The Fireman's Fund iRise Cookbook
Revision 1.2

Matt Denko

Fireman's Fund iRise Cookbook

This document details a variety of "recipes" that document how to accomplish a variety of data programming tasks within the iRise environment. The intent is to create a library of common data manipulation examples, to facilitate re-use and reduce wasted/duplicated effort when updating iRise project files or when starting a new iRise project. Note that the recipes in this Cookbook are specific to the 8.5 version of iRise, although most if not all will work in earlier (and possibly later) versions of iRise as well.

The document is structured to begin with recommended best practices and common, basic editing tasks, and moves on to more intermediate and advanced simulation topics. The following topic groupings are covered in this document:

- General Best Practices
- Basic Editing
- Common Formatting Tasks
- Working with Data
- Performing Calculations and Conversions
- Form Validation Routines

This document is not necessarily meant to be read in a linear, sequential sequence: you can pick or choose "recipes" at random based on your level of interest or current iRise editing task. A baseline level of proficiency with the iRise application is required to master the more difficult recipes that appear later in the Cookbook. If you find yourself challenged by a recipe, you may find valuable hints and tips in earlier portions of the Cookbook and might benefit by brushing up on these earlier topics.

While generally geared towards simple editing tasks that don't always map well to more complicated real-world examples, the integrated iRise Help (accessible by clicking your F1 keyboard key while in the Studio environment) does a good job of introducing new users to the basics of working with the application. Several additional resources are also available that can greatly help in becoming conversant with a variety of common iRise tasks:

- **Online Tutorials** (http://www.irise.com/services/training_elearning.php). Provide a good starting framework for working with simple tasks using the application.
- **Online Seminars** (http://www.irise.com/resources/seminar_center.php)
- Common Sample Files (http://www.irise.com/resources/idoc_library.php). Make sure to grab the version specific to your current version of iRise.
- **iRise Developer Network** (signup URL: http://idn.irise.com/um/signup.action). An online support community that can help answer many common questions as you're coming up to speed with working the with application.
- **IRise Support** (1.866.361.3900, or support@irise.com). Standard support hours are from 6:00 am – 6:00 pm Monday through Friday, Pacific Time. A surprisingly strong support organization; they have saved my bacon more than once when up against a project deadline. There are only a handful of support technicians, but they all have solid technical backgrounds, and each

have unique strengths working with different aspects of the application. They will quickly become your new Best Friends.

This document assumes the reader has a solid foundation in working with the application, and has familiarized himself/herself with the basics covered in the online Help and has successfully completed at least the following online tutorials:

- iRise Overview
- iRise Explained

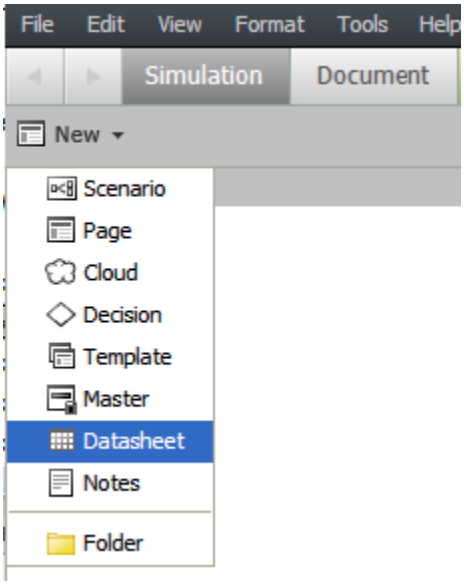
Working with This Cookbook

It is recommended that you create a local project in which to work through the various recipe examples contained within this guide. To do so, complete the following steps:

1. If not already open, launch iRise Studio on your system.
2. From the File menu, choose New > Project (Blank).
3. In the New Project dialog box, enter a name for your project, then click Create.

If you plan to work through any of the datasheet examples later in this guide, it is recommended that you import the sample States datasheet that is included with the application. This datasheet is in a subdirectory below the root installation directory of the application (exact location detailed later in the recipe steps).

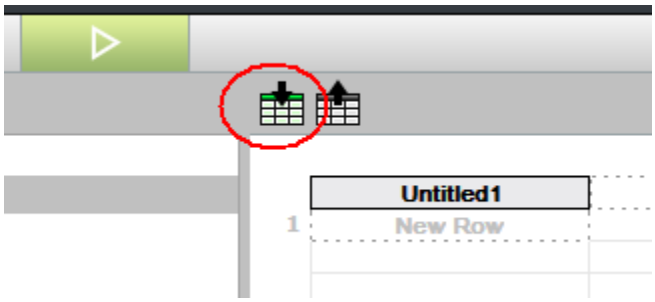
1. From the tree at the left of iRise Studio, choose New > Datasheet.



2. While the datasheet is still highlighted, type "States" to change the default name.

3. Select the Datasheet you just created from the tree.

4. Click the Import CSV button at the top of the Datasheet window.

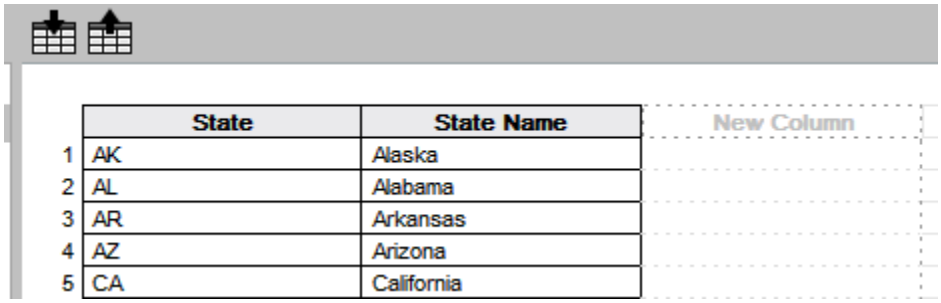


5. In the CSV Import From dialog box, navigate to the Sample Datasheets directory below your iRise installation.

   By default, iRise is installed to C:\Program Files\iRise. If you installed to a different directory, the datasheet samples are stored below the root folder as follows: \\ installation directory\Studio\Sample Datasheets.

6. Select State.csv, then click Open.

   This should display a two-column table within the Studio environment.



	State	State Name	New Column
1	AK	Alaska	
2	AL	Alabama	
3	AR	Arkansas	
4	AZ	Arizona	
5	CA	California	

General Best Practices

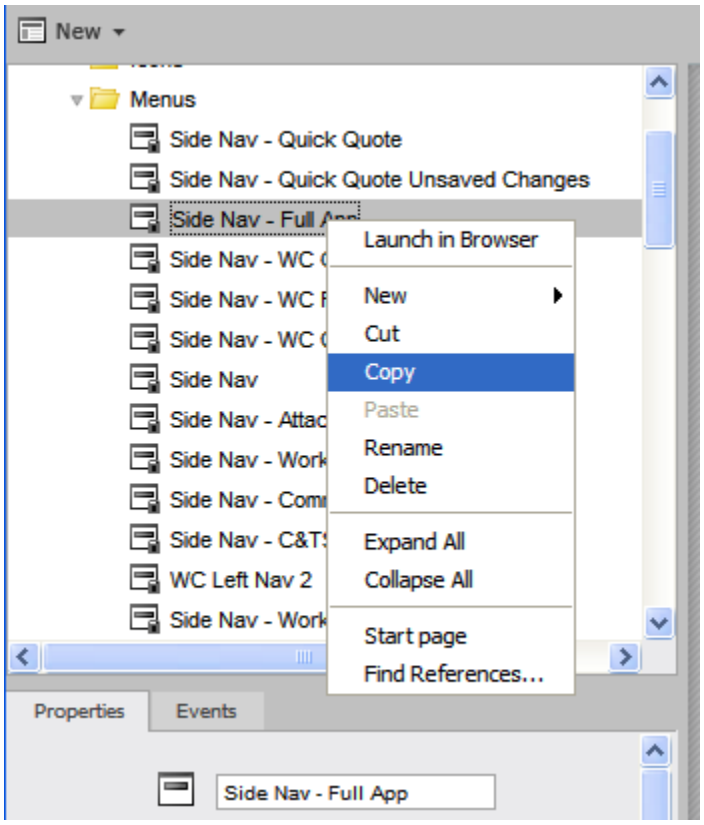
The following section details some high-level recommendations for general best practices (not presented in any order of importance) when creating an iRise simulation.

1. When making more than superficial changes to a project part (e.g., pages, datasheets, decision branches, etc.), first create a backup copy of the current page.

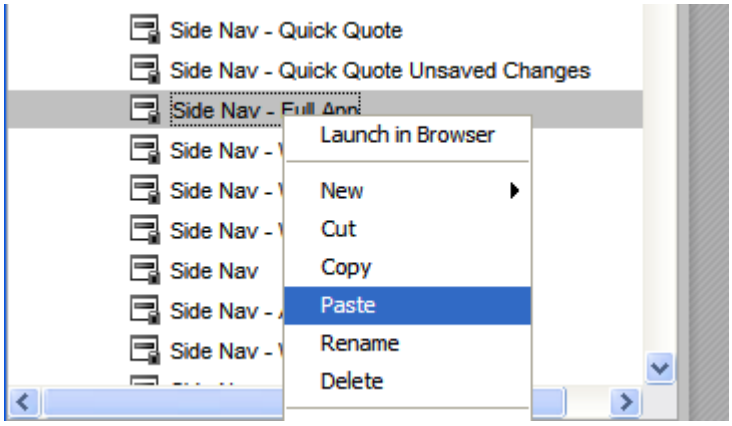
As page logic becomes more complex, it becomes increasingly easy to accidentally break routines that had previously worked without hitch. Recreating this page logic can be quite time consuming without a backup copy to guide your efforts.

To make a backup copy of a project part:

1. From within iRise Studio, right-click the project part that you want to copy from the project tree, then choose the Copy context menu option.



2. Right-click the region within the tree where you want to insert the copied project part, then choose the Paste context menu option.



2. If multiple versions of a given project file exist, take steps to ensure you're working on the correct version of the file.

An important caveat related to the first best practice is taking steps to mitigate working on the wrong file when performing edits. Make sure to clearly differentiate the file versions with clearly descriptive label differences that quickly jump out when viewing the files from the project tree. It is not always sufficient to merely append a single digit number to the end of a project file – it is extremely easy when caught up

in an editing session to get fifteen minutes or so into a series of intensive edits, only to realize you're working on the wrong version of the file. Try to build check points into your workflow to pause and apprise your progress, and to ensure you're doing your work in the appropriate file. Undo can help save you to a certain extent, but not all operations within iRise can be undone, and there does seem to be a limit on the number of operations that can be undone, particularly if you are jumping back and forth between multiple files while editing.

3. Save incremental versions of your project files.

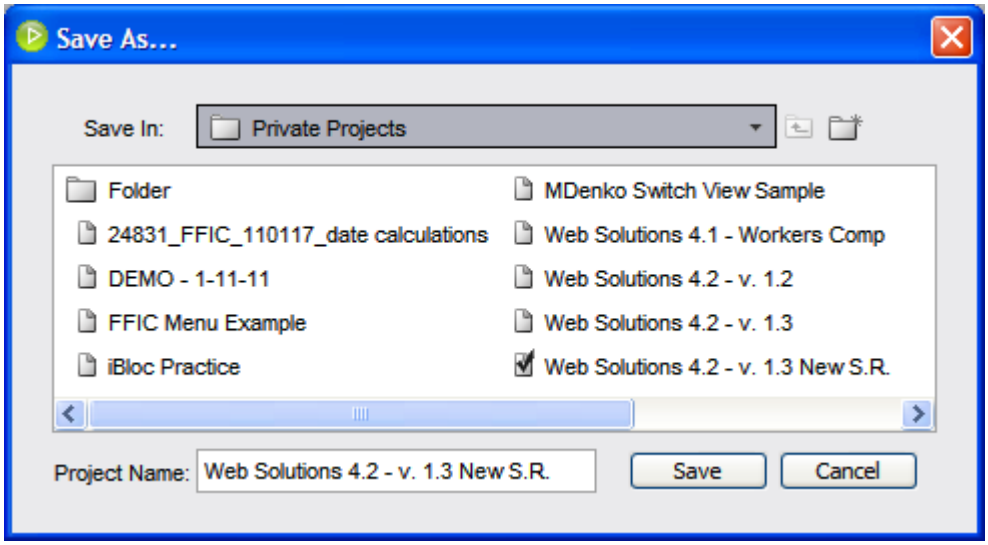
IRise project pages quickly become quite complicated, especially if they have extensive data passing or page logic. Due to the cluttered nature of certain pages within the iRise Studio environment, it is extremely easy to inadvertently move or delete objects without realizing what has been done until there is no easy way to reverse the damage. If you forget to create backup versions of project components before beginning extensive edits, having an earlier version saved to the Definition Center or as an iDoc can be a life-saver. Irise automatically saves all edits you make to a project during an editing session (including ones you might not be aware of), so having an earlier backup of a known good state allows you to import an earlier state of one or more project files into the current project.

To save a new version of an iRise project:

1. From within iRise Studio, choose Save As from the File menu.

2. From the Save As dialog box, choose one of the following:

    a. To save a local version of the project, choose Private Projects from the Save In field.



    b. To save a version to the Definition Center, choose Connect to Definition Center in Save In, then enter your login credentials in the Definition Center Login dialog box.

3. Specify a unique name for the file in the Project Name field, then click Save.

To create an iDoc of an iRise project:

1. From within iRise Studio, choose Export > To iDoc from the File menu.

2. In the Export to iDoc dialog box, navigate to the directory location you'd like to save the file to.

3. Enter a unique name in the File Name field, then choose Save.

To import a project part from another project into the current project:

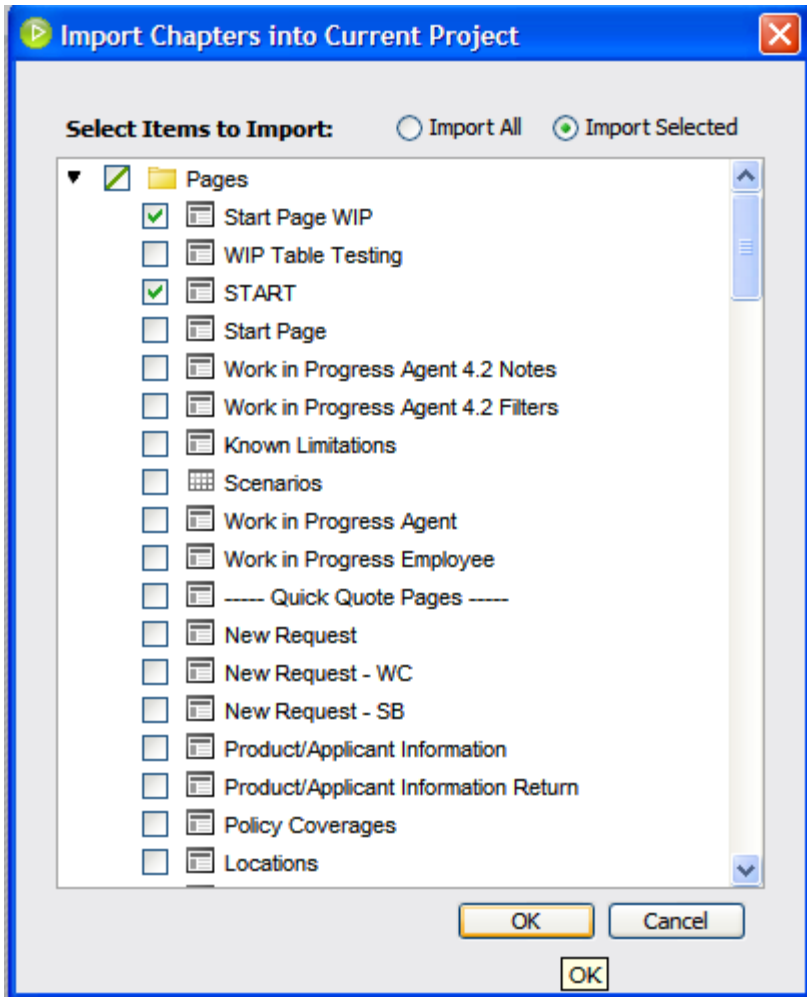
1. From within iRise Studio, choose Import, then do one of the following:

    a. To import from another project, choose From Another Project.

    b. To import from an iDoc, choose From An iDoc.

2. In the dialog box that is displayed, navigate to the project or iDoc that contains the project parts you want to import.

3. In the dialog box that is displayed, place check marks next to each project part that you want to import, then click OK.



4. When working strictly on page formatting/layout issues, toggle iRise to Page Layout View.

The Page Layout View temporarily hides all page logic, displaying only the actual page widgets that comprise the current page. This can dramatically ease the process of laying out your pages and decreases the chances of accidental edits that can be difficult to detect when viewing the page logic controls. Which of these two document views would ***you*** rather perform extensive editing operations on?

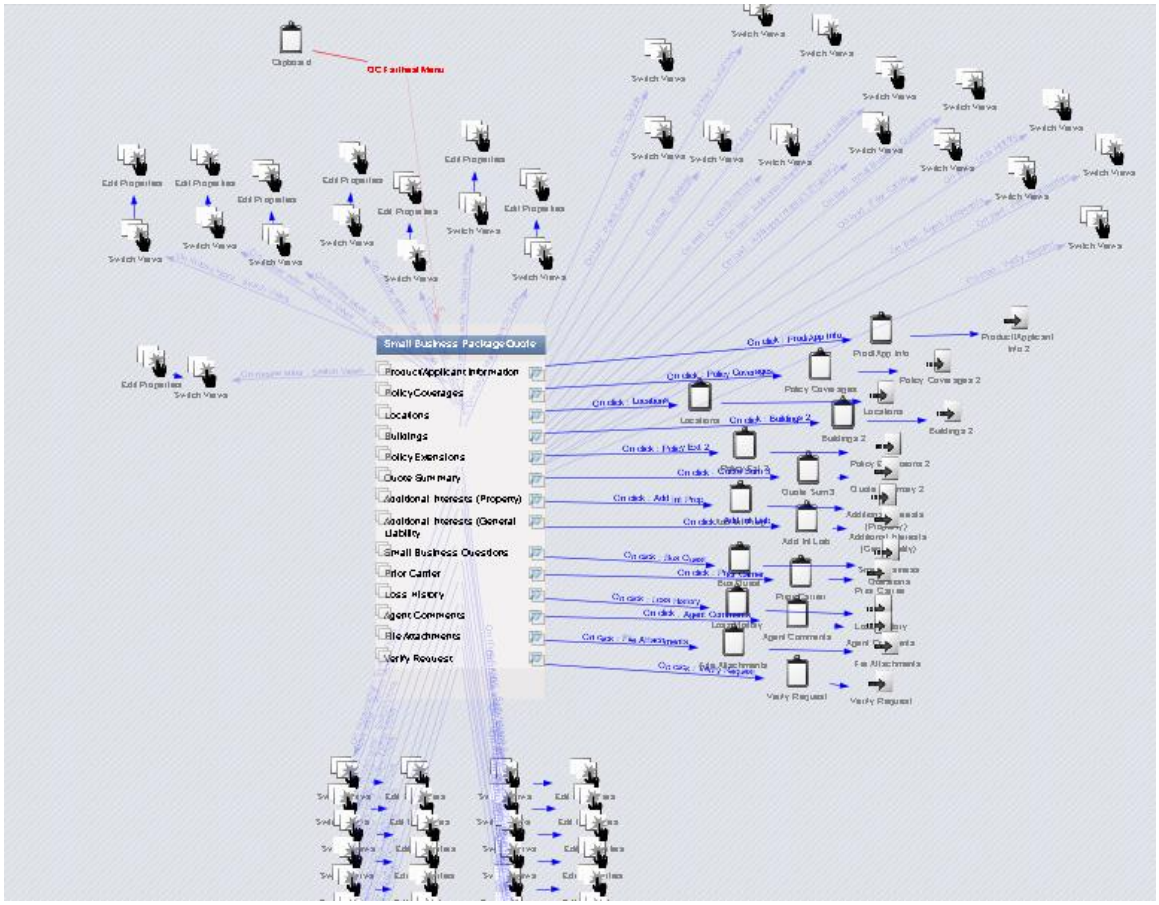


Figure 1: Page View with Canvas Widgets

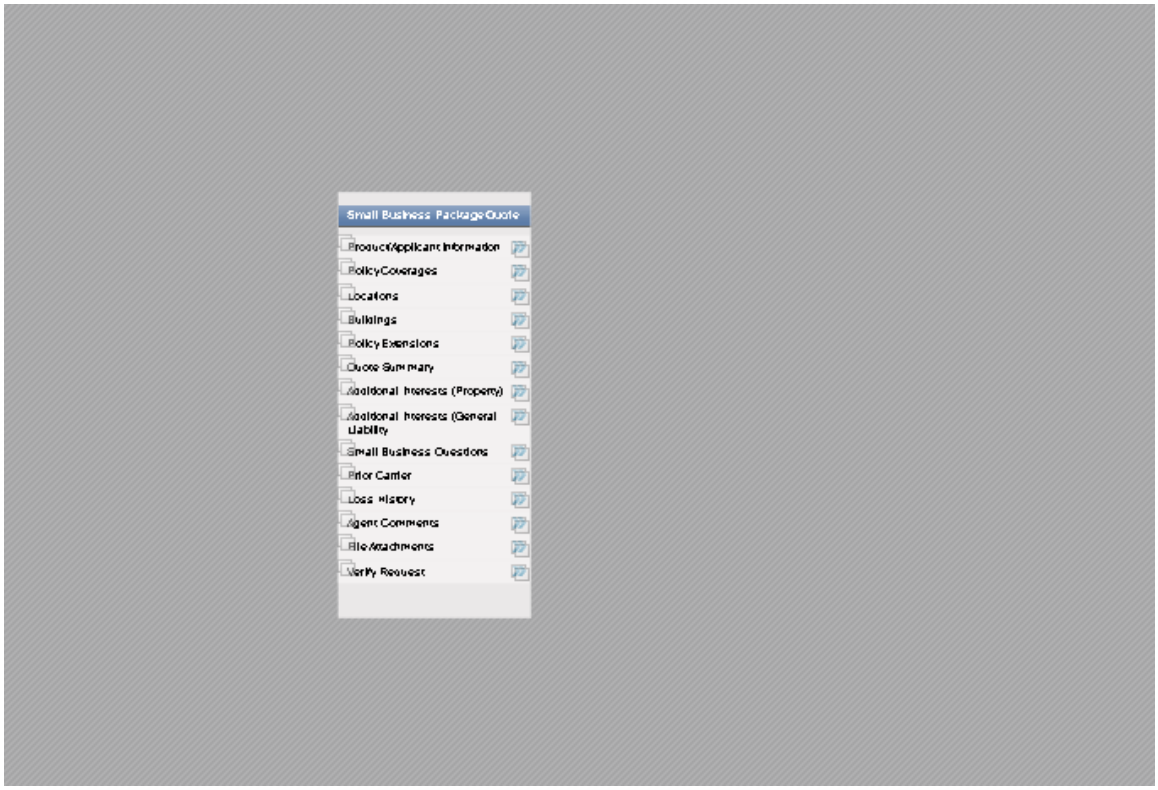


Figure 2: Page Layout with Canvas Widgets Suppressed

To Toggle Page Layout View on Or Off:

1. Press the F12 key on your keyboard.

5. When working with large amounts of data, consider using datasheets instead of clipboard widgets.

iRise provides two primary methods of capturing and manipulating user data: clipboards and datasheets. While initially somewhat trickier to work with and to grasp conceptually, datasheets offer some distinct advantages that clipboards lack, the primary one being the ability to collapse linkages to individual page widgets down to a single line. For example, when working with either clipboards or datasheets, after your initial pass of hooking individual variables up to specific page widgets, your editing environment might look something like the following:

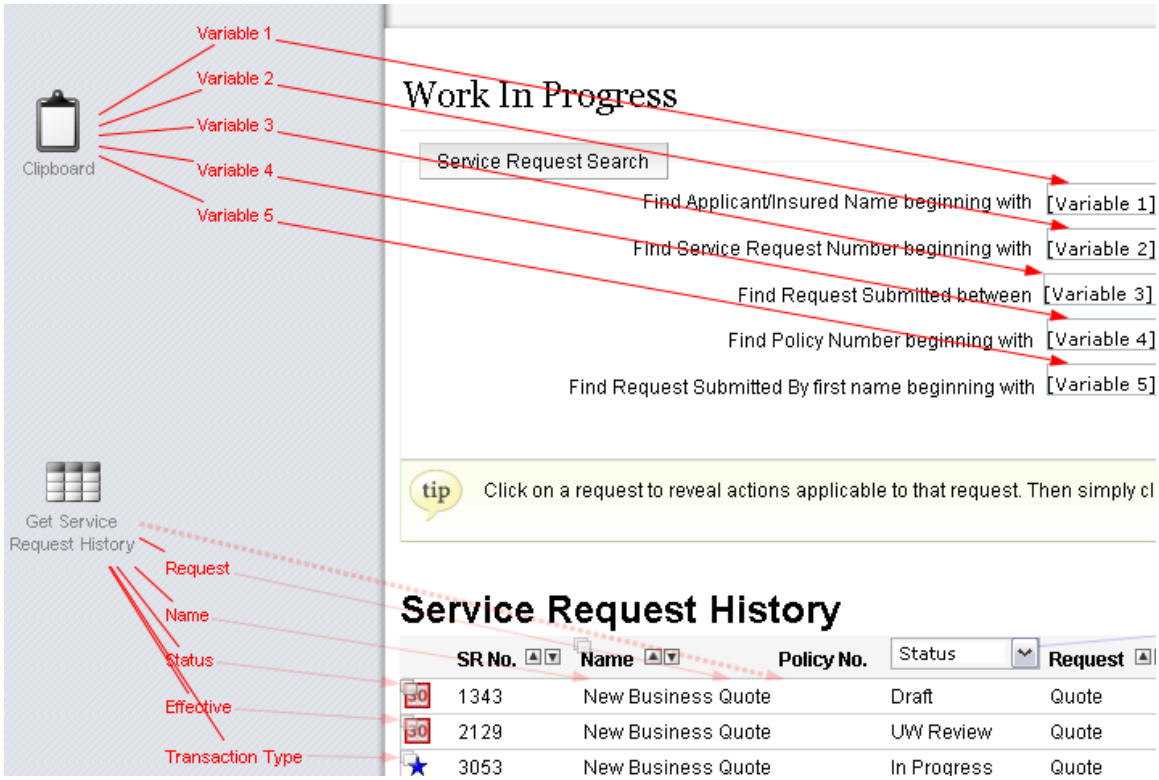


Figure 3: Clipboard (Top) and Datasheet Canvas Widgets

While simple pages can easily accommodate a small number of clipboards/datasheets with one or two handfuls of variables mapped to page widgets, the situation can quickly become overwhelming, making it difficult to perform editing or formatting operations on the page. As previously described in the Best Practices section, you can mitigate this difficulty by toggling to Page Layout view. If, however, you still want to be able to visualize how data is mapped to various regions on the page, you can collapse your datasheet data lines down to a single line, as depicted in the following illustration.

