novel application.

The relevance of software may be observed in other ways. That is, software may allow for experimentation to continue and expand at a reduced cost to the researchers. In contrast, the use of software that reduces experimental control to the point of preventing the testing of a research problem, of course, has no scientific relevance to this problem, regardless of its cost. In other words, reducing the cost is a valuable contribution to science as long as it does not compromise the investigation of the research problem (i.e., involving software in research just because it is cheaper is not reliable).

Another criterion of relevance is related to the generality of a scientific problem. Software may not be accessible in a domain of a scientific field of interest, but its benefits are known in another known domain. Different fields of knowledge (e.g., behavioral sciences, smart agriculture and computer vision) can share common features that allow the reuse of work between domains. After the translation from one field to the other, the source domain will have expanded its relevance, just as the target domain will have a new mean of investigation under its reach. Consequently, making the software accessible in the new domain has scientific relevance for both domains.

Yet another criterion of relevance is related to the exploratory and innovative character of the scientific enterprise. Novel software may emerge and its immediate value, having only been foreseen, may justify further investigation. A technological resource that was not accessible before, but now is, may generate new research questions, new lines of research, increase experimental control (i.e., control over the rate/location of behavior), reduce costs and so on. On the other hand, these predictions may not hold true and this could pose a risk to researchers. There is also the possibility of correct predictions, but that will not arrive at the right time, as software can quickly become obsolete. In fact, software that was a contribution in the past can lose its relevance in the future. Exploratory methodological research with contributions that guide other researchers to not make the same mistakes has

scientific relevance, although they do not necessarily guarantee an increase in experimental control, cost reduction and most problems.

Despite the vast resources invested in software, it will not always have scientific relevance, however a researcher may be induced to think so by mere popularity. In a general sense, when you have many researchers reporting using software on their own research, new researchers may be induced to use it too without proper reasoning, as a herd effect. Software has been seen by scientists as very important for the development of scientific research. For example, Hannay, MacLeod, Singer, Langtangen, Pfahl and Wilson (2009) conducted an opinion survey using a five-point likert scale (“not at all important”, “not important”, “somehow important”, “important”, “very important”) in which 84.3% of participants (who were scientists from different areas and regions of the world) responded that software development is “very important” or “important” for their own research, and 46.4% responded that software development is “very important” or “important” for other people's research. However, general opinion about the importance of software does not automatically convert into relevance. In fact, the inclusion of software in a research methodology may be completely unnecessary in some cases or it may, even if relevant, make the research unfeasible (for economic or logistical reasons, for example). Consequently, generalizations of this type should be avoided.

### **On practical actions**

Although the decision to include software in research methods may support scientific relevance, the decision may not be achievable in a practical sense. A critical variable that affects the practical execution of such decisions is time, whether it is the time needed to develop the program, or its adaptation and maintenance, or the time needed for training the people who will make use of the program.

Hannay et al. (2009) also investigated the opinion of scientists about software development time, again using a five-point likert scale (“much less time”, “less time”, “the

same time”, “more time”, “much more time”). The authors documented that 53.5% of the interviewed scientists reported spending “much more time” or “more time” developing software than 10 years ago, 44.7% reported that they spend “much more time” or “more time” developing software than 5 years ago and 14.5% reported that they spend “much more time” or “more time” developing software than 1 year ago. Another data obtained in this research was that the scientists reported spending approximately 30% of their working time on developing software.

A researcher's time is a finite resource, and time administration directly affects the quality of the research enterprise. After all, meeting deadlines is part of the scientist's routine and this professional often accumulates several functions (e.g., advising, giving classes, administrative and research functions). By including the programming activity as part of a research activity, there will inevitably be less time for other activities. Consequently, there must be moderation in the time spent in the software development cycle, and this scenario leads to (programming) activities shared by teams that may include students in graduate, master's or doctorate, for example. The development cycle in the academic environment has different characteristics than the software development cycle in a business environment. These differences have implications for the authorship of the final research manuscript, as elaborated below.

### **The development cycle in the academic environment**

We live in a culture with people used to pay for a product and receive it on time. Then, you may think that an alternative to spending hours developing software is to hire a professional. In this way, a contract can be signed between the parties transferring the copyright of the software to the authors of the research, ensuring exclusivity of the authorship rights of the final work. However, in the academic environment, hiring a professional can severely delay or fully compromise a research. A first challenge will be to obtain, in a timely manner, the financing that covers the costs of the self-employed professional or development

company. However, funding may be the least of such challenges. The contractor (e.g., through a company) will need to work together with the scientist and herein lies the difficulty.

Hannay et al. (2009) noted that there is a gap between professional programmers and the scientific community. A professional programmer can develop any software as long as the client (in this case, the scientist) can describe its purpose and execution. And it is exactly in this description that a difficulty is encountered, the scientist deals with knowledge under construction and, by definition, unfinished and temporary, which makes it difficult for the programmer to initially write the requirements of the service (i.e., to write the blue prints of the software). This characteristic of research has led us to prefer an “on demand” style of writing software. We chose to write software in a modular way and in short development processes, remaining open to improvements and correction in the long term. In addition, whenever possible, instead of writing software for a long term research agenda, we choose to writing a computer program for a specific research, simplifying the survey of program requirements.

Different from the “on demand” style of development suggested and illustrated in the previous paragraph, Segal (2005) reports a case in which a group of independent programmers developed software, with difficulties, for a group of scientists. The author cites two main problems throughout the development process: (1) Scientists were accustomed to developing software in a more interactive way and with successive meetings established on demand, hence they had a lot of difficulty when the programmers requested all the demand in a single moment; (2) The use of contract documents and limited time meetings was insufficient to create an understanding between scientists and programmers.