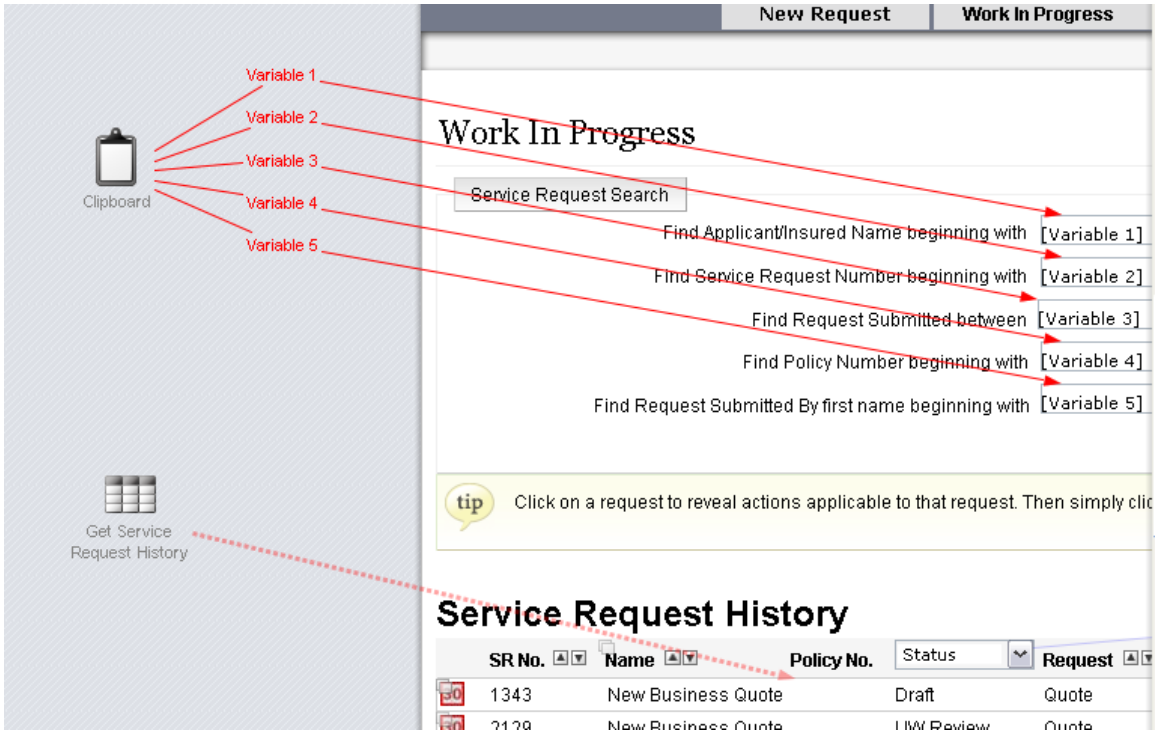
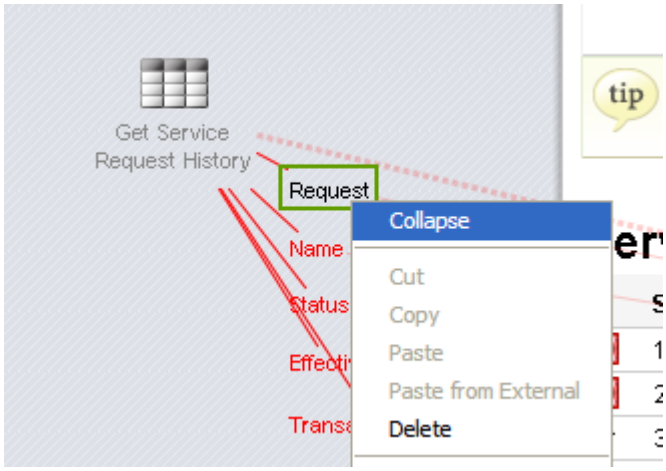


Figure 4: Datasheet (Bottom) Collapsed

To Collapse Datasheet Logic Down to a Single Line:

1. Right-click on any one line running from the datasheet widget to the page, then choose the Collapse context menu option.



To Expand a Collapsed Datasheet:

1. Right-click on the collapsed data line, then choose the Expand context menu option.

6. Don't pass data from a single clipboard to multiple views.

Results are unpredictable – sometimes things work as expected; sometimes not. When passing data to a control that is contained in more than one view, use separate clipboards for each view. Note that this suggestion is only applicable when passing data to analogous control representations in more than one view: a single clipboard can be used to pass data to multiple containers within a single view space (for example, data can be passed from a single clipboard to multiple page elements; to canvas operators; etc.).

7. When inserting a single image into multiple projects, first create a master reference that contains the image.

I learned the hard way that iRise project files have a limit with respect to file size and project performance. One evening while preparing to leave for the day, I got a phone call from a colleague requesting an iDoc of a project for use in a presentation the next morning. "No problem!" I replied, and as I packed up my belongings to prepare to head out for the day, I kicked off the operation to output the iDoc. Normally a fairly quick operation, I picked up the phone to call my colleague after 30 minutes had elapsed and the operation had not completed. When I came in the next morning the operation had successfully output the file, but I once again started the export operation to time how long it took. Once again at the 30-minute mark, I picked up the phone to call iRise support. While still talking with the support technician, the operation finally completed after approximately 50 minutes.

I learned over the course of that conversation with support that the recommended ceiling on file sizes is approximately 5MB. The project in question at this point had bloated to over 12 MB.

In helping to troubleshoot the issue, iRise requested that I forward the bloated file to them. In looking through the project XML file, iRise noted that the project contained many thousands of individual file instances. In more closely looking at the XML file, it soon became evident that many of these instances were duplicates – sometimes hundreds of instances of the same source file were placed into the project.

The iRise recommendation for cleaning up this mess was to find the most egregious examples in the project, and capture these as masters. One of the primary benefits of masters is that they can be used as references within a project. Define the master reference a single time, and individual instances of the masters inserted into project files point back to the master. From a file size perspective, if you have a single 1 MB image that needs to be inserted 100 times into a project, masters result in a 100 times savings in file size. Rather than having individual instances of the 1 MB file resulting in 100 MB of project file bloat, the individual instances point back to the single master reference, taking up only 1 MB of file size.

8. Leverage masters in other ways to minimize duplicated effort and rework in a project.

An important distinction in this discussion is that iRise offers two different master types, and each is appropriate for different contexts. References behave as described in the previous best practice recommendation: define the master once,

and have individual instances point back to the parent master. iRise also supports master copies, which behave like individual, single instances, and thus negate the file size advantage previously discussed. In many cases, however, this is not really a problem with respect to file size and offers other advantages that reference masters lack. For example, when working solely with native iRise widgets, file size savings are negligible (if existent at all) since the internal program logic of iRise is intelligent enough to internally treat the multiple widget instances in a manner like what was discussed with respect to external file types such as images. A big benefit of copy masters is that they don't limit you to having the same content in each instance. This is highly useful when defining masters of common page elements, such as sections are tabular elements that need to expose variable content.

With respect to master references, one other benefit is worth highlighting: when working with a feature team on rapidly evolving requirements, defining repeatable project elements a single time in a master reference allows you to iterate the reference without having to update individual instances across multiple project files. Update the reference a single time, and those updates are propagated automatically to all project files that reference that master.

The following sections highlight some of the important distinctions with respect to the two different master types.

Advantages of Reference Masters

- Minimize project file sizes when working with external (i.e., non-native) object types
- Reduce rework if a reference project type needs to be updated: the edit is performed a single time in the master reference, and automatically propagated to all instances

Disadvantages of Reference Masters

- Because the reference is inserted as a single block into a project page, individual elements that comprise the reference can not be accessed or manipulated
    - Due to this limitation, actions and data flows cannot be defined at the page level – they must be built into the master
        - This limits certain capabilities – any view or other switches must refer to elements contained in the master. For example, using a reference master, you can't define logic to invoke a view change for a page element that is not contained within the master without breaking the reference to the parent master.
    - If multiple instances of the master have slightly variable content, either multiple master references for the different permutations must be defined, or the linkage to the parent master must be broken.

Advantages of Copy Masters

- Ideally suited for situations where individual components of the master need to vary across multiple project instances
- Allow for greater flexibility in interacting with data and other widgets on individual project pages

Disadvantages of Copy Masters

- Any subsequent edits require updating multiple instances across project files
- Can lead to project bloat if they include external file types such as images

In the next section, we'll be covering a clever work-around that helps to blend the good points of both master types, while eliminating the bad.

9. Use nested masters to maximize benefits of both master types, while eliminating the limitations.

I accidentally stumbled upon a solution that eliminated the problems discussed in the previous section for both types of masters. While working through the mitigation steps outlined by iRise for reducing file size in my problem project, I realized that some of the image instance I was striving to eliminate were contained in multiple different masters. For example, various message icons appeared across multiple different project masters. What would happen, I wondered, if I further subdivided my masters into multiple parts? At the micro building block level, I experimented with creating masters of the smallest project parts – individual image references for icons and other small images that were repeated hundreds of times across the project. I next began creating slightly larger masters that contained references to the smaller masters – essentially, I began nesting the smallest masters into slightly larger parents, and the mid-sized masters into still larger ones.

The primary beauty of this approach was that it eliminated the limitations previously described, while maintaining the advantages of the two types. More specifically, a nested master could be built up of reference masters pointing to immutable content such as the project icons and images. These reference masters could then be placed into copy references for variable elements such as masters of various error messages. File bloat is avoided by defining reference masters for images and other sources of external content, while still providing the flexibility to vary content by placing those elements into a copy reference.

When appropriate, use nested references to maximize benefits while minimizing drawbacks of the two different reference types.

10. iRise support: your new best friend.

(Phone: 1.866.361.3900, or support@irise.com)

I must confess to a certain stubborn streak in my personality: when confronted with a problem, I have a bulldog tenacity in attempting to solve it that at times borders on OCD. This can be both a good and a bad thing. I have realized over time that there is a point of diminishing returns at which I must admit to myself that I am not going to solve the problem on my own, at least not without leaving it to stew for a while in the back chambers of my subconscious. My experience with support organizations has been spotty at best: while occasionally helpful, oftentimes my experience has been that I know more about working with the application I'm having trouble with then the support staff I call when I'm hopelessly stuck.

Thankfully, this is not the case with the iRise support team. While some team members are stronger than others, all bring unique skills and capabilities to the table, and all have been helpful in different contexts when grappling with a problem.

If you take no other advice in this manual to heart other than this one, my experience has been that after grappling with a problem for more than an hour or two, it is time to call in reinforcements: pick up the phone and call iRise support.

For newcomers to iRise, there are also strong internal resources available who have grappled with many of the same issues that you will be as a newcomer to iRise. Rather than trying to reinvent the wheel, call up one of these resources and have them walk you through an example they've created in the past to solve an actual design problem relevant to Fireman's Fund. Some excellent internal resources who might be able to lend a hand when you run across a thorny iRise prototyping task include:

- Meri Dreyfuss
- Anna Poznyakov
- Rachel Wahlberg
- Matt Denko

11. Avoid crossing widget/data flow lines to improve page legibility.

In a former work life while a designer for the Autodesk product AutoCAD, I had to learn a bit of architectural drafting while conducting my design research. It is considered a standard best practice in drafting disciplines to avoid (where possible) crossing notational drawing elements such as dimension lines to improve legibility of drafting documents. Similarly in iRise, pages can quickly become quite complicated and it greatly improves the readability of iRise pages and helps to visualize data flow if line crossings are kept to a minimum.

iRise Objects and Event Types

iRise many different events that can trigger actions. In Object Oriented programming languages, the various components and widgets that make up the application are referred to as *objects*, each of which have *properties* and *methods* available to them. The various page widgets that you place within your simulation in iRise all loosely correspond to the notion of objects exposed in many programming languages. When you select an object from the page canvas, a Properties Palette control docked at the left of the application environment displays information about the currently selected object. In the illustration below, a single text widget is selected on an iRise page, and the Properties Palette is displaying properties related to the widget, including the following:

- The name of the widget (Text 1)
- Vertical and horizontal offsets of the widget from a parent container such as a section (as the widget is not contained within a parent container, these values are blank)
- The X, Y coordinate location (in pixels) of the object from the origin (0, 0 location) of the page (in other words, the upper left portion of the page, which begins on the canvas at an assumed position of 0 pixels for both the X and Y axes)

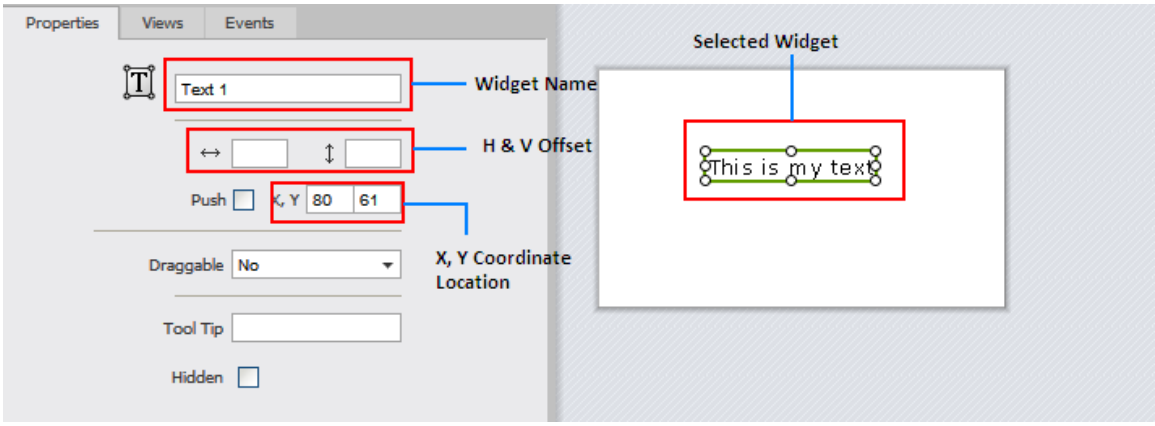


Figure 5: A Few Selected Properties of a Simple Text Widget

Objects also have methods that are available to them. Whereas properties can be thought of as variable physical characteristics of a given object (for example, the name of a text widget, or the color or font that comprise how it is rendered on the page), methods can be thought of as verb-type actions that the object can perform. For example, button and drop-down widgets usually have corresponding actions that are associated with them: when you click a button on a web page, you typically expect that button click to result in some noticeable action: perhaps navigating to another page within the work-flow, or submitting changes on a web form to a back-end database. These actions are fundamental methods that are defined in the object-oriented language and associated with a given object type.

Many programming languages do some of the grunt work involved in predefining a number of object types, determining what properties these objects expose, and associating pre-baked methods with the objects. Many objects also expose events, which trigger an associated action when the event is invoked. In our button example, the button exposes an **onClick** event, which can fire off an associated method that invokes an action when the button is clicked.

While not a true development environment, iRise borrows many elements from object-oriented programming languages, including exposing several predefined events for various object types. For example, the parent object in iRise (the page widget) exposes predefined events, including the following:

- **On click** – an event that triggers an associated action when the parent object is clicked using the left mouse button.
- **On context** menu (right-click) – identical to the On Click event, except the invoking action is clicking using the right mouse button as opposed to the left.
- **On load** – an especially powerful event that is only exposed (unfortunately) for a limited number of object types (pages and masters). The On Load event is triggered every time the parent object is loaded into the browser: when the page/master is first loaded; when the page is refreshed; etc.

To view the events that are available for various objects, start a new page in your project and place a few different widgets onto the page. To see the events that the object exposes, select it on the drawing canvas, then click the Events tab located to the right of the Properties Palette.

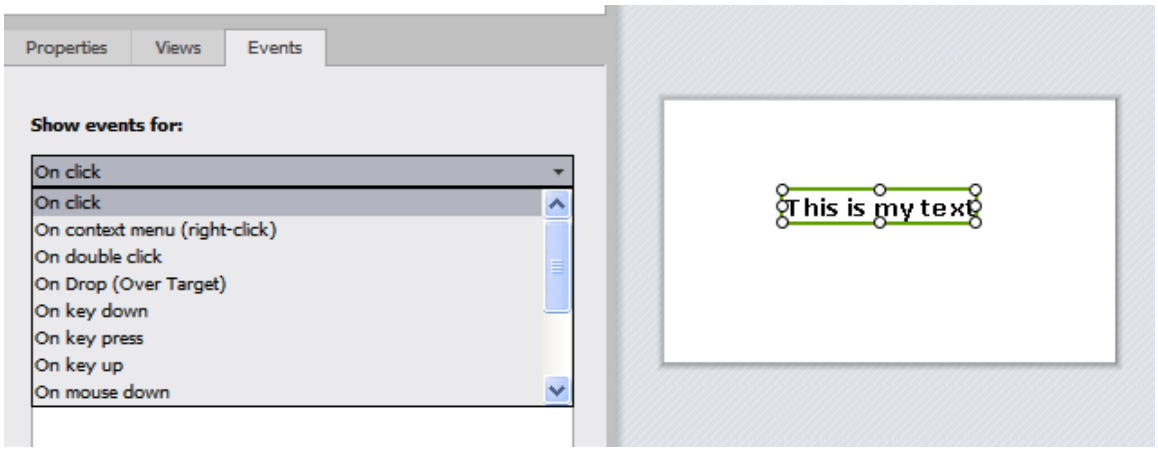


Figure 6: A Subset of the Events Available for Text Widgets

Events are a fundamental cornerstone of creating dynamic simulations using iRise, so it is important that you become familiar with the various event types that iRise makes available in your prototyping tool belt. We'll be regularly working with events in the coming recipes, so if you don't have a basic understanding of the types that are exposed at what they can do, it is recommended that you review the online Help content provided with the iRise Studio application.

Since iRise is not truly a programming environment (with the notable exception of the new iBloc object, which is described later in this document), some important limitations exist that don't when working with a traditional programming language. It is also important to have a solid understanding of these limitations, and your awareness of them should guide decisions when figuring out how to perform various simulation tasks within iRise. The most important (and unfortunate) limitation is that as opposed to a programming language, a single iRise event can only be controlled by a single variable for a given object. This means, for example, that you can't setup decision logic (If/Then/Else sorts of statements) that perform different actions depending on whether a different variable is specified: in iRise, you're limited to an all or nothing reaction limited to a single variable. You can, of course, define multiple decision branches for that single variable: for example, when loading a page, one of several different view states can be invoked using the On Load event that branches depending on the current value associated with the passed variable.

As objects and events are fundamental to creating robust simulations in iRise, we'll be visiting these friends often in the coming recipes: become familiar with their basics before moving on to the recipes that follow by reviewing one or more of the training resources mentioned at the start of this document in the Introduction.

Working with Data

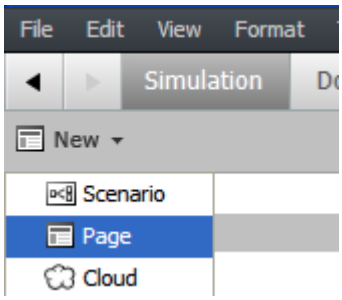
Pass Datasheet Data to a Select Widget

In many projects, multiple project pages expose selection widgets with identical list options. If these lists have many options, the process of recreating the selection values from list to list can be tedious and time-consuming. Of course you can always copy an existing selection widget from one page to another, or defined the widget in a master, but in this exercise we'll be exploring an alternative that leverages iRise

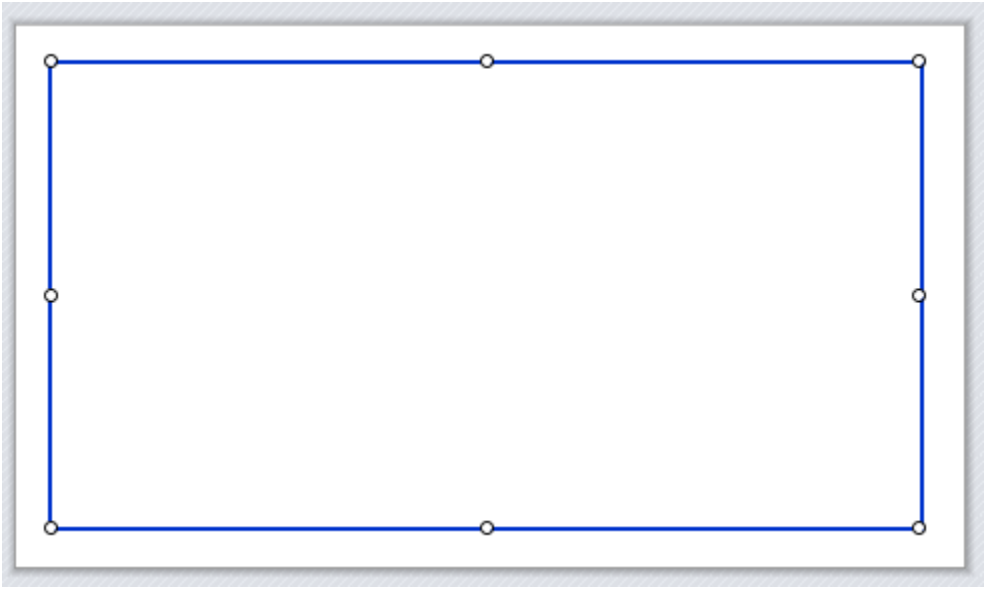
datasheets. In this exercise we'll be using the sample States datasheet that ships with iRise. If you haven't done so already, return to the Introduction and repeat the steps outlined in the Working with This Cookbook section. When you're ready, we'll dive into associating this datasheet with a selection widget.

1. Import the sample States datasheet if you haven't already done so.

2. Create a new page in your project.

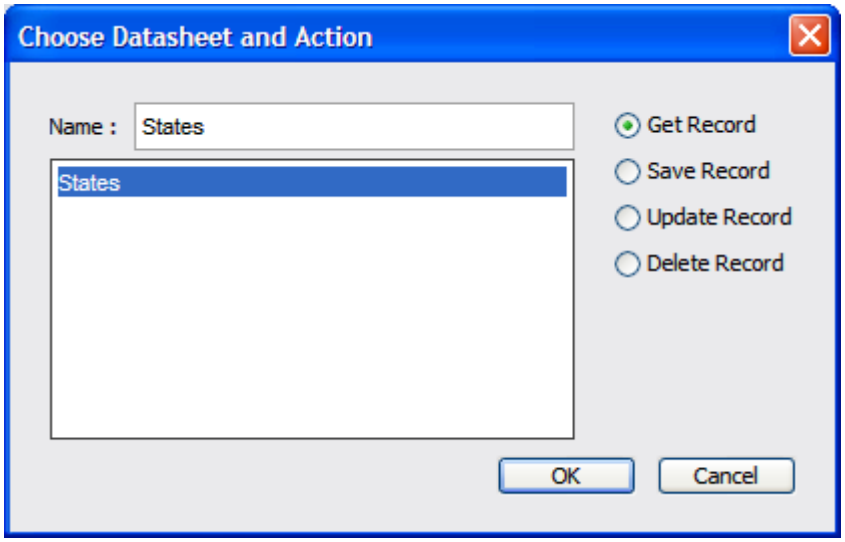


3. Place a form widget onto your page.

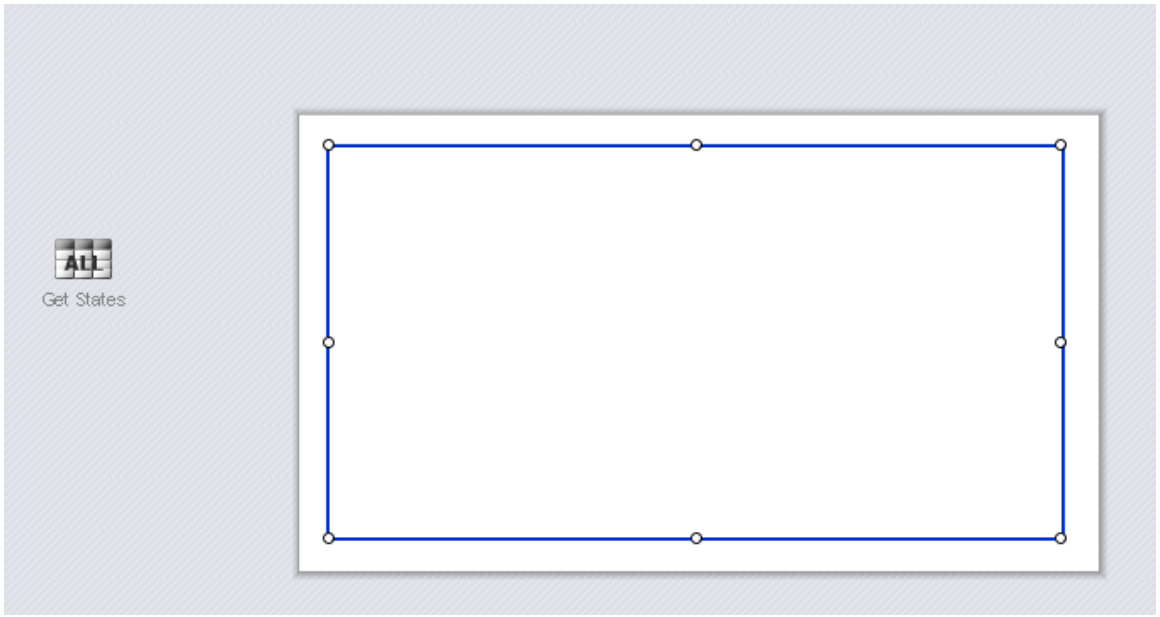


4. From the Studio toolbar, click the Record toolbar icon, then click on the canvas to the left of your page.

5. In the Choose Datasheet and Action dialog box, choose the State datasheet and the Get Record radio button option, then click OK.

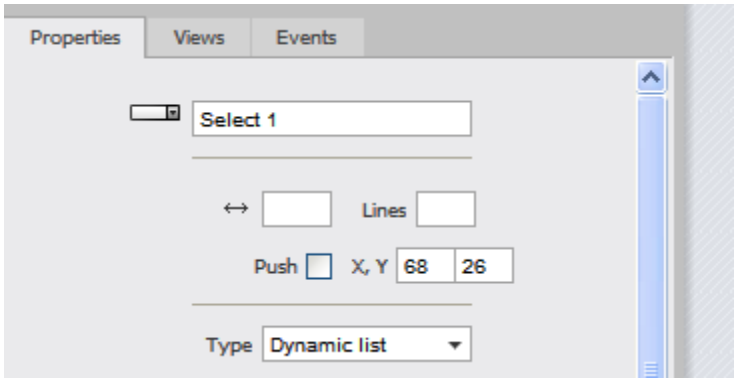


Your working environment should look like the following at this point:



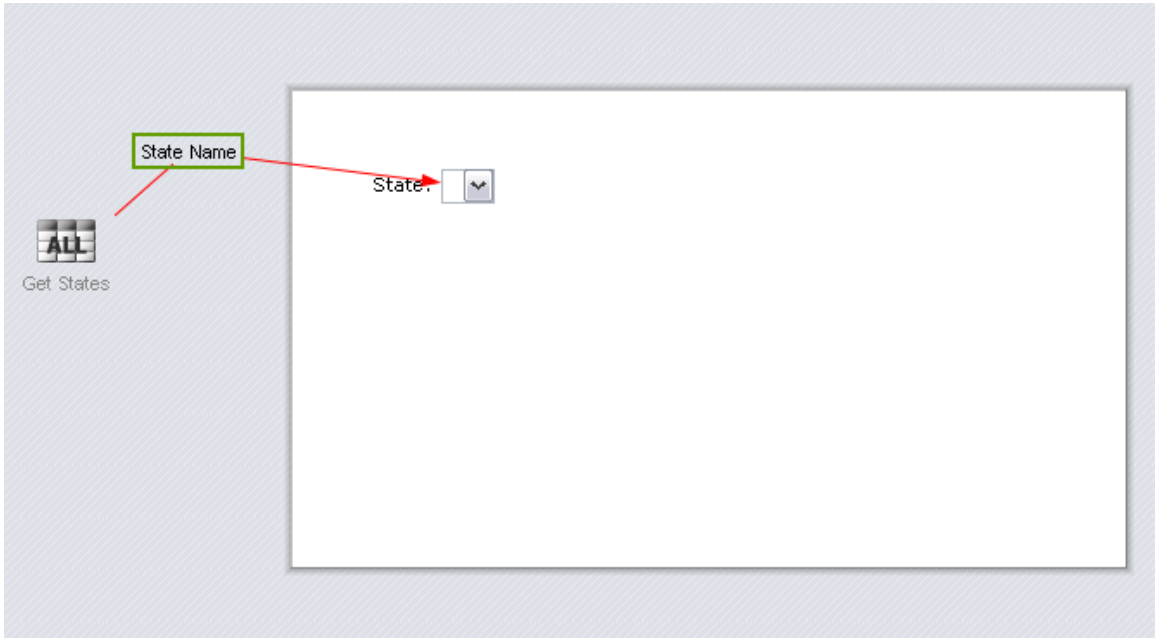
6. Place a text object labeled State and a select widget within the boundary of the form widget.

7. With the select widget selected, change the type to Dynamic List in the Properties Palette.



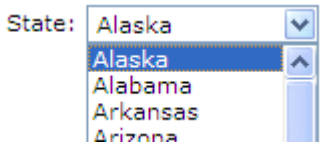
8. Select the Get States record widget and drag it to the drop-down widget to create a linkage between the two.

9. In the Select a Field dialog box, choose State Name, then click OK.



10. Launch your project by clicking the launch icon from within Studio to verify the work done so far.

At this point, your rendered page should contain a functioning drop-down list displaying the full name of each state.



Using Datasheets as Lookup Tables

Oftentimes, data within a simulation needs to be displayed in a variety of ways. For example, sometimes within an application you might want to display the full name of a state, and at other times simply the state abbreviation. The full state name might be exposed in a select widget, but due to space or other constraints it might be valuable to present the user with the abbreviated version in another UI component. In these situations, datasheets can function as lookup tables, where a lookup value can be substituted with another value from a datasheet based on a common lookup key.

In this recipe, we'll be building on the work started in the previous one. If you haven't already worked through those steps, do so before proceeding.

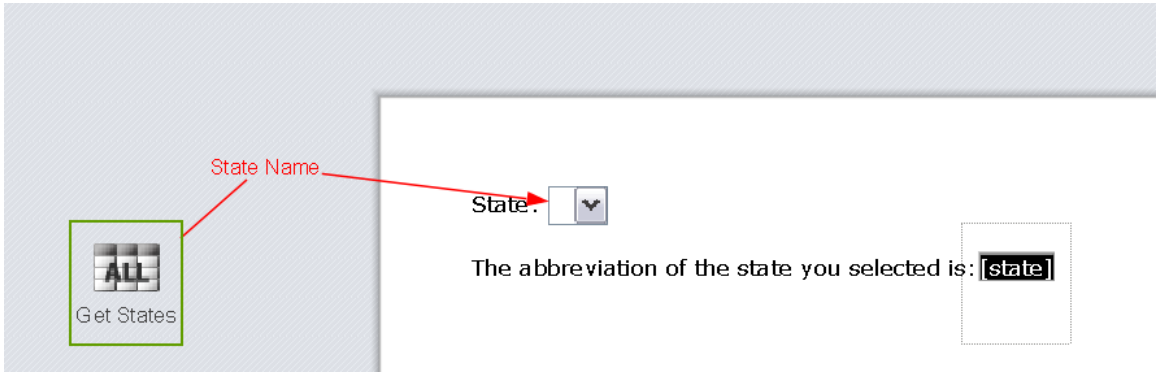
To Use a Datasheet as a Lookup Table:

1. Create a copy of the project drawing containing the state Select widget.

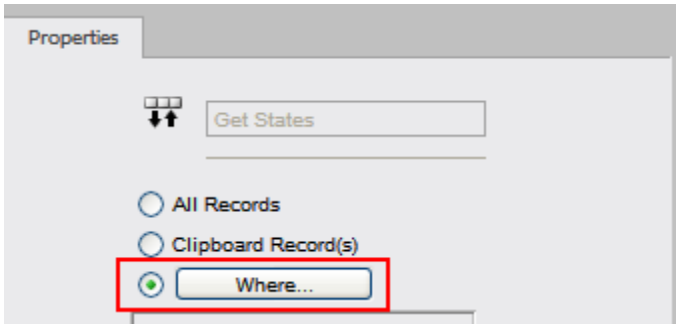
2. In the copied drawing, place a text widget comprised of the following string: "The abbreviation of the state you selected is:"

3. Double-click on the page to the right of the text widget you just created and enter the following string: "[state]".

4. Select the Get States widget, then drag and drop it onto the [state] text widget.

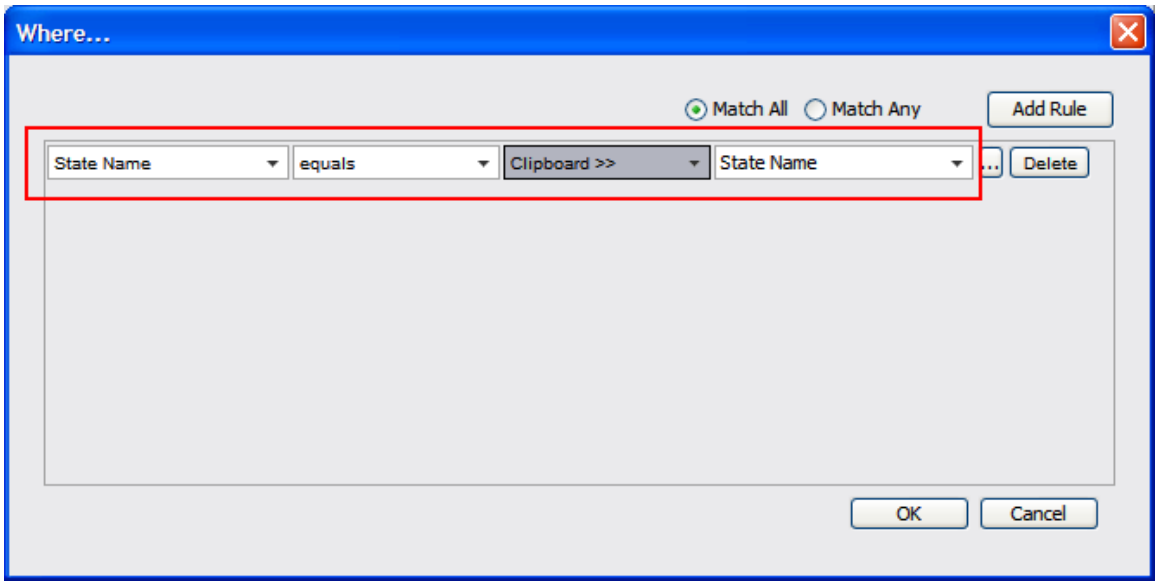


5. With the Get States widget still selected, select the Where radio button from the Properties Palette.



6. In the where dialog box, click the Add Rule button.

7. Using the drop-down controls that are exposed, pacify the following values in the Where dialog box, then click OK:



8. Add a Button widget below the two text strings.

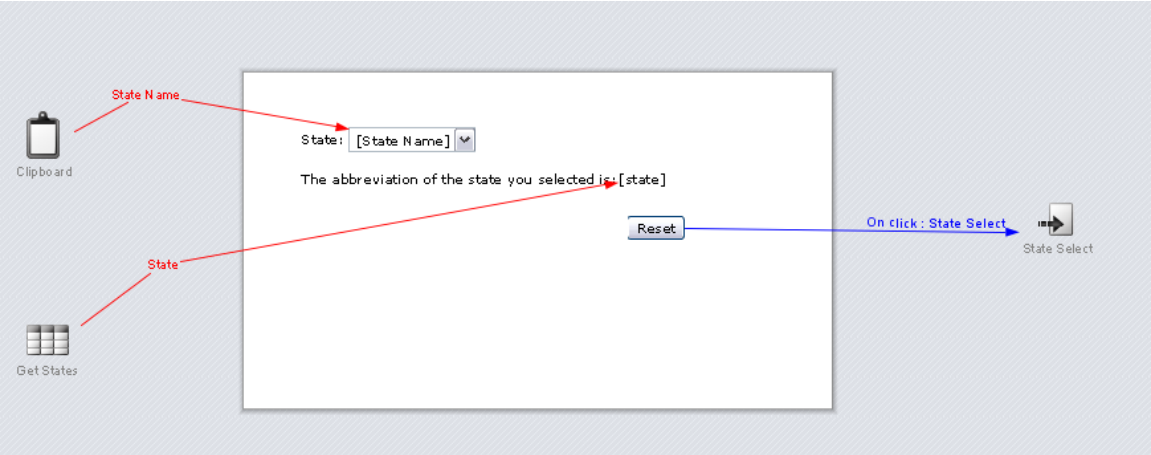
9. Double-click the button to select its label, then type a value of "Reset".

10. Click the Link toolbar button, then click on the right side of the page canvas to place the link.

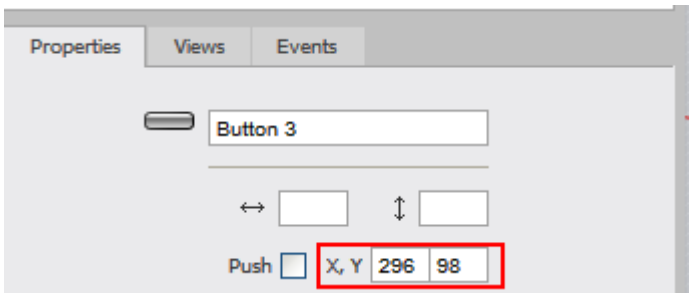
11. In the Set Destination dialog box, select the original page that you copied earlier in step 1.

12. Select the Reset button, then drag and drop it onto the Link widget to establish an On Click event.

    When the following steps have been completed, your drawing canvas should look like the following:



13. Select the button widget on the page, and write down the X, Y coordinate values exposed through the Properties Palette.



We will be using these values to place a button at the exact same coordinate location in the original drawing containing the State Select widget.

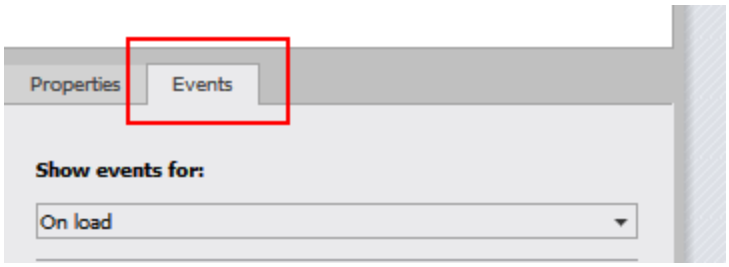
14. Return to the original project drawing you copied in step 1.

15. Place a button widget in the drawing, and with it still selected, use the Properties Palette to enter the X, Y coordinate location you wrote down in step 16.

16. Place a Clipboard widget at the right side of the page and drag and drop the State selection widget to it.

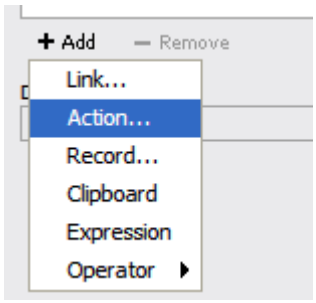
17. From the context menu that is displayed, select State Name from the list, then click OK.

18. Select the State Select widget, then click the Events tab in the Properties Palette.



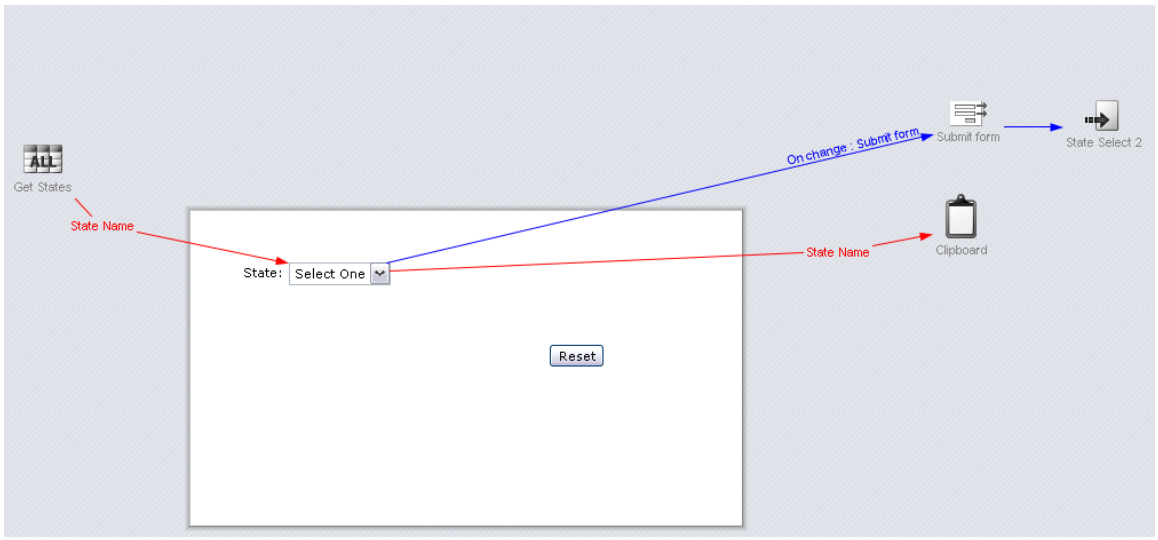
19. In the Show Events For drop-down, select the On Change event, then click the Add button at the bottom of the palette.

20. In the context menu that is exposed, choose Action.



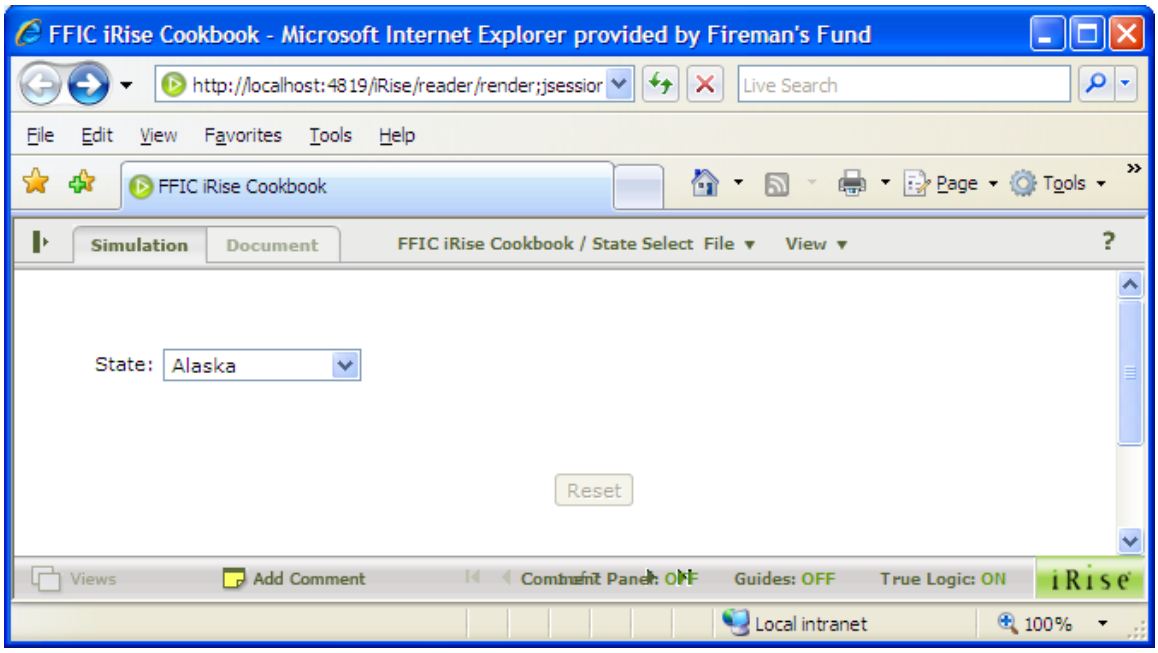
21. In the Set Page Action dialog box, choose the Submit Form action.

22. Place a Link action to the right of the Submit Form action and set it to navigate to your copy of the current page that you created earlier in this recipe.

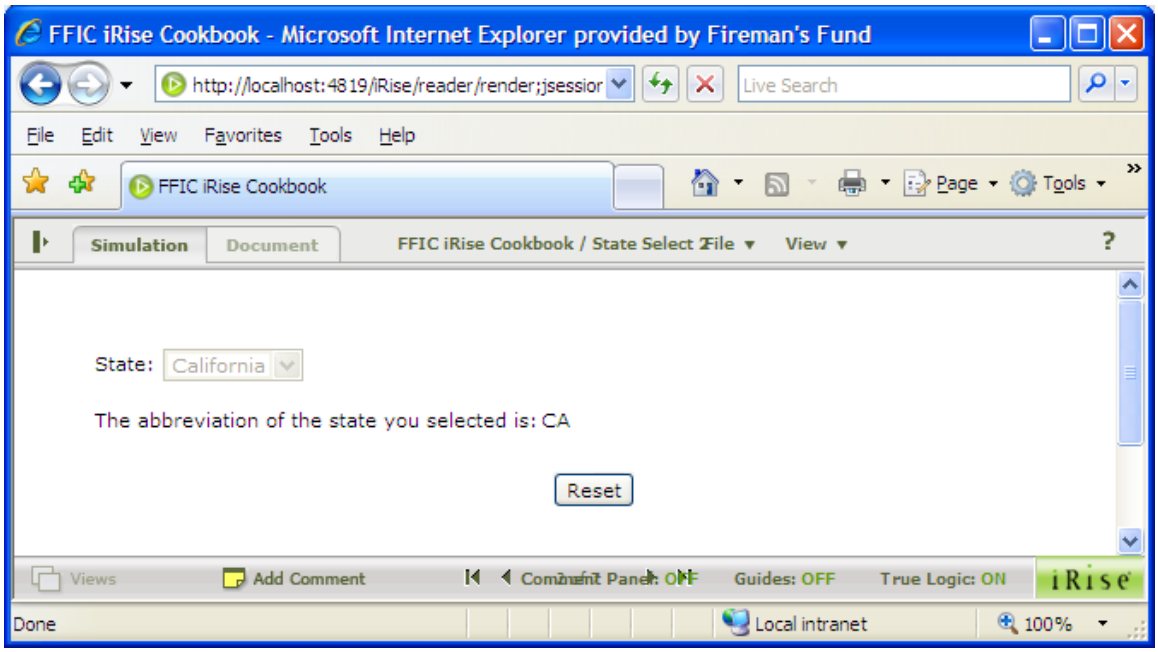


23. Launch your project by clicking the launch icon from within Studio to verify the work done so far.

When initially launched, the page should render with the State selection widget enabled and the Reset button disabled.



When you select a state from the State select widget, the On-Change event is fired, causing the form to submit and capturing the current state selection to the clipboard. The On Change event then navigates to the copy of the current page:



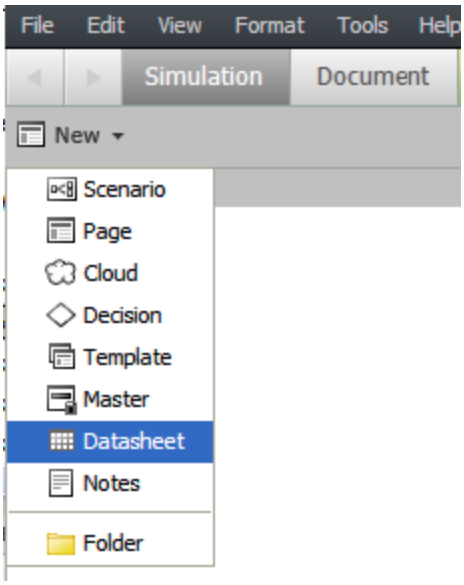
Clicking the Reset button invokes a Link event, navigating you back to the original page and resetting the State selection widget to its default value.

Create a Dynamic Table

Dynamic tables are tables that are populated with external records stored in a datasheet. They are extremely handy for in populating iRise table elements with potentially large numbers of data records, translating into significant time savings in formatting and laying out the table in iRise. Rather than manually inputting each of the values into individual table cells within iRise, you define the records once in either a datasheet or in an Excel spreadsheet. A single datasheet can be linked to multiple data tables in one or more project files, each potentially displaying different sets of data based on Where clause criteria. In this recipe we'll import one of the iRise sample datasheets that ship with the project and create a data table that is populated with a subset of the table columns from the datasheet. In the next recipe, we'll build upon this work by adding a filtering condition that displays just a subset of the records defined in this recipe.

To create a dynamic table:

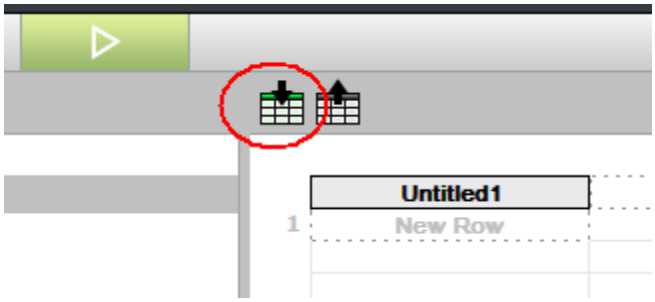
1. From the tree at the left of iRise Studio, choose New > Datasheet.



2. While the new datasheet is still selected, type "Person" to change the default datasheet name.

3. Select the Datasheet you just created from the tree.

4. Click the Import CSV button at the top of the Datasheet window.



5. In the CSV Import From dialog box, navigate to the Sample Datasheets directory below your iRise installation.

   By default, iRise is installed to C:\Program Files\iRise. If you installed to a different directory, the datasheet samples are stored below the root folder as follows: \\ installation directory\Studio\Sample Datasheets.

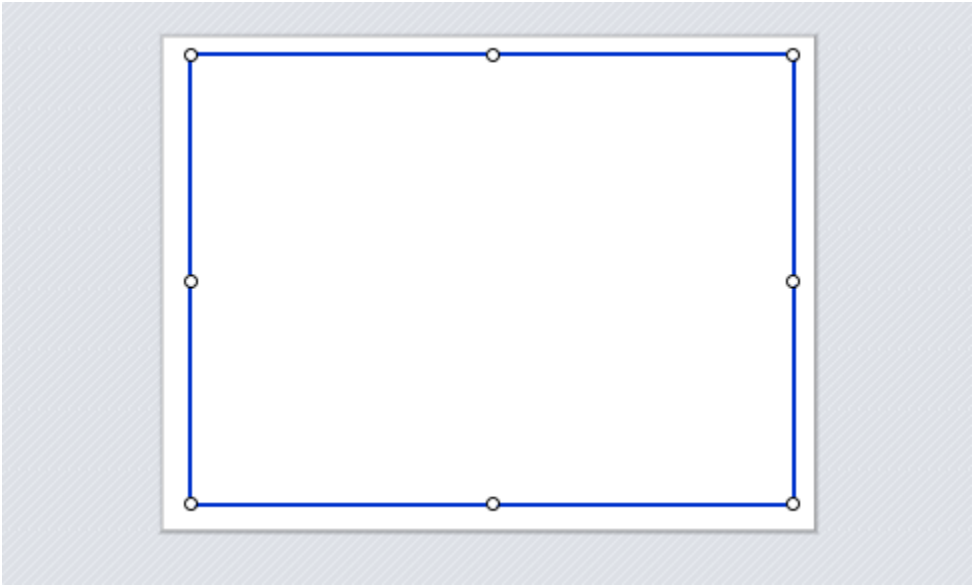
6. Select Person.csv, then click Open.

   This should display a multi-column table within the Studio environment.