In fact, the software development cycle in the academic environment has challenging characteristics. In an ideal research environment, there is an ongoing development process, in which the advisor, students and programmer will set aside time to (1) help the programmer specify the program requirements and (2) to test the program and report problems.

### On ethics

As suggested earlier, the professional programmer in general has no interest in authorship and will transfer rights to researchers. In the context of the publication of scientific research in indexed journals, this means that the responsibility for matters related to the software will rest exclusively with the authors (or author) of the published work, not with the company that developed the software and that, it should be noted, knows it in depth. This means that the researcher who receives demands related to the responsibility of the software, even in full rights, will not be able to respond to these demands.

On the other hand, the researcher who is also the programmer of the research software will share authorship and responsibility with the research group. The people in the group who participated in the development process are authors of the software, not just who writes it, as well as the people in the group who participated in the development of the research are authors of the manuscript derived from the research. For the purposes of this essay, the authorship of a scientific work will be defined according to the four criteria presented by the International Committee of Medical Journal Editors (2018), the same criteria currently used by the journal *Frontiers in Psychology*. According to this definition, the author of a work is someone who meets at least one requirement of each of the criteria below:

- First criterion:
  - contributes substantially to the conception of the work; *or*
  - contributes substantially to work planning; *or*
  - contributes substantially to the acquisition of data for the work; *or*
  - contributes substantially to the analysis of data for the work; *or*
  - contributes substantially to the interpretation of data for the work.
- Second criterion:
  - drafting the work *or*
  - revising the manuscript critically for important intellectual content.



- Third criterion:
  - approves the final version of the manuscript to be published.
- Fourth criterion:
  - Agree to be accountable for all aspects of the work, ensuring that issues related to the accuracy or integrity of any part of the work are properly investigated and resolved.

As previously developed in the subsection “The development cycle in the academic environment”, it is possible to notice that the researcher who is also a programmer contributes substantially to the acquisition of work data, as he builds and documents the use of the tool used for data collection. In this way, when building and documenting the tool, he will be able to respond to the demands of software responsibility with autonomy. In addition, the researcher who is also a programmer contributes substantially to the planning of the work when he fills methodological gaps that were not foreseen during his conception. Due to his programming skills, the researcher who is also a developer contributes substantially to the analysis of data when he generates alternative analyzes and when he automates the process making it feasible in a timely manner. By knowing the data acquisition tool and science in depth, it can also contribute substantially to the interpretation of the data, either by identifying points for future corrections, or by contributing to the exploration of alternative strategies in a creative way through programming languages specifically developed for data analysis (such as R and Octave). There is no doubt, therefore, that the researcher who is also a programmer can meet the first criteria of authorship.

Consequently, working as a reviewer also allows each of the other three criteria of authorship to be met. But in the case of reviews there is a problem for those who identify themselves as researchers and programmers; invitations may not happen, as the programmer's work is not recognized as a substantial contribution to research, which may overshadow his contributions related to the first criterion. The case of short-term researchers in the medical

tradition (Newman, 2006) is similar, and illustrates the social consequences of this problem for researchers who are also programmers: low recognition and difficulties in pursuing a scientific career.

A thought that illustrates the picture of low recognition considers that the researcher who is also a programmer acts only as a “developer” and nothing else. Phrases like “The programmer does not help in the development of the research”, illustrate this type of thinking. Now, as previously discussed, the programmer can make substantial contributions to research, for example, by creating a tool that ensures strict control for experiments, that establishes strict protocols or that registers new data of interest to the area. To argue that the programmer does not help in the development of the research is, therefore, untrue.

Finally, it should be noted that the criteria mentioned here are applicable to specific research conducted by the group members or by individuals and not to derivative research conducted by third parties. In this sense, the present essay argued that someone who is a researcher and also a programmer makes substantial contributions to a specific research when writing software specifically for it, and does not make substantial contributions if a third party research, derived from that initial research, only uses the software previously developed. The use of a software previously developed by third parties does not guarantee the fulfillment of the first criterion of authorship in the research of these third parties. Consequently, we maintain that it is ethical that the researcher (also programmer) is not invited as an author for the research of third parties who only use the software. In this case, it is ethical for those only using the software to cite it (in accordance with the standards recommended by the chosen means of communication, e.g., in accordance with the standards of the American Psychological Association in psychology journals).

### References

- Hannay, J. E., MacLeod, C., Singer, J., Langtangen, H. P., Pfahl, D., & Wilson, G. (2009). How do scientists develop and use scientific software? In *2009 ICSE Workshop on Software Engineering for Computational Science and Engineering* (pp. 1–8). Vancouver, BC, Canada: IEEE. <https://doi.org/10.1109/SECSE.2009.5069155>
- International Committee of Medical Journal Editors. (2018). Defining the Role of Authors and Contributors. Retrieved November 17, 2018, from <http://www.icmje.org/recommendations/browse/roles-and-responsibilities/defining-the-role-of-authors-and-contributors.html>
- Newman, A. (2006). Authorship of research papers: ethical and professional issues for short-term researchers. *Journal of Medical Ethics*, 32(7), 420–423. <https://doi.org/10.1136/jme.2005.012757>
- Segal, J. (2005). When Software Engineers Met Research Scientists: A Case Study. *Empirical Software Engineering*, 10(4), 517–536. <https://doi.org/10.1007/s10664-005-3865-y>