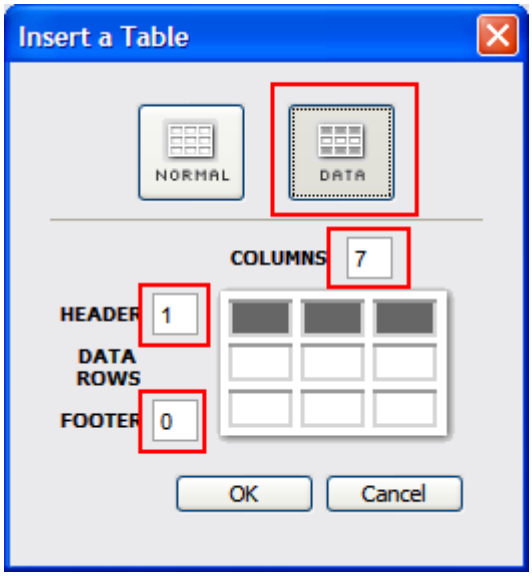
	Id	Username	Password	Title	First Name
1	653659	JWeiss	Tsvyeud5	Mrs.	Jason
2	313826	QBenjamin	Oxewrgp9		Quail
3	293173	SBender	Kqqgrho7	Mr.	Sean
4	302346	NDillon	Xhvqnab7		Norman
5	485816	TKnapp	Xfxedkj3	Mrs.	Tyler
6	381064	CHarmon	Nymridv7	Mr.	Cathleen
7	897738	CFoley	Qzrcdzj5	Dr.	Cassidy
8	285373	JCombs	Lpvjcaw3	Dr.	John
9	110273	AWare	Ltumrcz9	Dr.	Amanda

7. Create a new page in your project named Dynamic Datasheet, then select it from the tree to make it active.

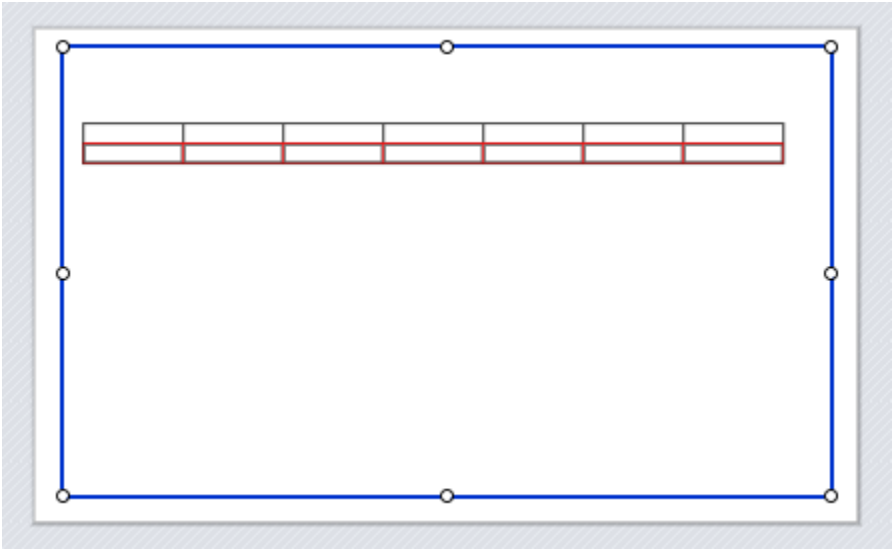
8. Drop a Form widget onto the page, then place a table object within the border of the form.



9. In the Insert a Table dialog box, click the Data button, then enter the following values into the dialog box:
    a. Columns = 7
    b. Header = 1
    c. Footer = 0



10. Click OK.

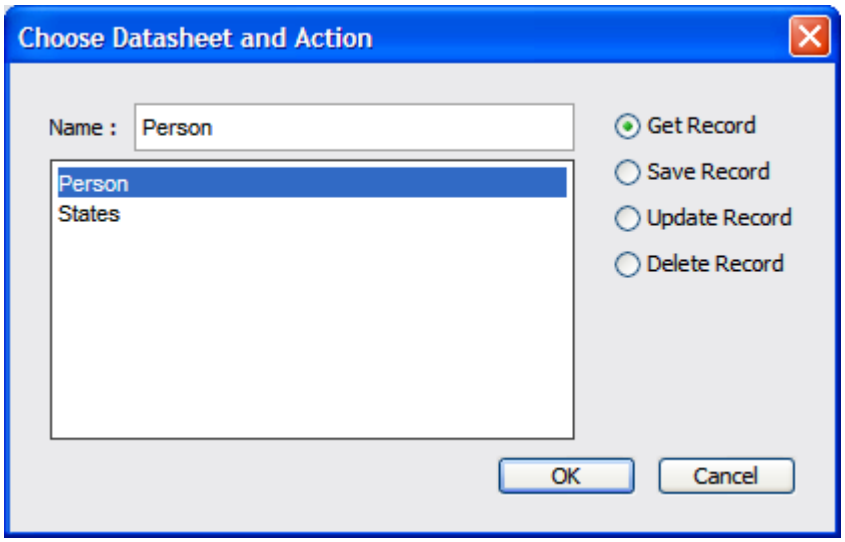


11. Starting with the left-mouse column, double-click each cell in the top row and enter the following values to specify the column labels:
    a. User Name
    b. First Name
    c. Last Name
    d. Address
    e. City
    f. State
    g. Zip

12. Click the Record toolbar button, then click at the left of the drawing canvas to place a record widget.

13. In the Choose Datasheet and Action dialog box, perform the following steps:

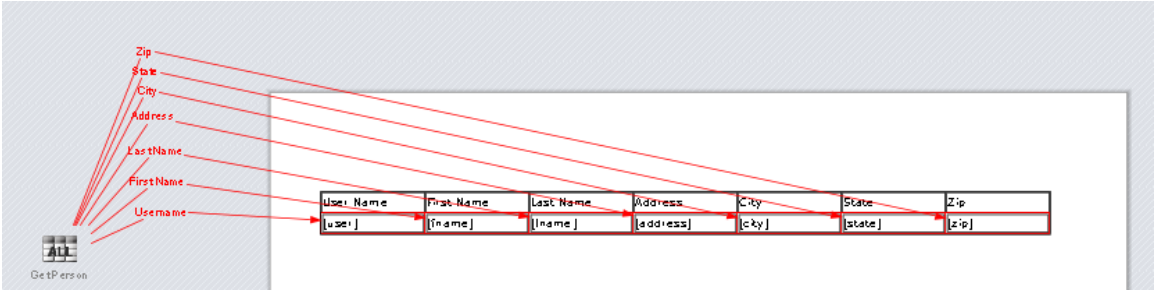
a. Select the Person datasheet from the list.
b. Select the Get Record radio button option.
c. Click OK.



14. In the data table, double-click each cell in the second row and input the following variable placeholder labels:
    a. [user]
    b. [fname]
    c. [lname]
    d. [address]
    e. [city]
    f. [state]
    g. [zip]

    Data rows are visually represented in iRise with a red border to differentiate them from regular tables. Any data row within a simulation is merely a placeholder representation: individual table columns from a datasheet are mapped to individual cells within the data table. When the simulation is rendered in the browser, iRise retrieves all matching records from the datasheet and displays them in the browser. In the next series of steps, we'll be defining the mapping between individual datasheet columns and our data table.

15. Select the Get Record widget from the drawing canvas, and perform multiple drag and drop operations to each data row cell to establish the following data mappings:
    a. Username = [user]
    b. First Name = [fname]
    c. Last Name = [lname]
    d. Address = [address]
    e. City = [city]
    f. State = [state]
    g. Zip = [zip]

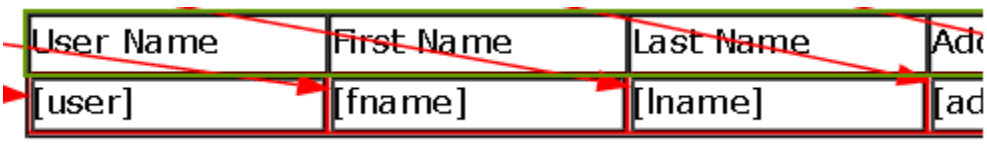


16. Select the Form widget on the page, and make sure the following Properties Palette values are specified:
    a. Expand in Studio = Yes
    b. Horizontal Scroll = No
    c. Vertical Scroll = No

Container objects such as frames and sections all expose these settings. I usually set the Horizontal Scroll and Vertical Scroll options to know (with a few notable exceptions): even when all child objects appear to be comfortably with in the margins of the parent container in Studio, these settings have an annoying tendency to cause scrollbars to show up unexpectedly on various controls when simulations are rendered in the browser.

17. Select the first row (column header row) on the page by holding the cursor at the left-most edge of the row until an arrow cursor appears, then click on the edge of the table.

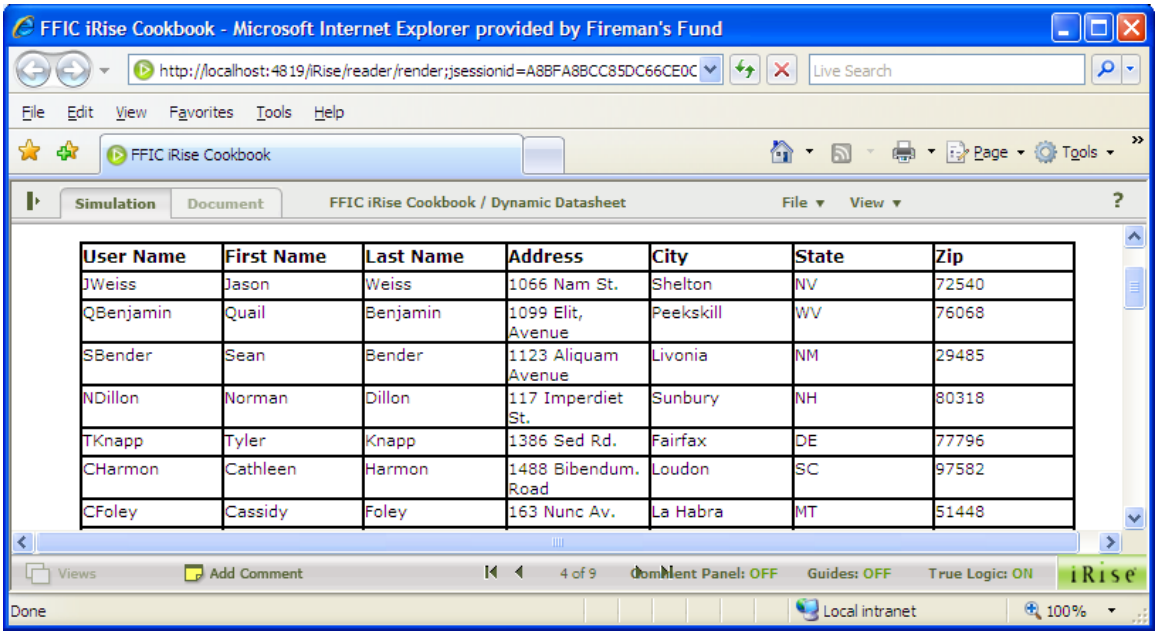
    When the entire row has been successfully selected, it will appear with a green border within the Studio environment:



18. From the Text Toolbar, set the font size to 12 and click the Bold toolbar button.

19. Launch your project by clicking the Launch toolbar icon to verify the work you've done so far.

You should see a table containing multiple data rows, similar to the following:



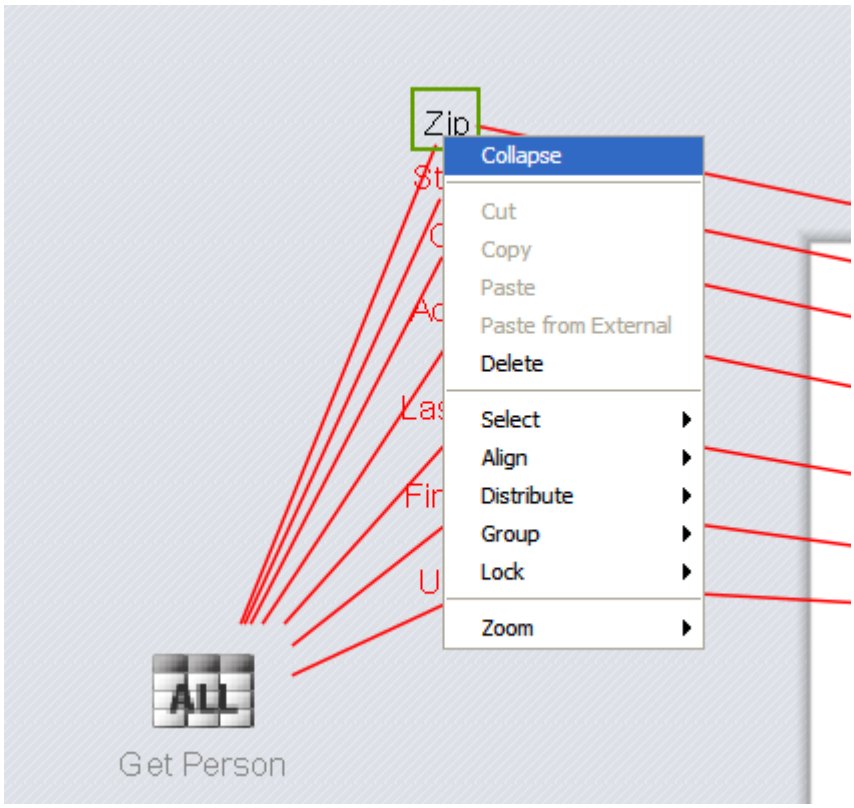
Create a Filter to Display a Subset of the Records Contained in a Data Table

This recipe builds upon the previous one. If you have not already done so, complete the steps in the previous recipe and return to this one when done.

In this recipe, we'll create a selection widget that filters the data displayed in the table to a specific match criterion. We'll be searching for all datasheet records that match the selection state of a State selection list. Any matching records will be returned to the data table. If no matches are found, an alert is displayed. You can extend the simulation if desired to include additional search criteria to match records from a different table column.

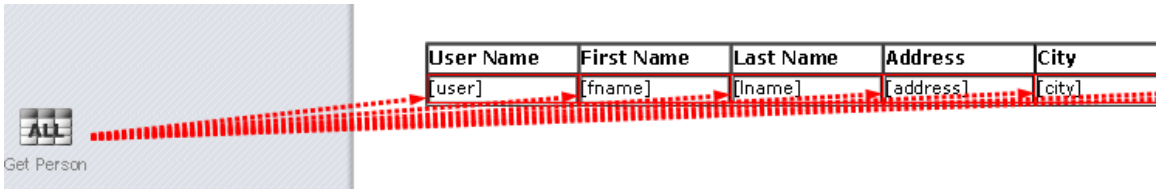
To create a data table filter:

1. Select the label associated with each data line from the Get Record widget to the data table, then choose Collapse.



The Collapse option is handy in temporarily getting rid of some of the temporary clutter on an iRise page, making it easier to work with the underlying page canvas. To revere a collapse operation, select the data lines and choose Expand.

At the completion of this operation, your Studio environment should look like the following:

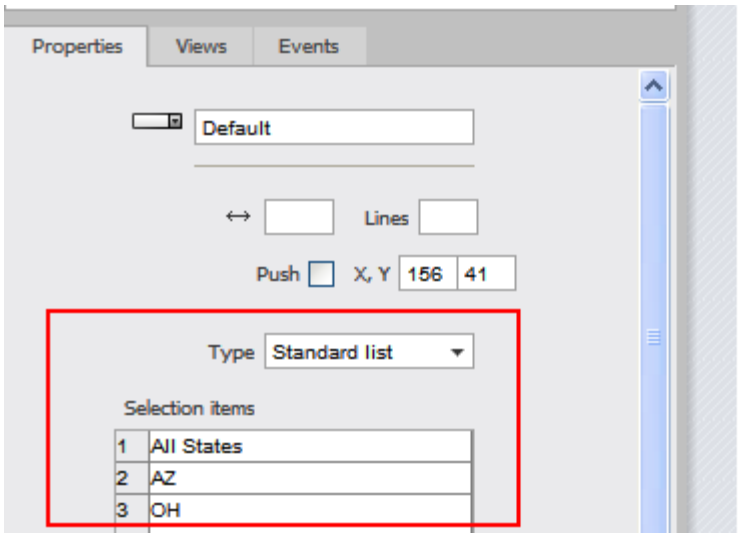


2. Double-click an area above the left side of the data table to insert a text widget and type the following string: "Display records from:".

3. Place a selection widget to the right of the text you just entered.

4. With the selection widget selected, specify the following values in the Properties palette:

    a. Type = Standard List
    b. Selection Items
        i. All States
        ii. AZ
        iii. OH

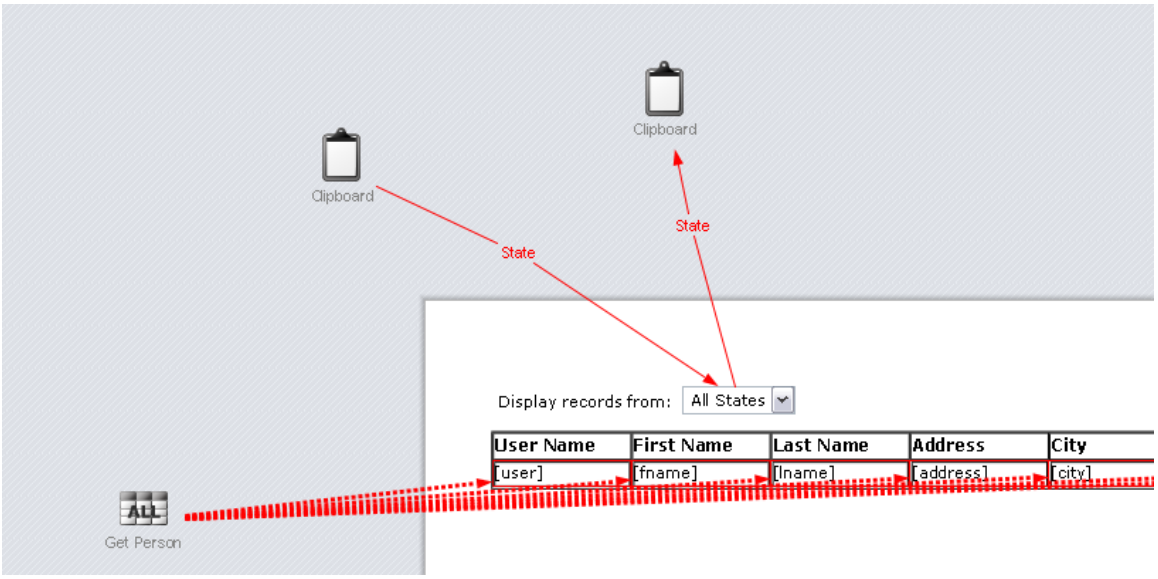


LONG WINDED, OPTIONAL SIDEBAR: You are probably wondering why we aren't using a dynamic list and populating it with values from the State datasheet. The primary reason is that in order to maintain the value selected by the user from the selection list, we need to pass their selection to a clipboard, and then back to the selection list. This unfortunately doesn't work well in this case, as we are already dynamically populating the list with values from a datasheet. Once the user's selection is passed to the clipboard, it clobbers the datasheet list, resulting in a selection with only a single entry available: the value the user just selected. I tried several alternative approaches, including performing a view swap that alternated between the user selection and the full list when the selection list is clicked, but none of the solutions I tried worked quite as expected.

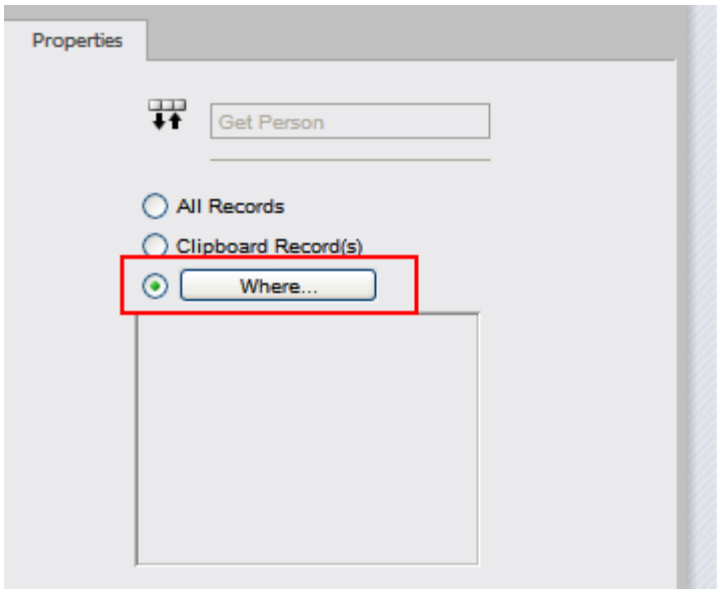
A robust solution that hopefully iRise will consider in the future is allowing users to specify a named variable that already exists in the project as the default value for dynamic selection lists. I eagerly await iris's (hopefully positive) response to this feature enhancement request! Side bar ended; Denko off soap box.

5. Place two clipboards onto the drawing canvas.
6. Drag and drop the first clipboard to the State selection widget.
7. In the Select a Field dialog box, choose State, then click OK.
8. Drag and drop the State selection wizard to the second clipboard.
9. In the Select a Field dialog box, choose State, then click OK.

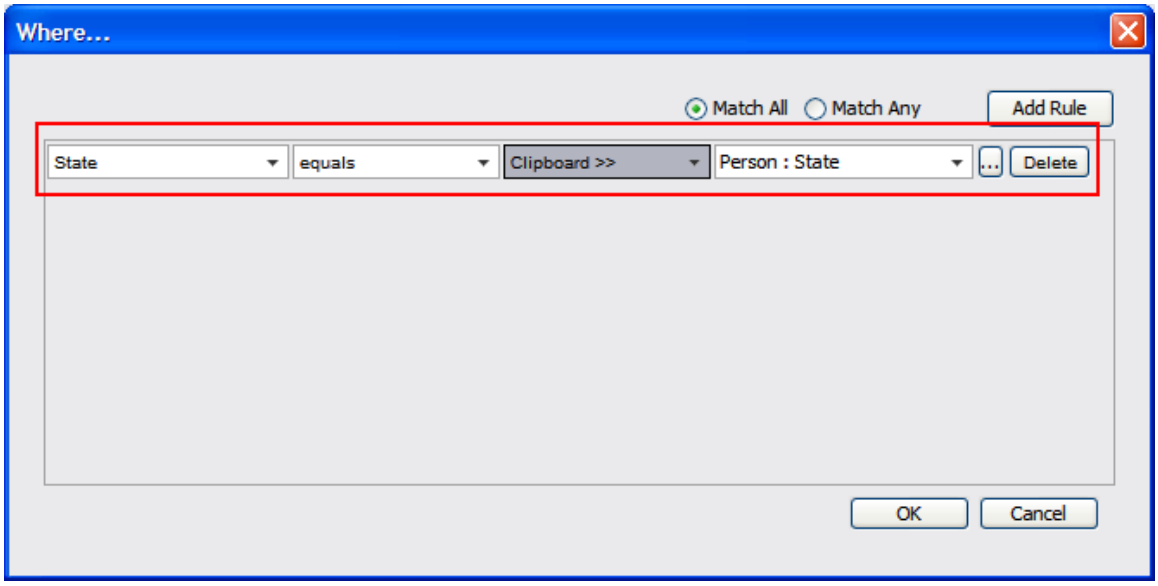
   At this point, your Studio environment should look something like the following:



10. With the State selection widget selected, switch to the Events tab in the Properties Palette.
11. In Show Events For, select On Change, then click the Add button.
12. From the context menu that is revealed, choose the Action option.
13. From the Set Page Action dialog box, choose the Submit Form action.
14. From the project tree, right click the current page and choose Copy.
15. With the current page still selected, right click and choose Paste.
16. Select the pasted page, then right click and choose the Rename option.
17. Type in the following name: "Dynamic Datasheet 2".
18. Select Dynamic Datasheet 2 from the project tree.
19. Select the Get Record widget from the left side of the drawing canvas.
20. In the Properties Palette, select the Where radio button option.

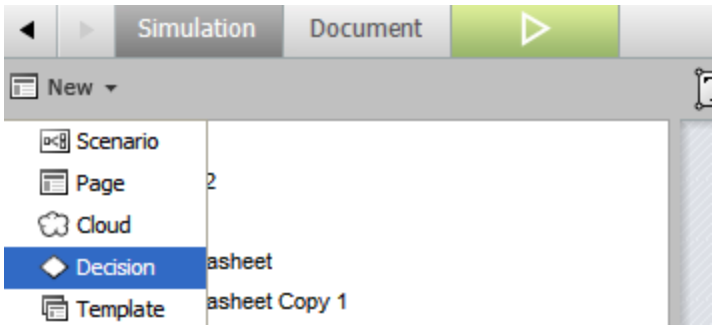


21. In the Where dialog box that is displayed, click the Add Rule button, then specify the following rule:



22. Click OK.

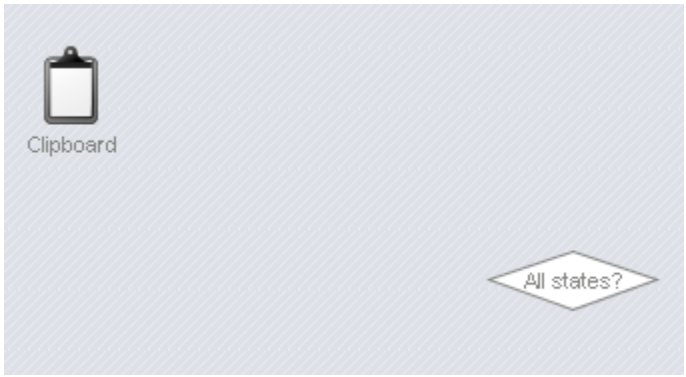
23. Click the New button above the project tree, then choose the Decision option.



24. While the new object is still selected, type "All states?" to rename it.

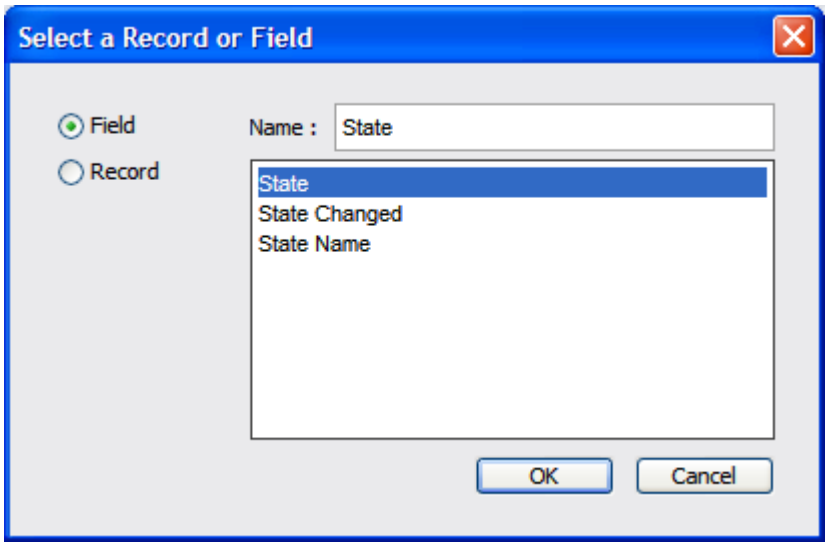
25. Click the All States decision branch to select it.

26. Place a clipboard to the left and slightly above the decision diamond.



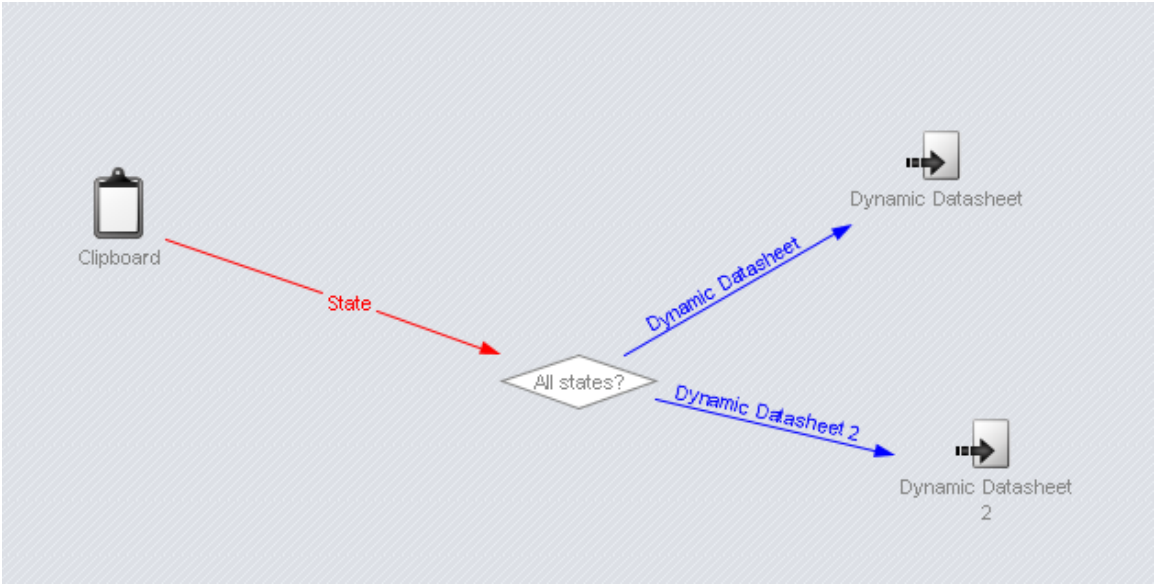
27. Drag and drop the clipboard to the diamond.

28. In the Select a Record or Field dialog box, specify the following values, then click OK:

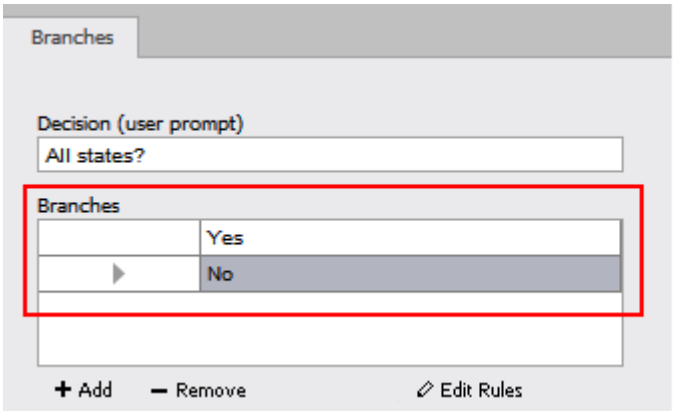


29. Place two link widgets to the right of the diamond, pointing to the following pages:
    a. Dynamic Datasheet
    b. Dynamic Datasheet 2

30. In two separate operations, drag and drop the diamond to each of the page links in turn.

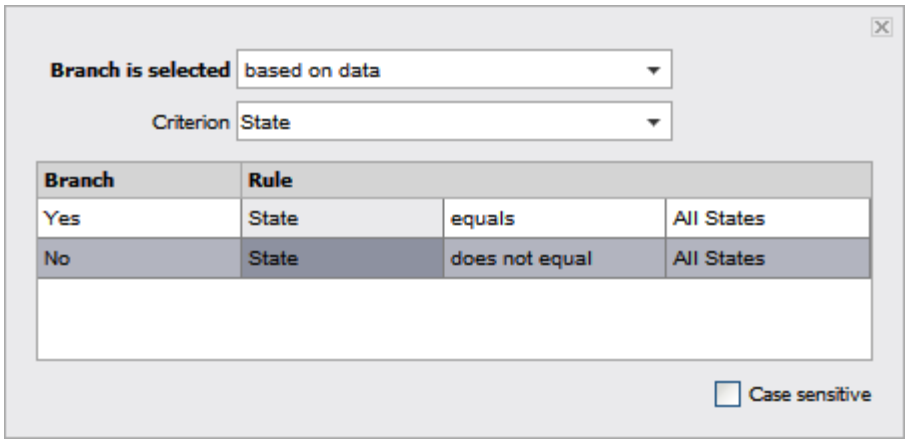


31. In the Properties Palette, double click each listing in the Branches section in turn to rename the default values as follows:
    a. Dynamic Datasheet: "Yes"
    b. Dynamic Datasheet 2: "No"



32. From the Properties Palette, click the Edit Rules button.

33. In the dialog box that is displayed, enter the following settings, then click the x at the upper right of the dialog box to dismiss it:



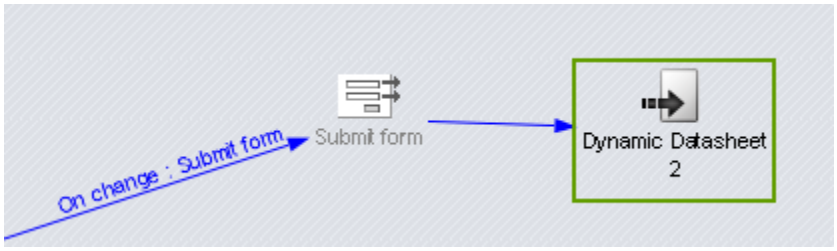
This sets up the rules for the decision branch: if the user selects the All States entry from the State selection dialog box, she will be returned to the Dynamic Datasheet page. If she selects any other option, she remains on the Dynamic Datasheet 2 page.

We're now ready to hook up the last couple of pieces of logic to our recipe and give it a taste.

34. From the project tree, select the Dynamic Datasheet page.

35. To the right of the Submit Form action, place a link pointing to Dynamic Datasheet 2.

36. Drag and drop the Submit Form widget to the Dynamic Datasheet 2 page link to establish a connection.

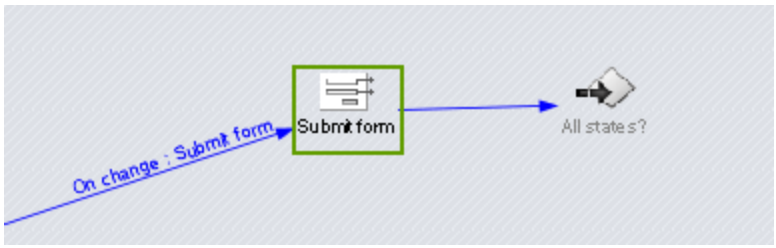


This completes the page logic for the first page: when the user changes a value from the State selection control, the form is submitted (which captures the value selected from the State list) and data flow is passed to Dynamic Datasheet 2.

37. From the project tree, click Dynamic Datasheet 2 to open that page in Studio.

38. To the right of the Submit Form action, place a link pointing the All States decision.

39. Drag and drop the Submit Form widget to the All States link to establish a connection.



This step completes the logic for the second page: when the user changes the page logic for the second page, the decision branch is invoked. If the user selected the All States option, she is returned to the initial page. If another option is selected, the user remains on the current page.

40. Select Dynamic Datasheet from the project tree.

41. Launch your project in your browser by clicking the Launch toolbar icon to verify the work you've done so far.

Try toggling through the various options exposed through the State selection widget and observe the filtering that occurs.

Save a Record to a Data Sheet

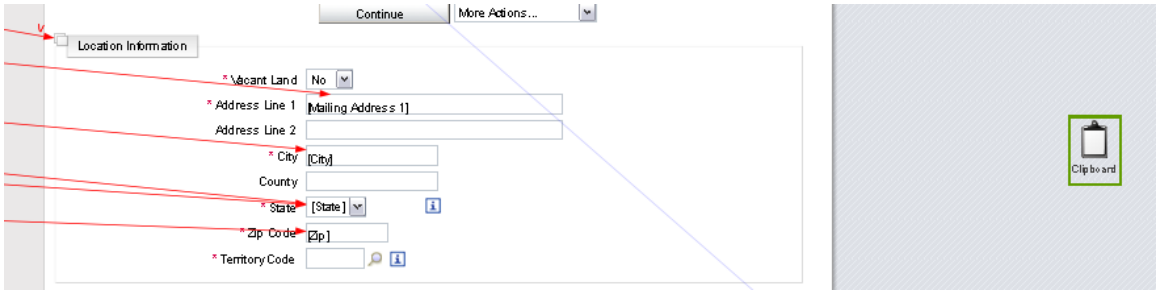
Datasheets are particularly useful in persisting user preferences/settings across multiple editing sessions. Whereas clipboard data is only persistent across the lifetime of a given browser session, datasheet records are maintained indefinitely until the user decides to delete them. In this recipe, we'll be creating a simple simulation that accepts user input and saves it off to a datasheet for future recall. The next series of recipes will share a common theme, borrowed from the Web Solutions ABC quoting tool. We'll cover all components of a data record lifecycle: creating a new record; copying the record to use as a starting point in creating a new record; updating an existing record; and deleting a record. This recipe will cover the initial step of creating a record and saving it to a datasheet.

Rather than detail the various steps for building out the basic page elements, we'll be starting with a precooked page that contains the basic form elements, and simply be building out the logic to hook up the required datasheet functionality. The materials needed for the next three recipes can be found on the following shared drive location:

1. If you haven't already, download the starting files from the Design intranet.

2. From the project tree, navigate to and select the Locations page.

3. Place a clipboard widget to the right of the Location Information section.



4. Drag and drop the following widgets in the Location Information section to the clipboard, and through the Select a Field dialog box, specify the following data mappings for each control:

    - Vacant Land selection widget = Vacant
    - Address Line 1 text input widget = Mailing Address 1
    - City text input widget = City
    - State selection widget = State Name
    - Zip Code text input widget = Zip
    - Territory Code text input widget

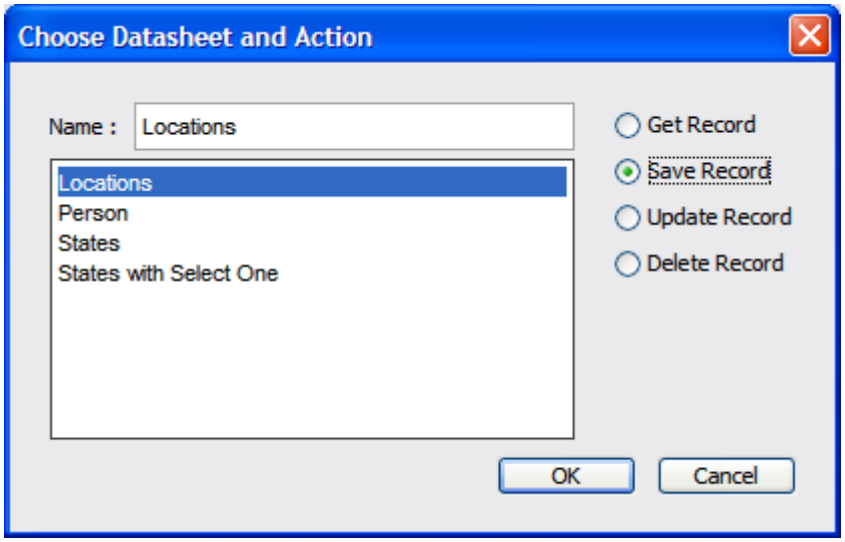
    At the completion of this operation, your Studio environment should look like the following:

    In this step, we've established data mappings for all required fields on the page (those indicated by red asterisks). Mappings to the other field in this section are not specified (mainly in the interest of keeping steps in this recipe to a minimum, and to reduce clutter in the drawing to keep editing simpler).

> **OPTIONAL STEP FOR EXTRA CREDIT**: Establish mappings for the additional controls in this section using the following variable values:
> - Address Line 2 text input widget = Mailing Address 2
> - County = County
>
> Hint: To complete this step, remember that you need to pass data both to and from the controls.

5. Place a record widget to the right of the clipboard you placed earlier in this recipe.

6. In the Choose Datasheet and Action dialog box, specify the following values:

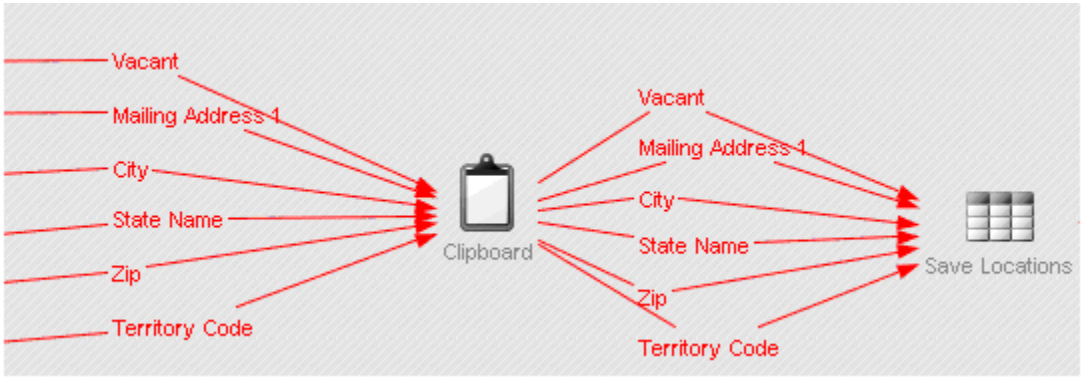


7. Drag and drop the clipboard to the save record widget you just placed, then choose the Send Data context menu option.

8. In the Select a Field dialog box, choose the Vacant variable.

9. Repeat steps 7 and 8, establishing data mappings between all required variables except for State.

   At the completion of this step, your Studio environment should look like the following:



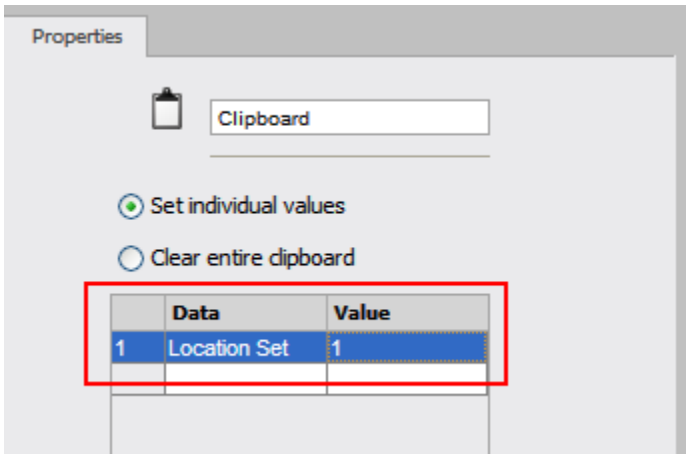
   In the next series of steps, we'll associate an action with the Add button that submits the form, thus saving the state of all data flows, including outputting the values captured from the clipboard to the datasheet via the save record widget.

10. Place a Submit Form widget to the right and slightly above the Add button on the drawing canvas.

11. Drag and drop the Add button to the Submit Form widget to establish an On Click event relationship.

12. To the right of the Submit Form action, place a Clipboard.

13. Select the clipboard and enter the following values into the Properties Palette:

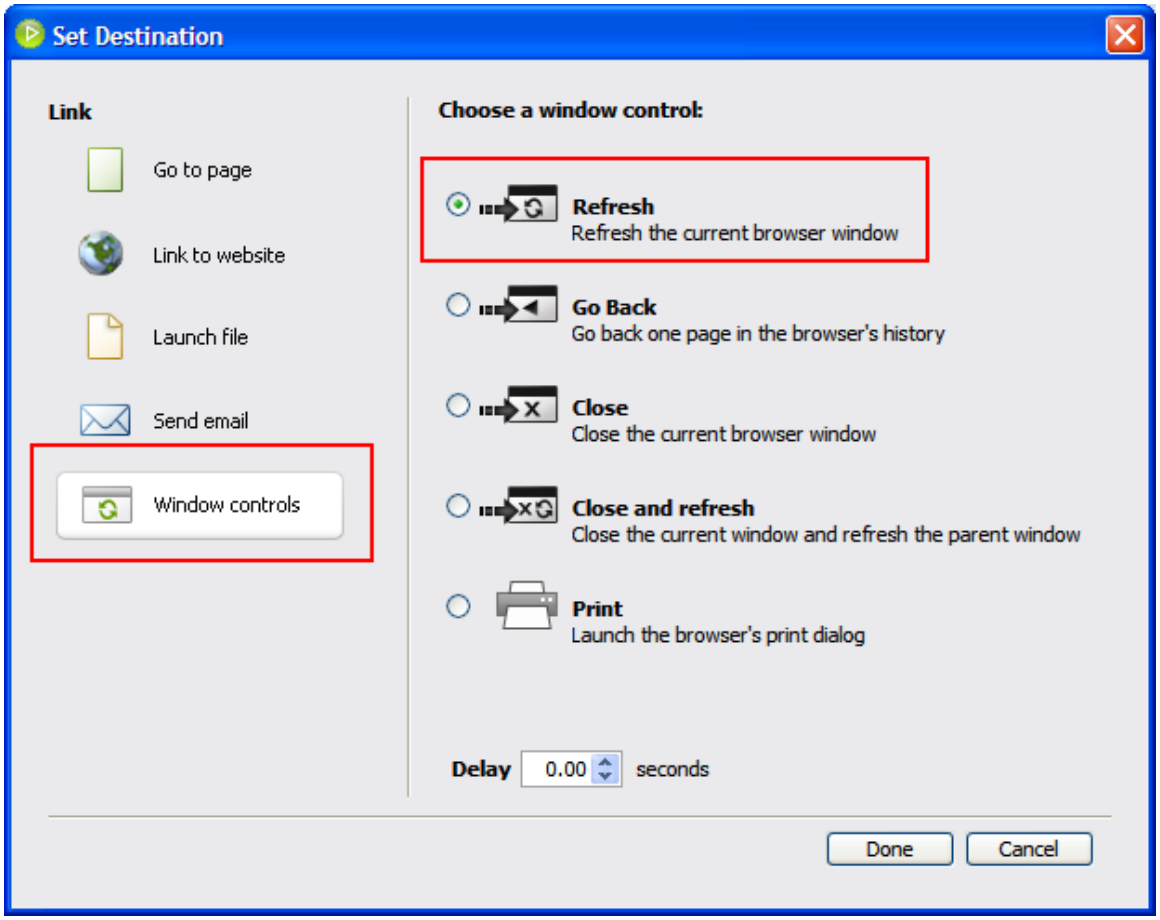


We will be using the Location Set variable to control some of the view logic already established on the page.

14. Drag and drop from the Submit Form action to the clipboard to establish a relationship.

15. Place a Link widget to the right of the Clipboard you placed in the previous steps.

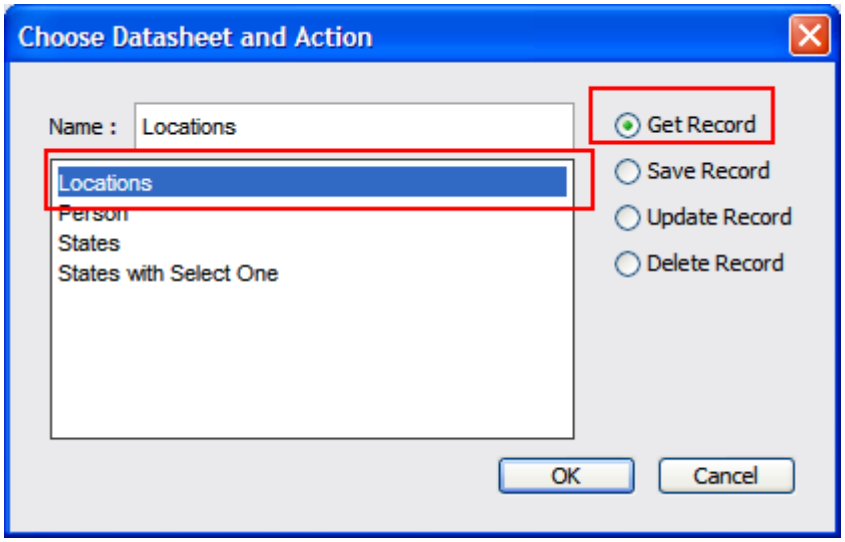
16. In the Set Destination dialog box, specify the following values:



The Refresh action reloads the current browser window. Since we're saving off our current data state via the Submit Form action, any captured data will be refreshed and updated in the page when it is reloaded.

In the next series of steps, we'll be adding a Get Record widget to retrieve all records from the Locations datasheet, and establishing mappings between predefined variable placeholders in the Location table at the top of the page.

17. Place a Record widget at the left of the page near the Location table, and specify the following values in the Choose Datasheet and Action dialog box:



18. Click OK to dismiss the dialog box.

19. Drag and drop the Get Record widget to the [Address] text element in the Location table.

20. From the Select a Field dialog box, choose Mailing Address 1.

21. Repeat steps 19 and 20 to define mappings between the following Location table variable placeholders and their corresponding Locations table equivalents:

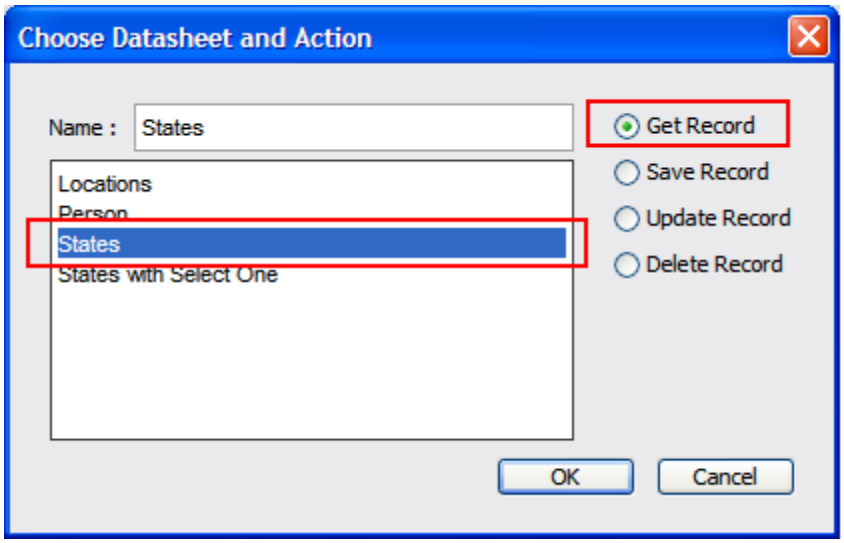
    - [City] = City
    - [State] = State
    - [Zip] = Zip

    At the completion of this operation, your Studio environment should look similar to the following:

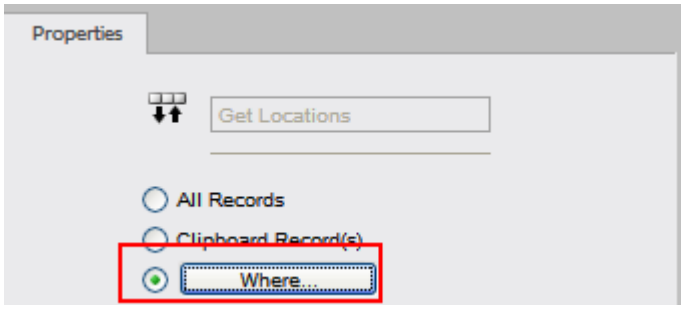
Our final task in completing this recipe is to pass the State name from our data clipboard to a Get Record action, retrieving the abbreviation associated with the state name.

22. Place a Record widget above the Save Locations widget.

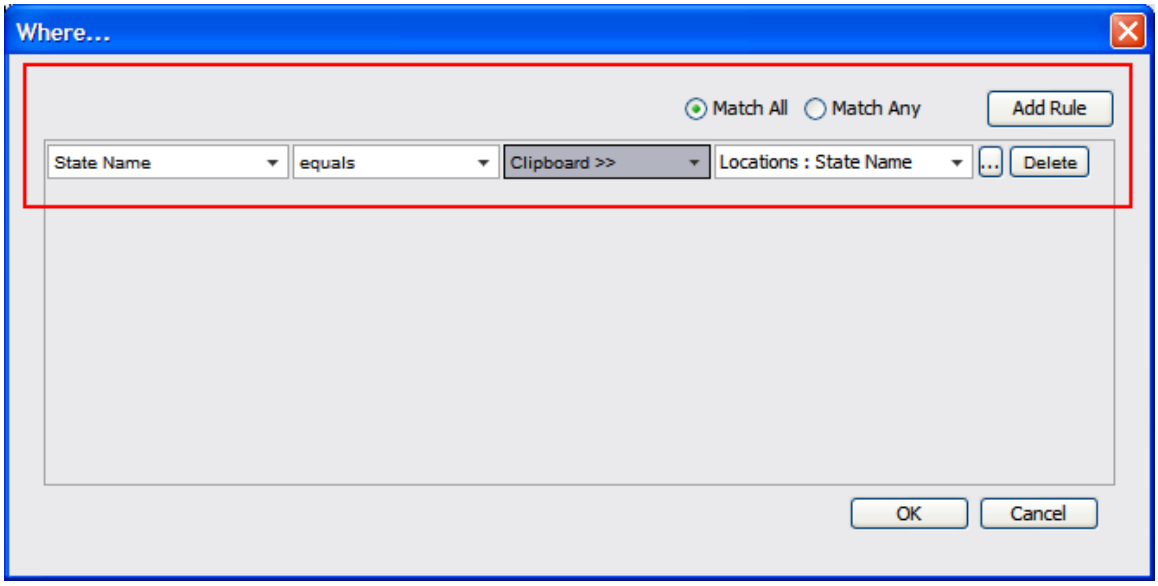
23. In the Choose Datasheet and Action dialog box, specify the following values, then click OK:



24. With the Get Locations widget select, click the Where button from the Properties Palette.



25. In the Where dialog box, confirm the following settings, then click OK:



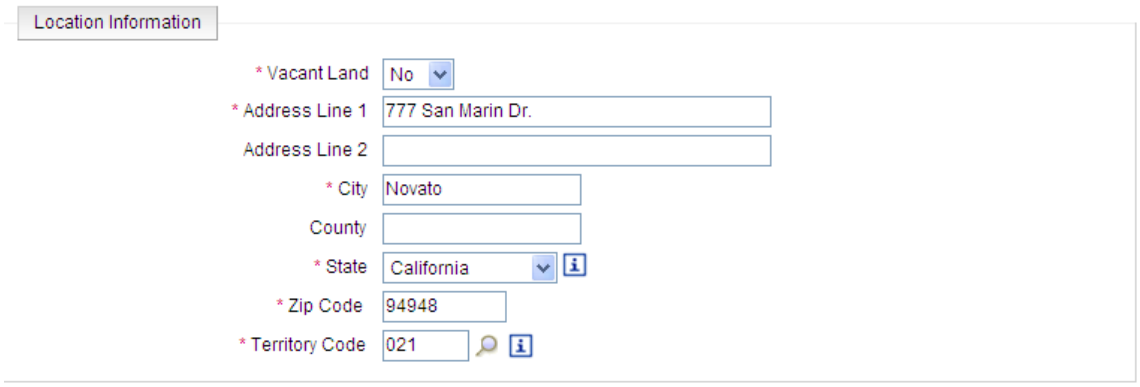
26. Drag a drop the Get Locations widget to the Save Locations widget.

27. From the context menu that's exposed, choose the Send Data option.

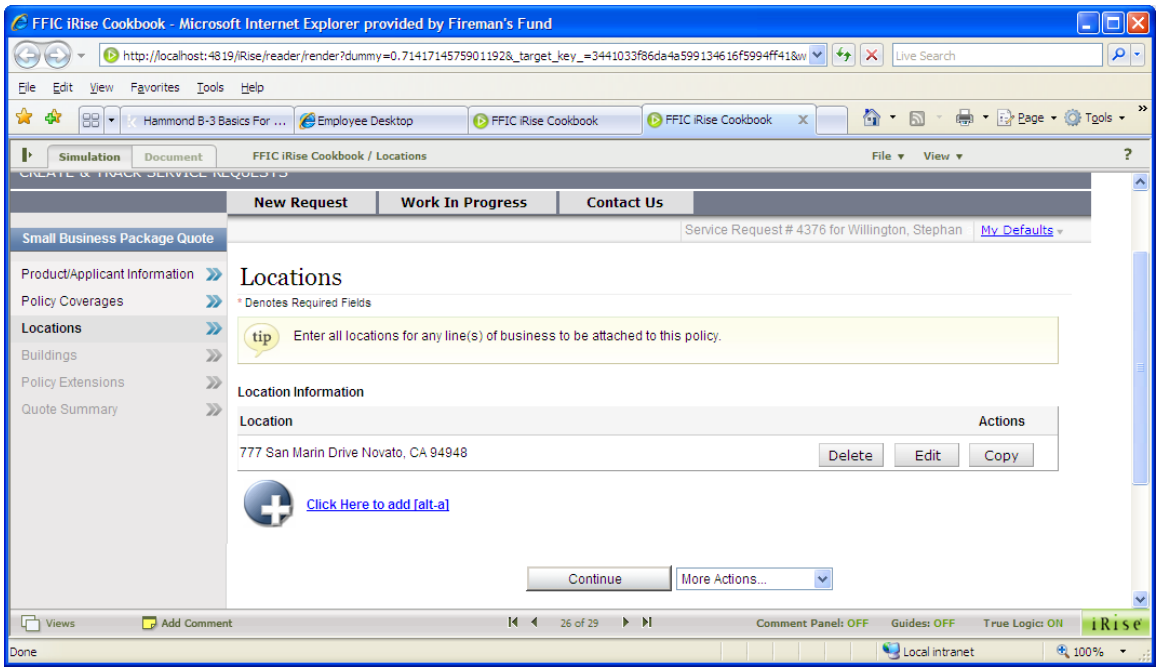
28. From the Select a Field dialog box, choose State, then click OK.

29. Launch your project in your browser by clicking the Launch toolbar icon to verify the work you've done so far.

30. Enter the following values in the Location Information section, then click the Add button.



If you have everything connected properly, this should submit the form and save the data you entered to your datasheet, initiating a view changed based on the On Load event associated with the Page widget based on the Location Set value we passed to the clipboard connected to the submit form action associated with the Add button.



Select the Locations datasheet from the project tree to verify that the data has successfully been saved to your datasheet.

	Mailing Address 1	City	State Name	Zip	Territory Code
1	777 San Marin Drive	Novato	California	94948	021
2	New Row				

Update a Datasheet Record

This recipe builds upon the preceding one. Again, we'll be starting with some prebaked elements. The primary difference between saving a record vs. updating a record is the type of record widget used. As many of the steps in updating a datasheet are the same as for saving a datasheet, we'll be skipping many of the steps from the preceding exercise.

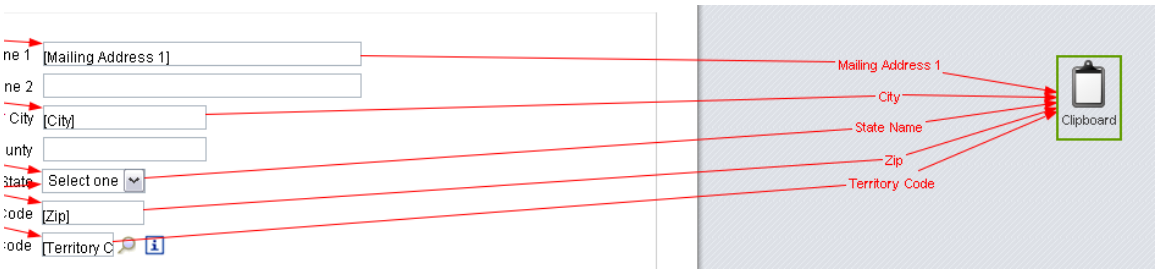
1. From the project tree, navigate to and select the Edit Location page.

2. Place a clipboard at the right side of the drawing canvas near the Location Information section.

3. Drag-and-drop the following fields from the Location Information section to establish the following data mappings:

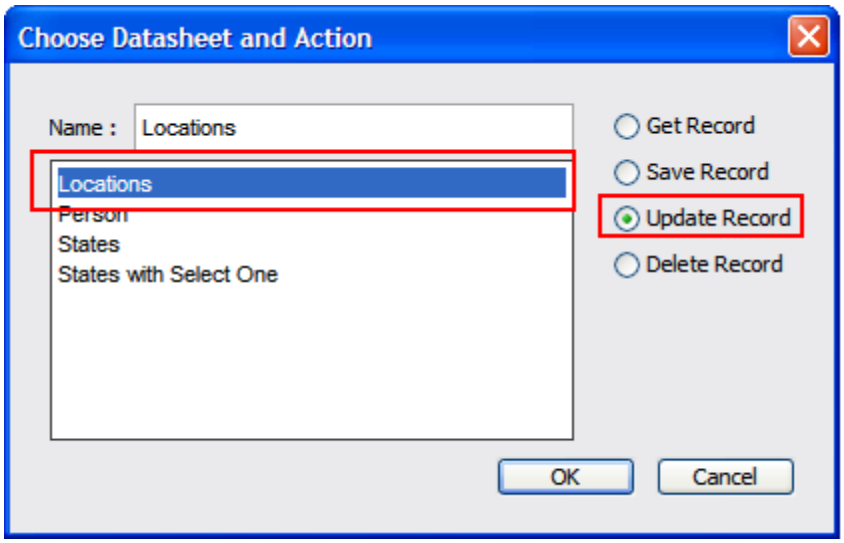
    a. Address Line 1 = Mailing Address 1
    b. City = City
    c. State = State Name
    d. Zip Code = Zip
    e. Territory Code = Territory Code

At the completion of this operation, your page should look like the following:



4. Drag-and-drop a record widget from the toolbar to the drawing canvas.

5. In the Choose Datasheet and Action dialog box, specify the following settings, then click OK:



6. Drag-and-drop the clipboard widget you placed in step 2 to the update record widget you just created.

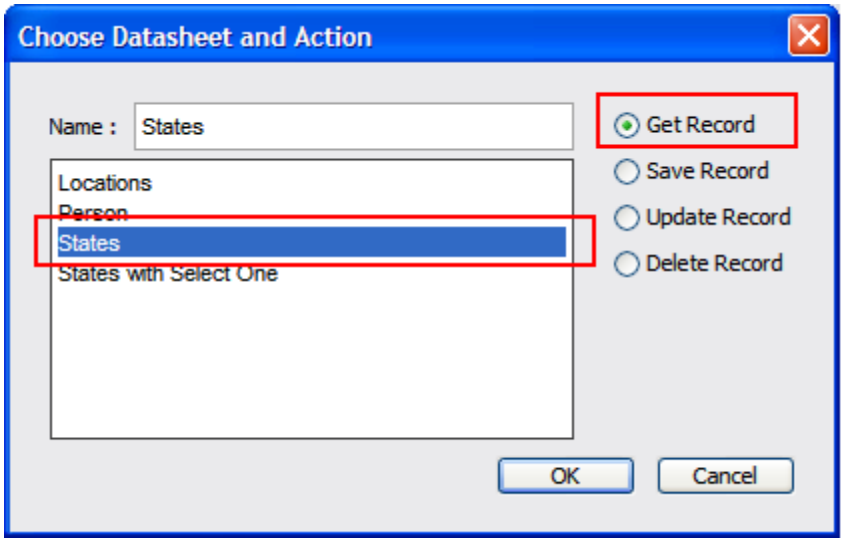
7. From the context menu that is displayed, choose Send Data.

8. In the Select a Field dialog box, choose Mailing Address 1.

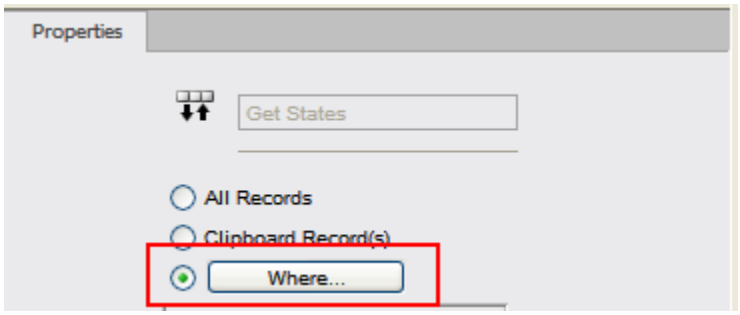
9. Repeat steps 6 through 8 to establish the following additional data mappings:
    a. City
    b. State Name
    c. Zip
    d. Territory Code

10. Drag-and-drop a record widget to the drawing canvas.

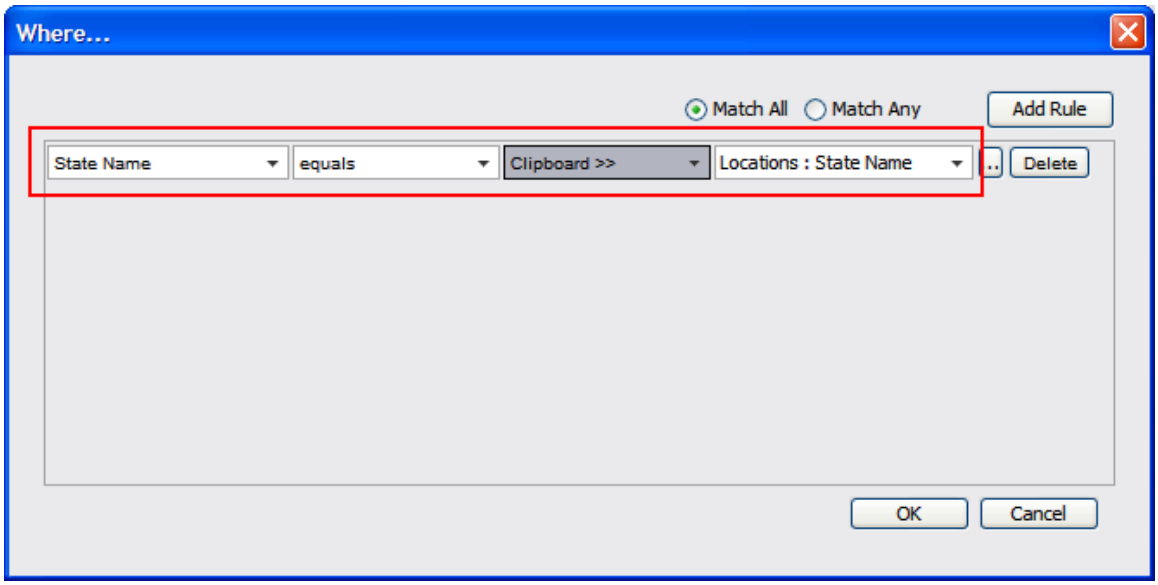
11. In the Choose Datasheet and Action dialog box, specify the following settings, then click OK:



12. With the Get States record widget you just placed still selected, click the Where button from the Properties Palette.



13. In the Where dialog box, click the Add Rule button, then specify the following settings:



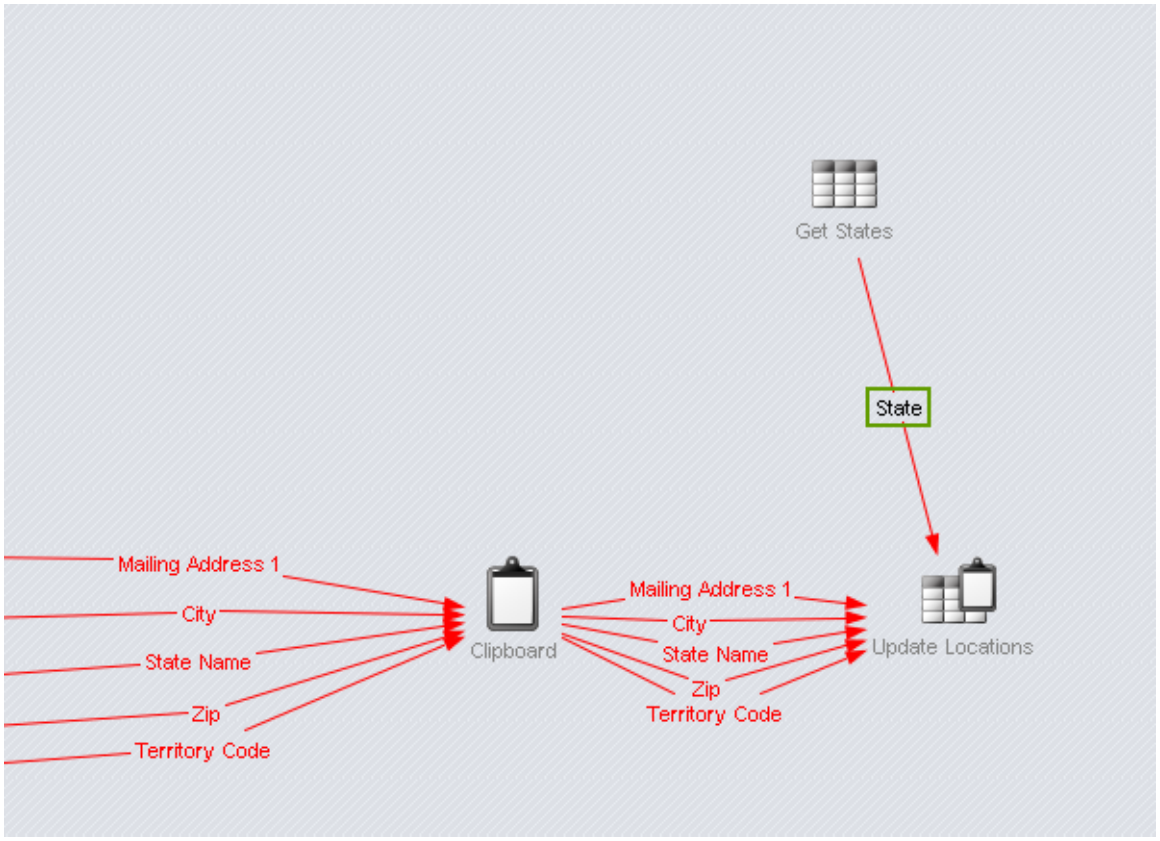
14. Click OK.

15. Drag-and-drop the Get States widget to the Update Locations widget.

16. From the context menu that is exposed, choose Send Data.

17. From the Select a Field dialog box, choose State.

    At the completion of this operation, your drawing canvas should look like the following:



18. Place a Submit Form widget to the right of the Save button.

19. Drag-and-drop the Save button to the Submit Form widget you just placed.

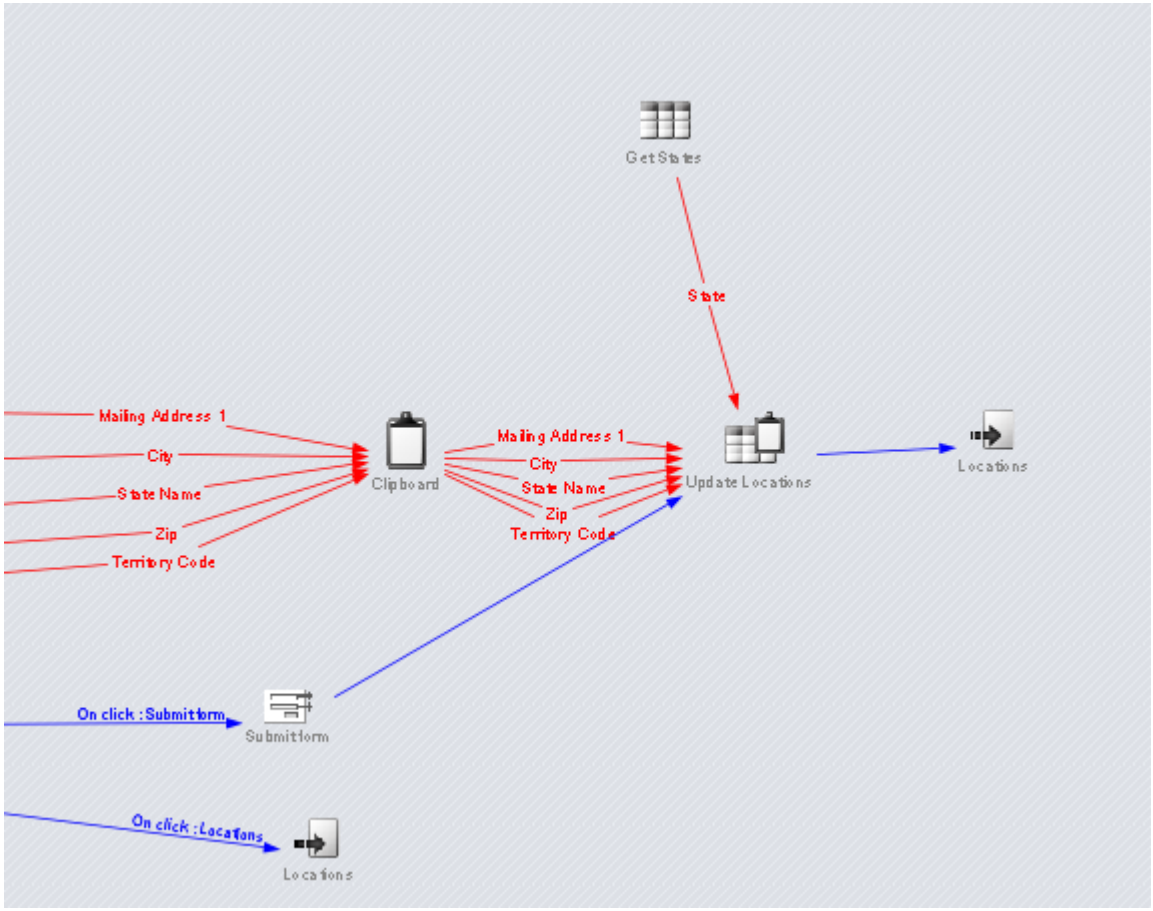
20. Drag-and-drop the Submit Form widget to the Update Locations record widget.

21. To the right of the Update Locations widget, place a Link widget.

22. In the Set Destination dialog box, choose the Locations page, then click Done.

23. Drag-and-drop the Update Locations widget to the Locations link you just created.

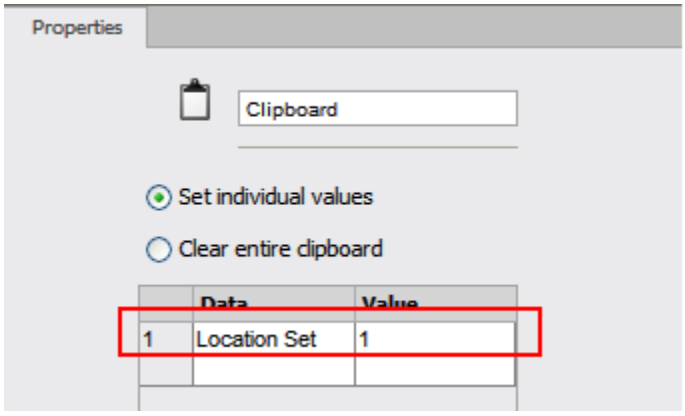
At the completion of this operation your page should look like the following:



24. From the project tree, choose the Locations 2 page.

25. Place a clipboard to the right of the Edit button.

26. With the clipboard widget selected, specify the following value in the Properties Palette:



27. Place a link widget to the right of the clipboard you placed in step 25.

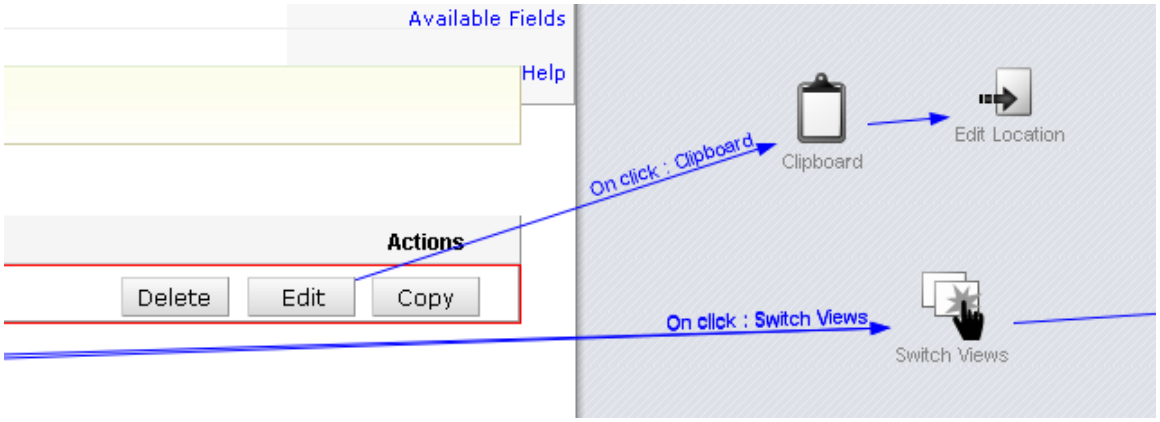
28. In the Set Destination dialog box, choose the Edit Location page, then click Done.

29. Drag-and-drop the Edit button to the clipboard created in step 25.

30. From the context menu that is displayed, choose Create Navigation.

31. Drag-and-drop from the clipboard to the Edit Location link widget.

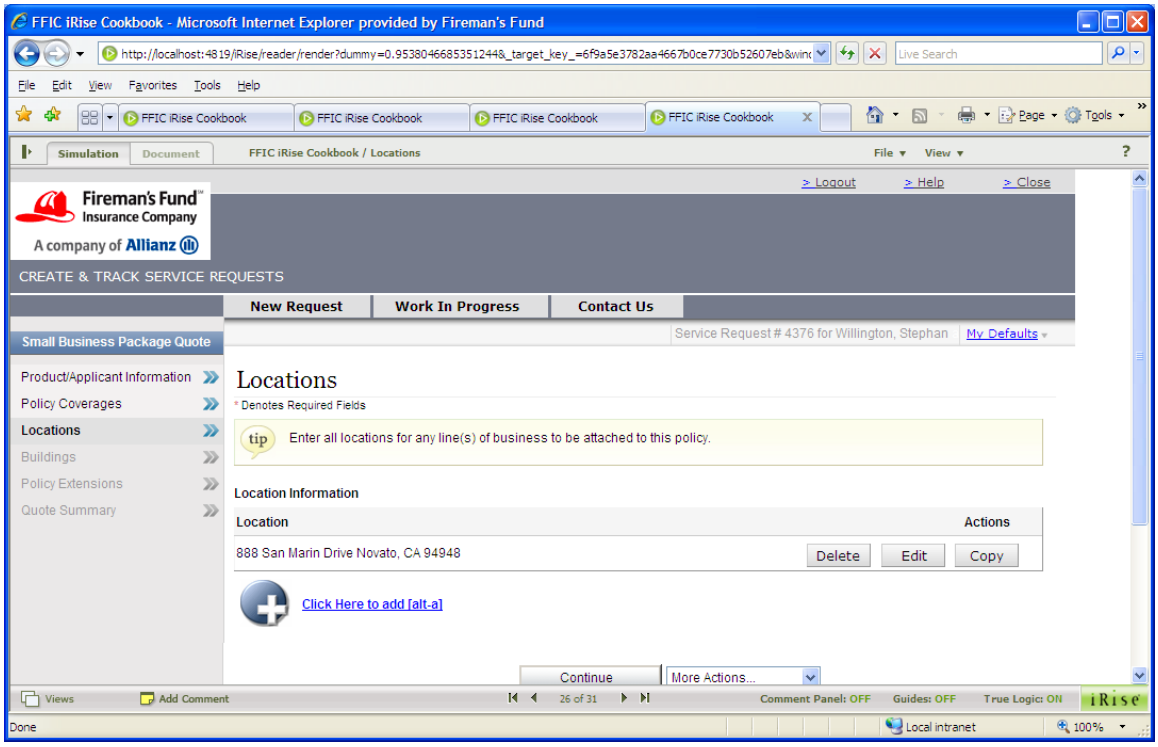
    At the completion of this operation, your page should look like the following:



32. With the Locations 2 page still selected from the project tree, click the Launch button to verify your work to this point.

33. In the browser window, click the Edit button.

34. Edit the Address Line 1 field to “888 San Marin Drive”, then click Save.



35. From the project tree, click the Locations datasheet to verify that it was updated with your edit.

	Mailing Address 1	City	State Name	Zip	Territory Code
1	888 San Marin Drive	Novato	California	94948	021
2	New Row				

Performing Calculations and Conversions

Create a Field That Displays the Current Date

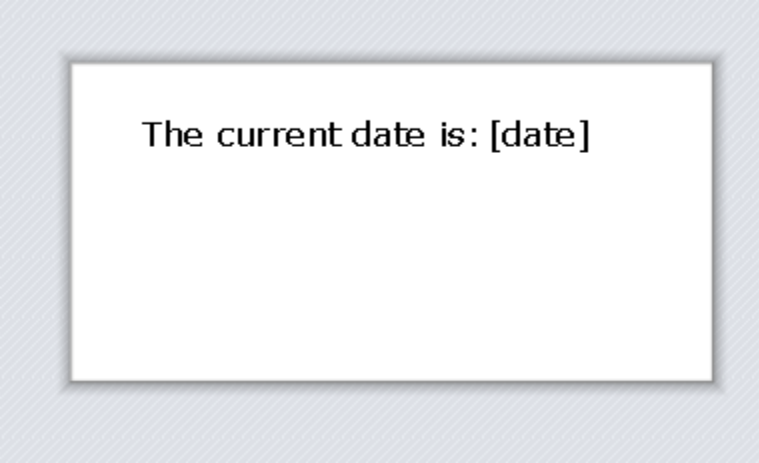
Oftentimes, you'll need to simulate an application that defaults a date field to the current date (for example, when creating a new online quote, the Web Solutions application defaults the Effective Date field to the current date. iRise provides some rudimentary date calculation widgets, including a number that can be used for calculating the current date. In the following recipe, we'll be creating some logic widgets that return and apply formatting to the current date, and then display the results of this logic in an associated page widget. We'll be using the pages On Load event to trigger calculation of the date.

To report the current date in a text widget:

1. Create a new page in your current project.

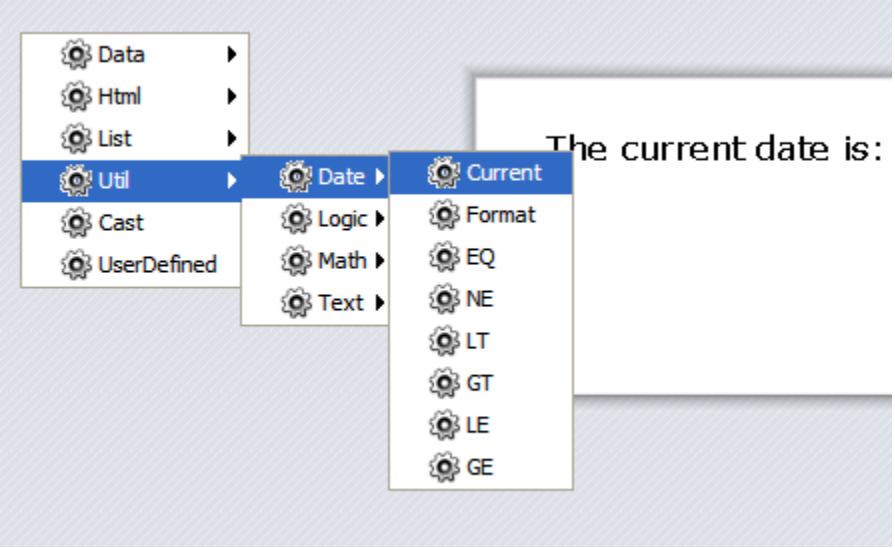
2. Place two text widgets with the following text values onto your page:

a. Text widget 1: The current date is:
b. Text widget 2: [date]

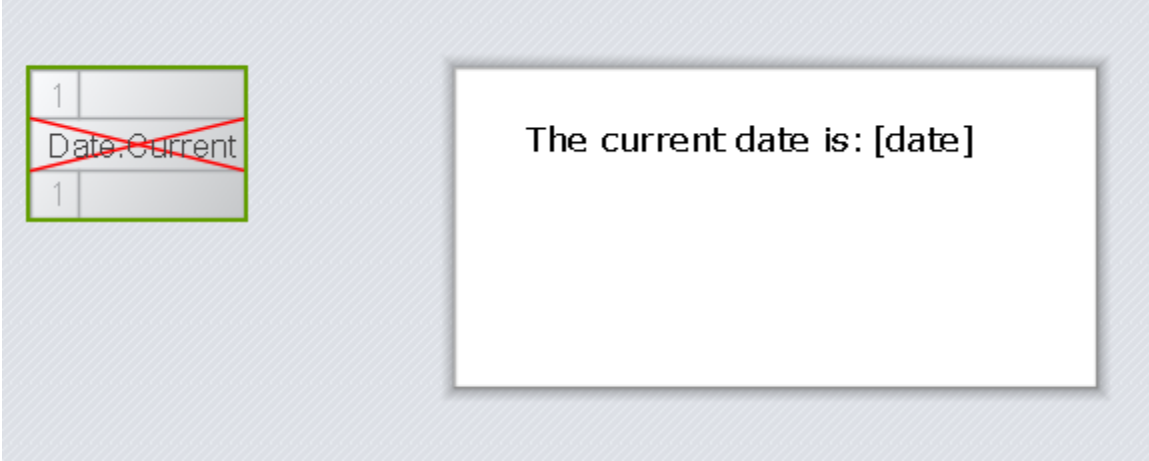


3. Click the Operator toolbar button, then click to the left of the page to place the operator on the drawing canvas.

4. In the context menu that is exposed, select the Until > Date > Current option.

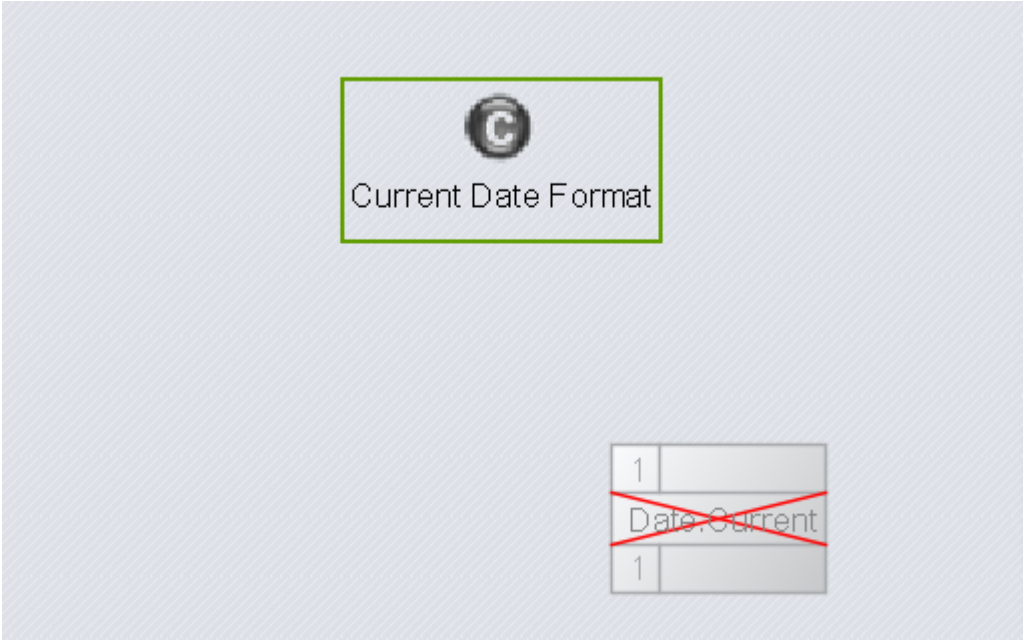


At the completion of this operation, your page canvas should look as depicted in the following screenshot.

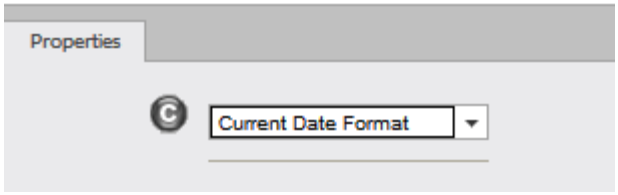


5. Click the Constant toolbar button, then click to the left of the page to place the operator on the drawing canvas slightly above and to the left of the Date.Current widget you placed in the preceding steps.

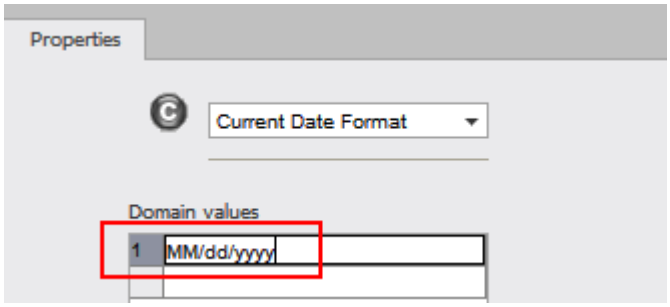
6. Select the Constant you just placed on the drawing canvas.



7. Optional (but recommended) step: provide a meaningful name for your Constant using the Properties Palette. For this example, I'm naming my constant Current Date Format.

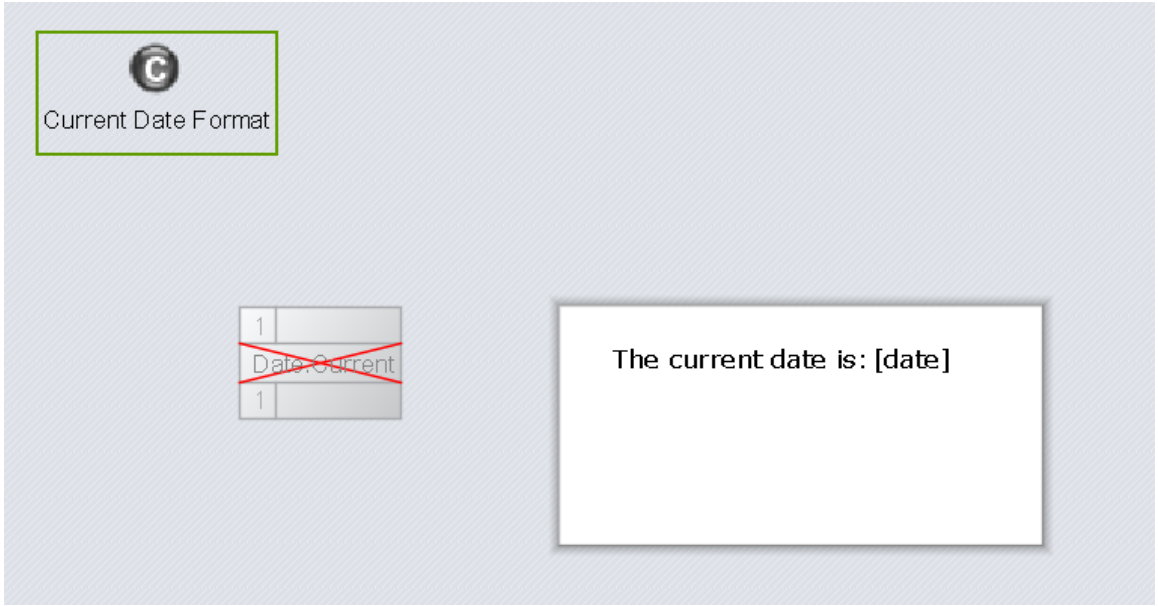


8. In the Domain Values section of the Properties Palette, specify the following string to set the date format: MM/dd/yyyy.

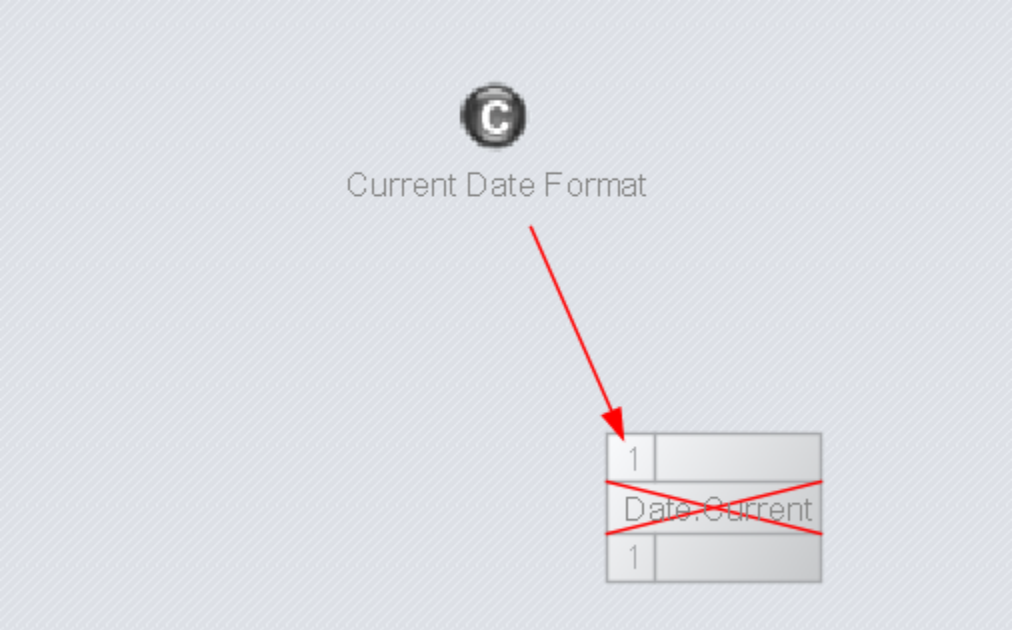


The preceding two steps define a new constant named Current Date Format, and specify that the date will be presented as a two digit month (MM) followed by a backward slash separator; a two digit day (dd) followed by a backward slash separator; and a four digit year (yyyy). If desired, you can substitute another separator (for example, a "-" character) or set your date to display more or fewer digits, or even right out the name of the month. Since iRise internally uses the backward slash for the included Date widget, you usually will achieve more predictable results with less effort if you stick with this format.

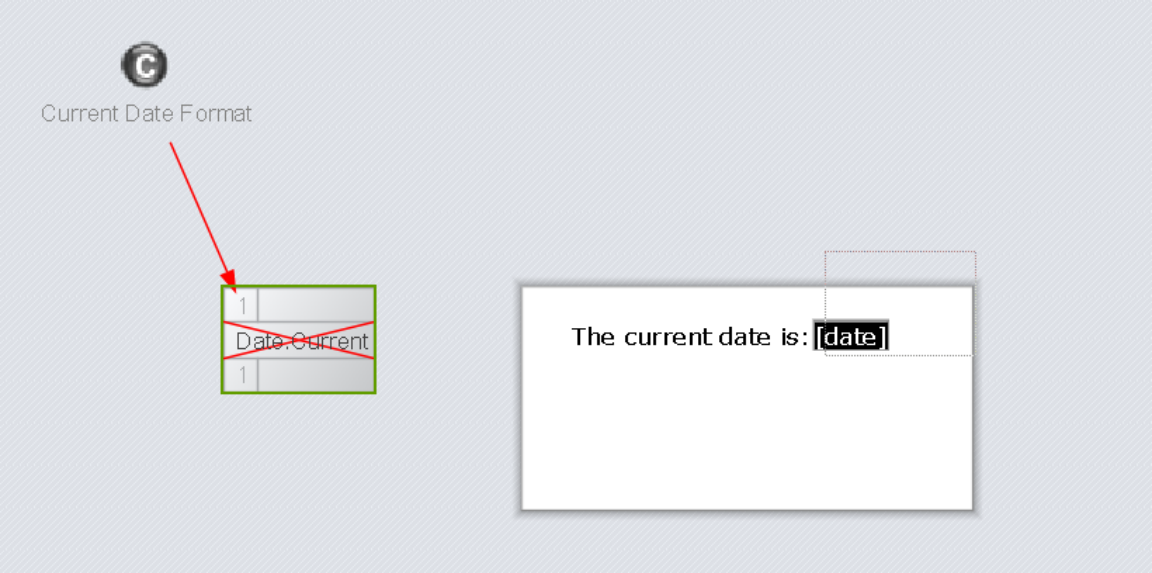
At the completion of the preceding two steps, your drawing canvas should look similar to the following:



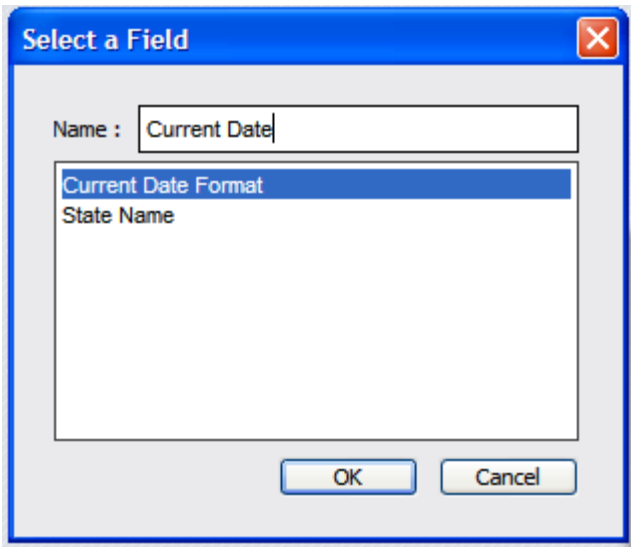
9. Select the Current Date Format constant, then drag it to the current date widget to establish a data flow relationship.



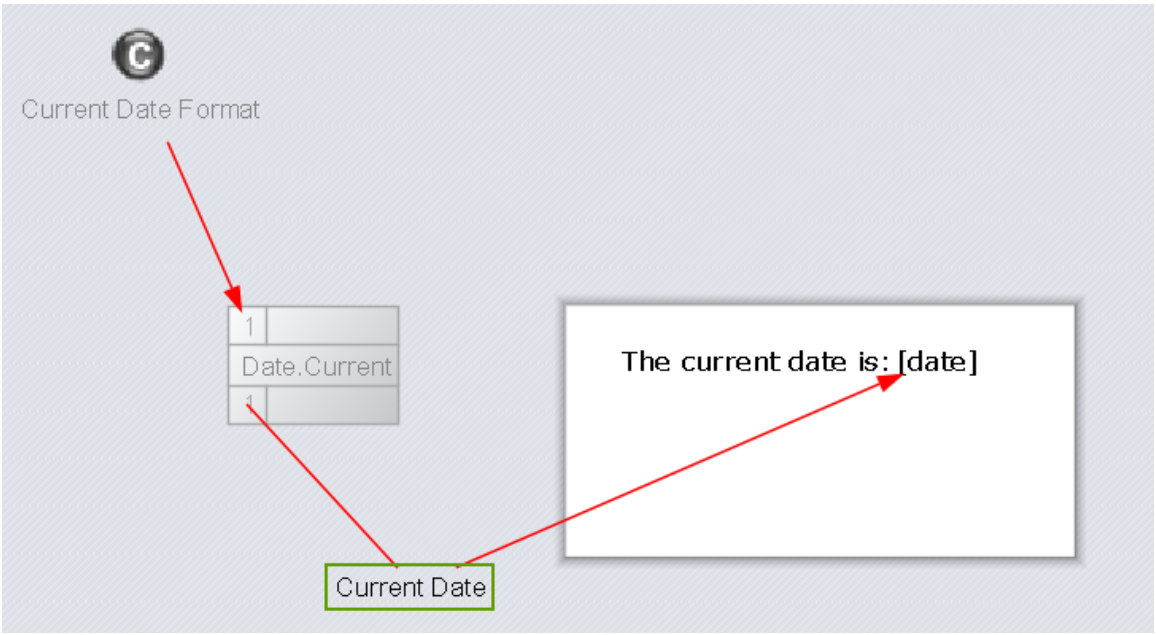
10. Select the current date widget, then drag and drop it on the [date] text object on the page.



11. In the Select a Field dialog box, define a new variable by typing Current Date in the Name field, then click the OK button.

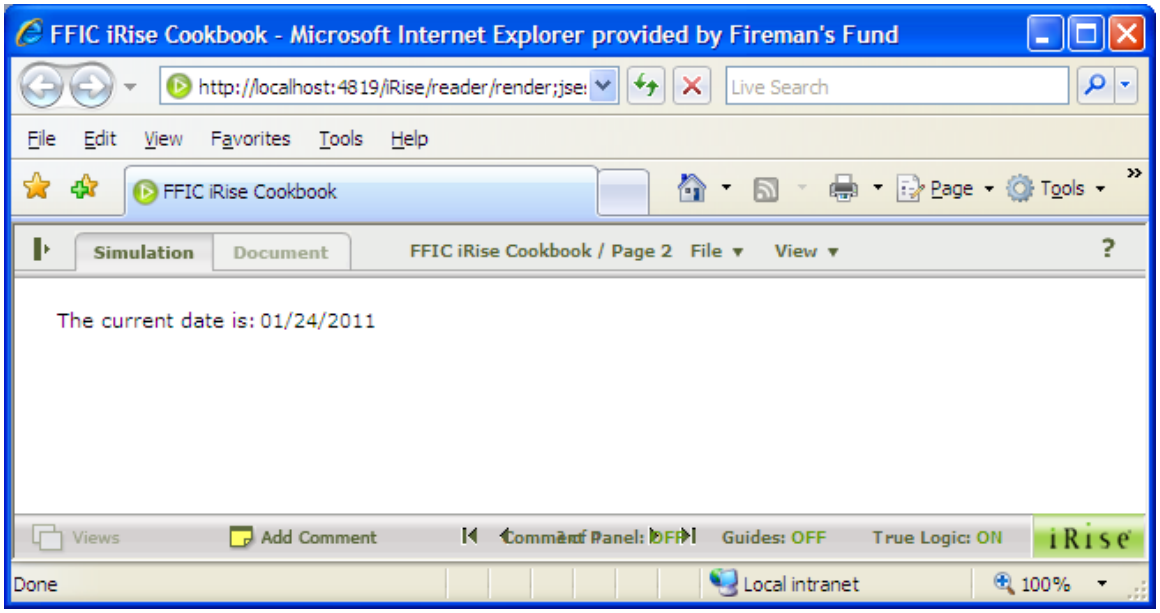


Your application environment at this point should look similar to the following:



12. Click the Launch toolbar button to load your completed web page into your browser.

At this point, your web page should render, displaying the current date in the specified format.



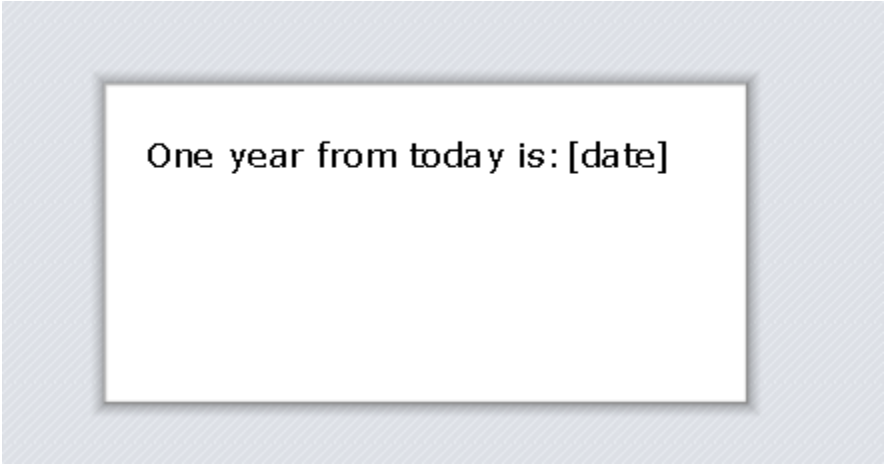
Create a Field That Adds One to the Year of the Current Date

While the following recipe has many more steps then the previous Current Date recipe and may initially look quite complicated, you can breathe easy and rest assured that it is only slightly more complicated in actual practice. We'll once again be working with The Current Date widget, but this time rather than constants, we'll be working with the User Defined widget. To perform our recipe magic, we'll need to create two Current Date widgets: one that will hold the month and day portion, and one holding the current year. We'll be using a new Expression widget to add one to the current year, and then be piecing our date back together using the Text Add widget.

1. Create a new page in your current project.

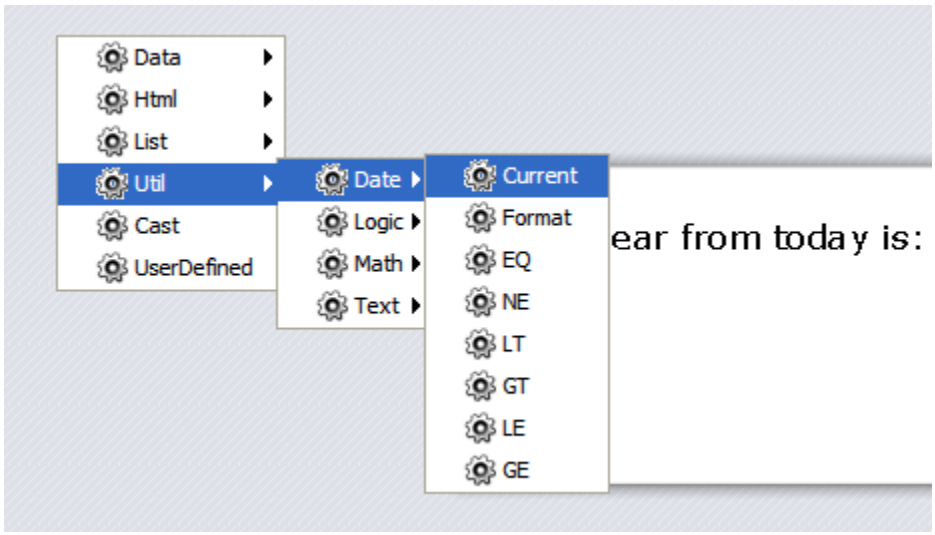
2. Place two text widgets with the following text values onto your page:

    a. Text widget 1: One year from today is:
    b. Text widget 2: [date]

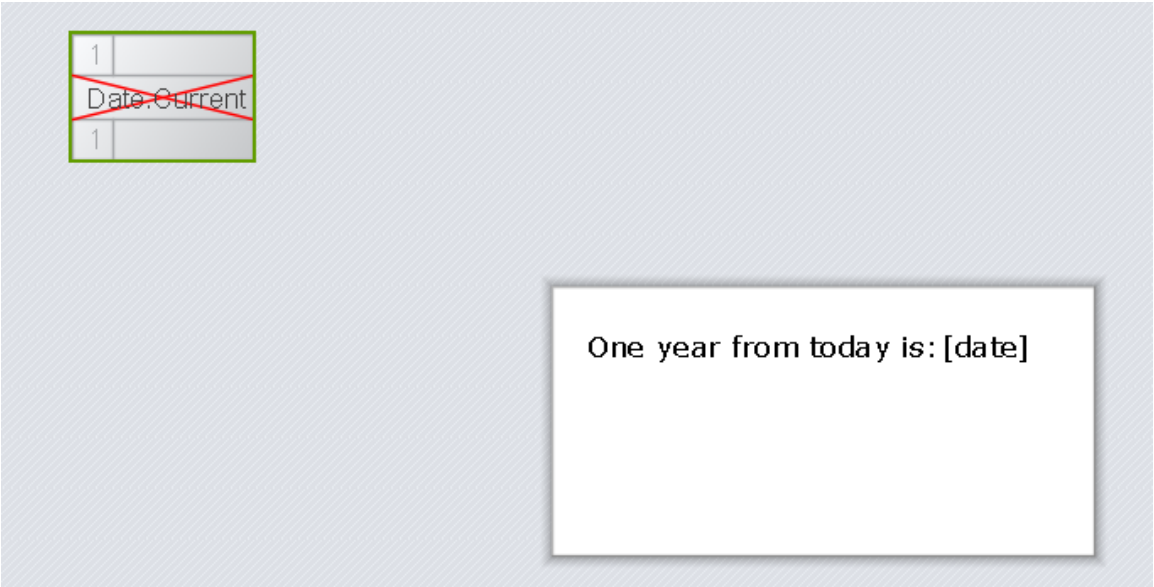


3. Click the Operator toolbar button, then click to the left of the page to place the operator on the drawing canvas.

4. In the context menu that is exposed, select the Until > Date > Current option.



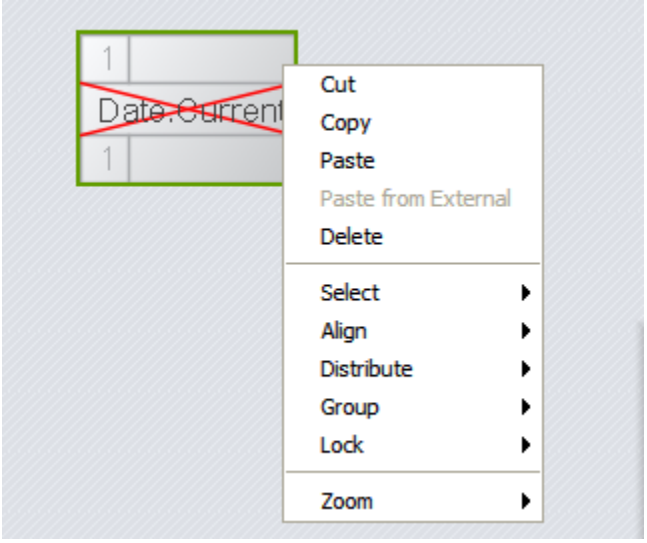
At the completion of this operation, your page canvas should look as depicted in the following screenshot.



Next, we'll be defining the format for the date. Since we are adding 1 to the current year, we need to split the current date into two pieces: one component

for the month and day, and another for the year. We'll be using some new widgets to complete this operation. But first, let's place one additional Current Date widget which will hold just the year portion of our date.

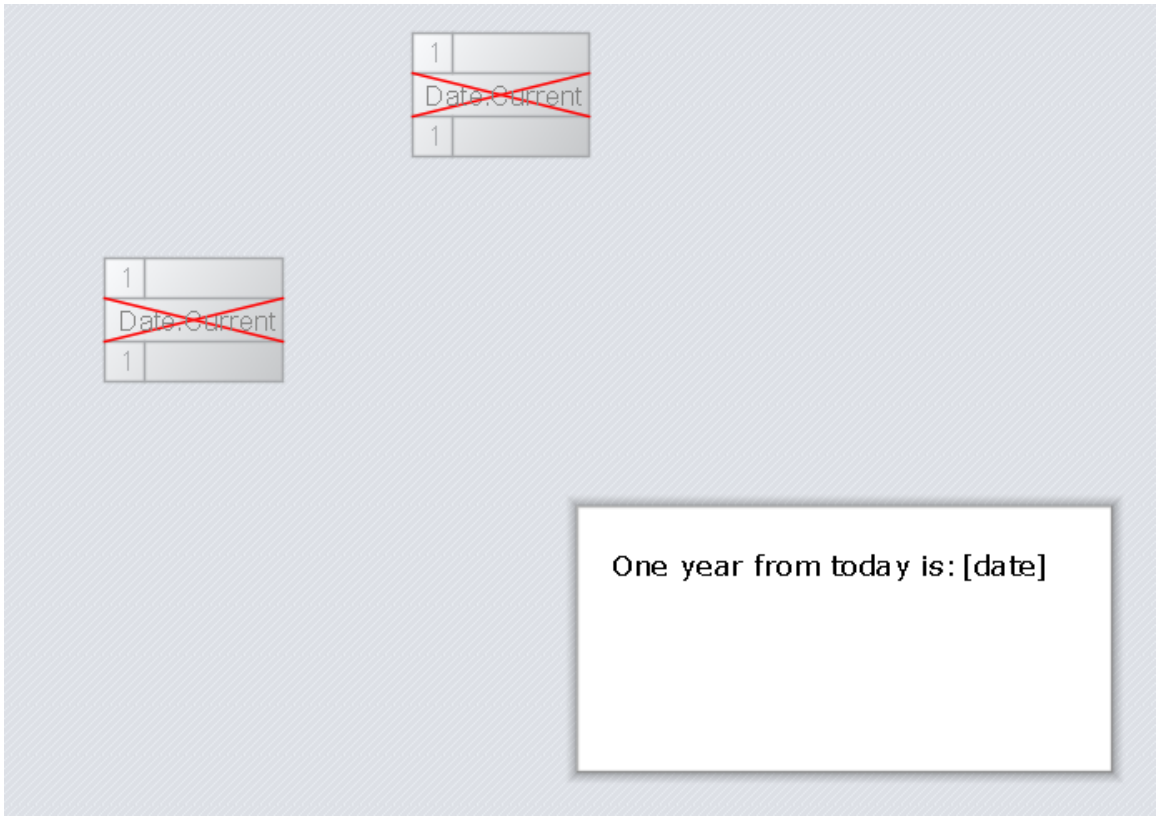
5. Right-click the Current Date widget you placed on the canvas, then choose the Copy context menu option.



6. Right-click on the canvas slightly above and to the right of the Current Date widget you previously placed, then choose Paste.

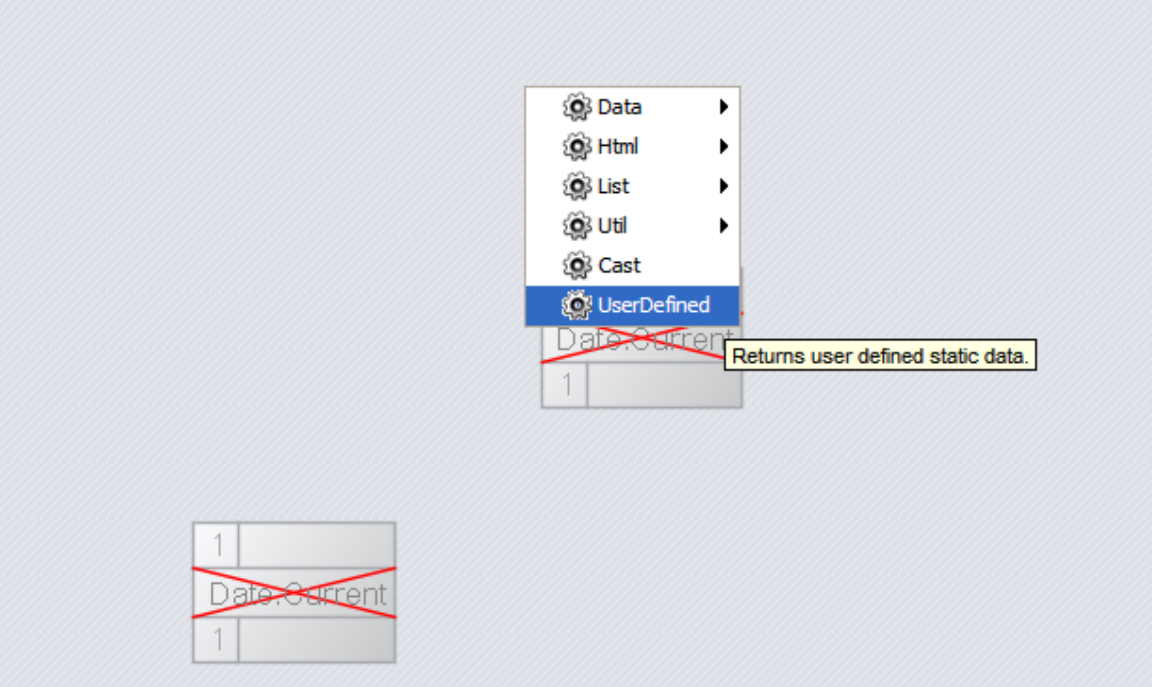
7. Right-click on the canvas slightly above and to the right of the Current Date widget you previously placed, then choose Paste.

   At the completion of this operation, your page canvas should look as depicted in the following screenshot.



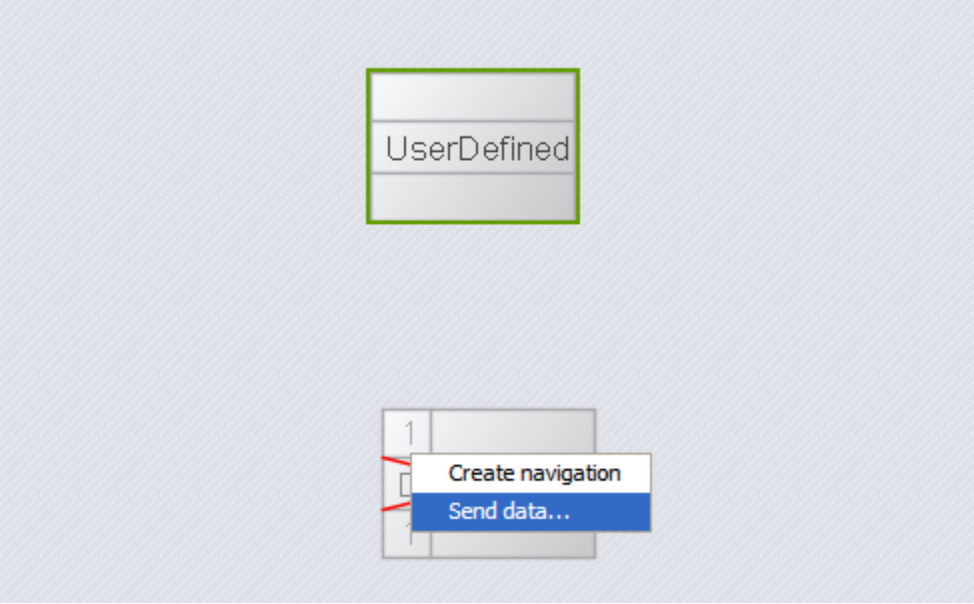
8. Click the Operator toolbar button, then click a region on the drawing canvas slightly above the Current Date widget you just placed.

9. Select the User Defined context menu option.

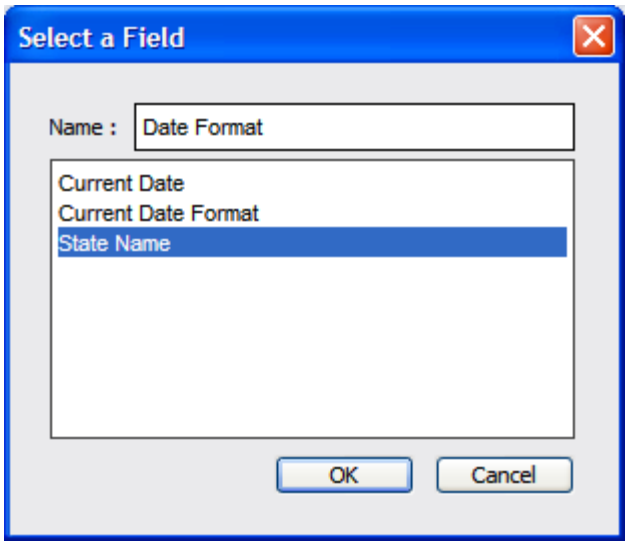


10. Select the User Defined widget, then drag and drop it on top of the new Current Date widget.

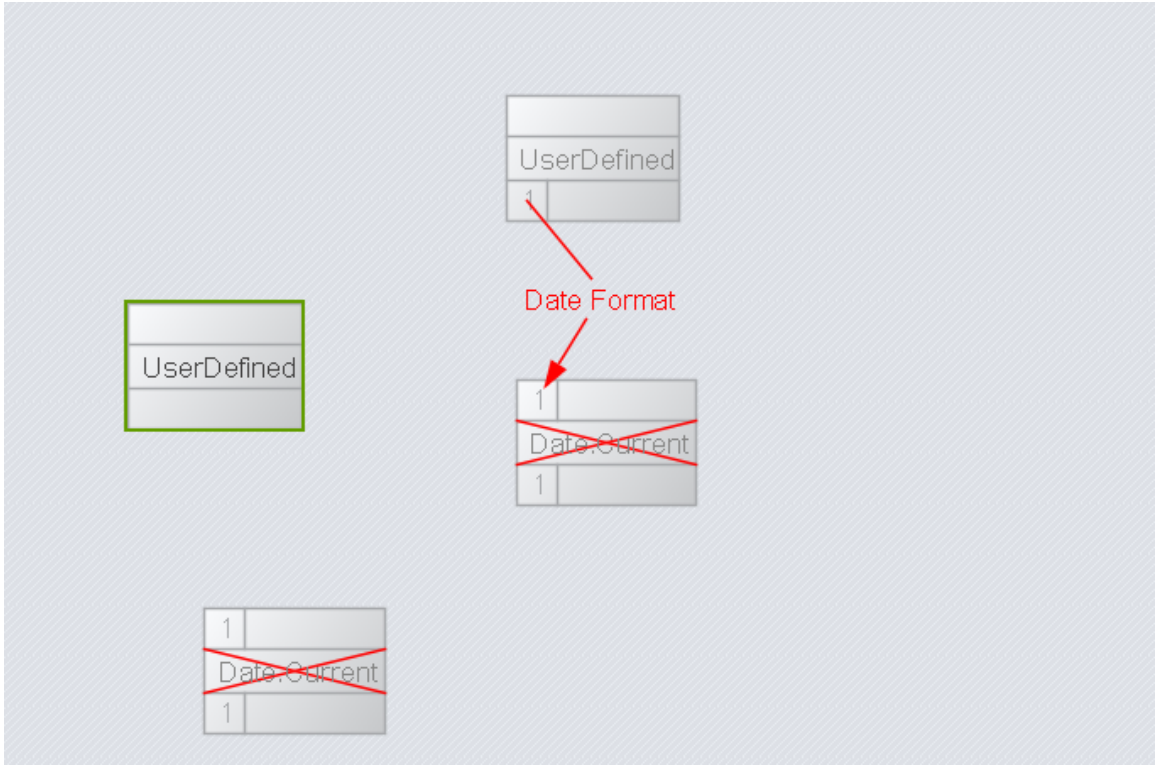
11. Select the Send Data context menu option.



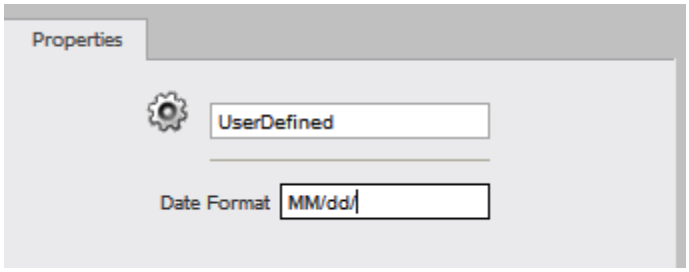
12. In the Select a Field dialog box, enter Date Format in the name field, then click OK.



13. Copy and paste the UserDefined widget to a location slightly above the other Current Date widget.

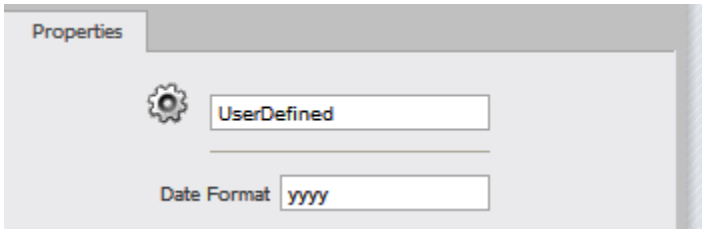


14. With the User Defined widget you just placed selected, enter MM/dd/in the Date Format field on the Properties Palette.



15. Select the other User Defined widget and enter yyyy in the Date Format field on the Properties Palette.

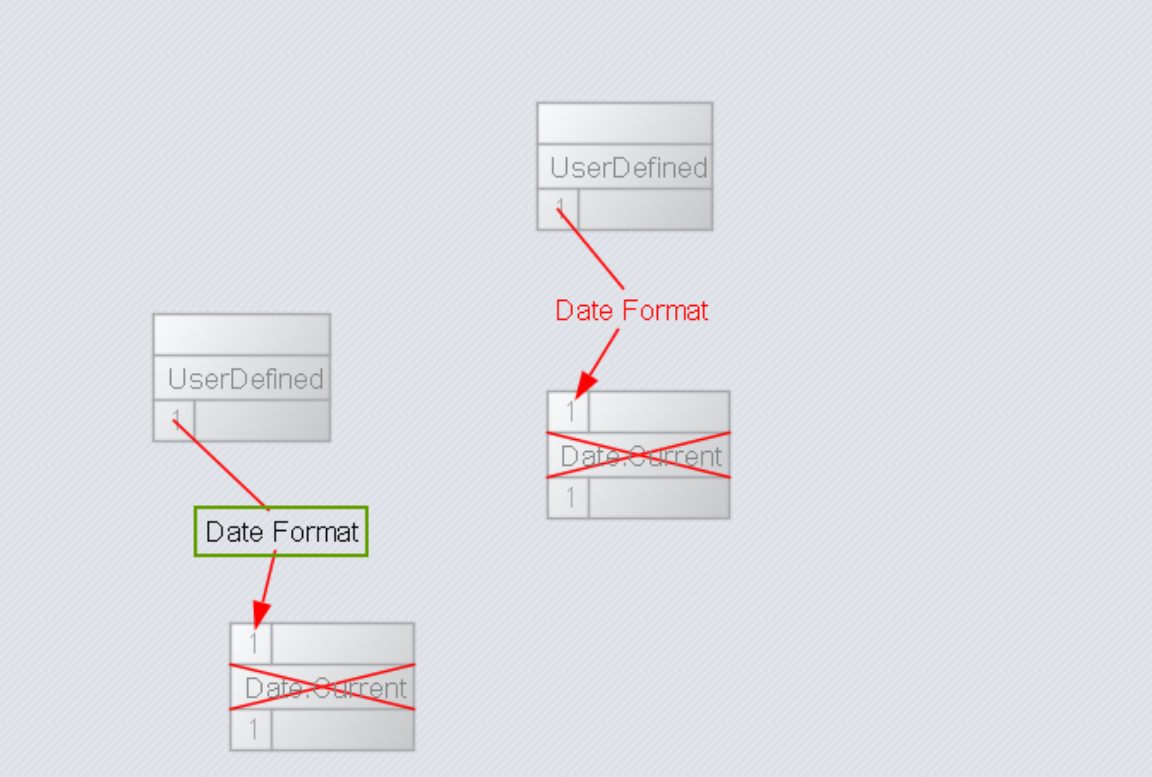
16. Select the other User Defined widget, then drag and drop it on top of the new Current Date widget.



17. Select the Send Data context menu option.

18. In the Select a Field dialog box, select Date Format from the list of available fields, then click OK.

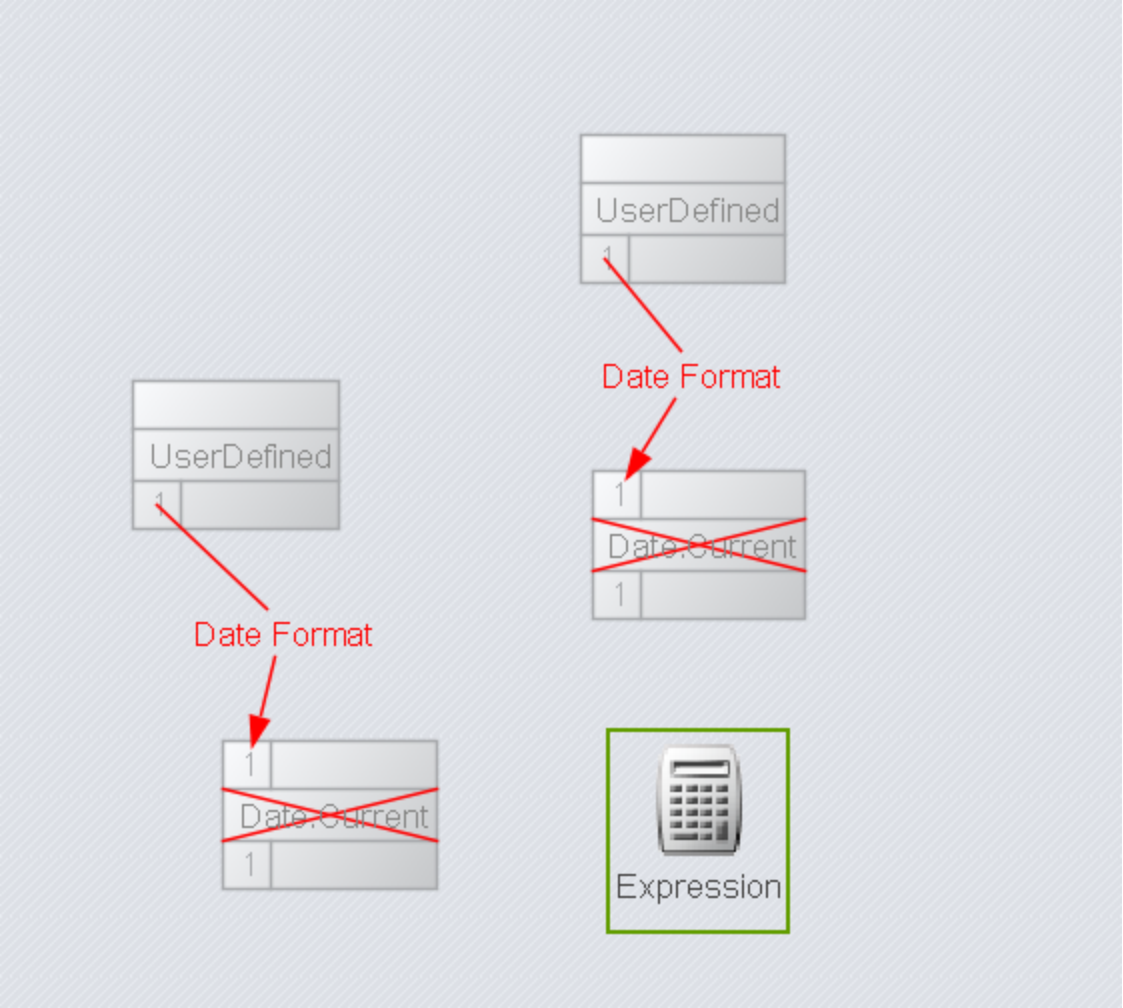
    At this point, your drawing canvas should look like the following.



Next, we'll place an Expression widget to add 1 to the current year.

19. Select the Expression button from the toolbar.

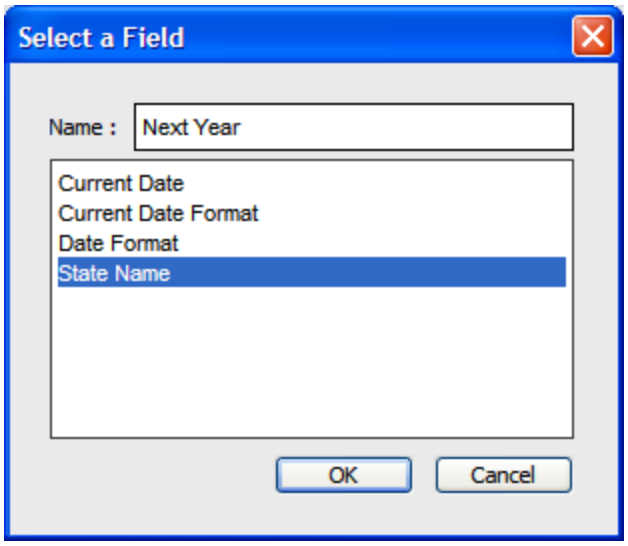
20. Click on the drawing canvas slightly below the upper Current Date widget.



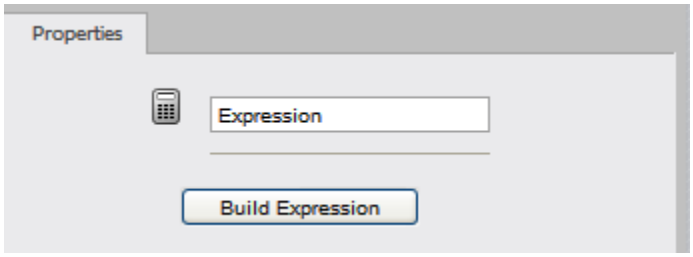
21. Select the upper of the two Current Date widgets, then drag and drop it on the Expression widget.

22. Select the Send Data context menu option.

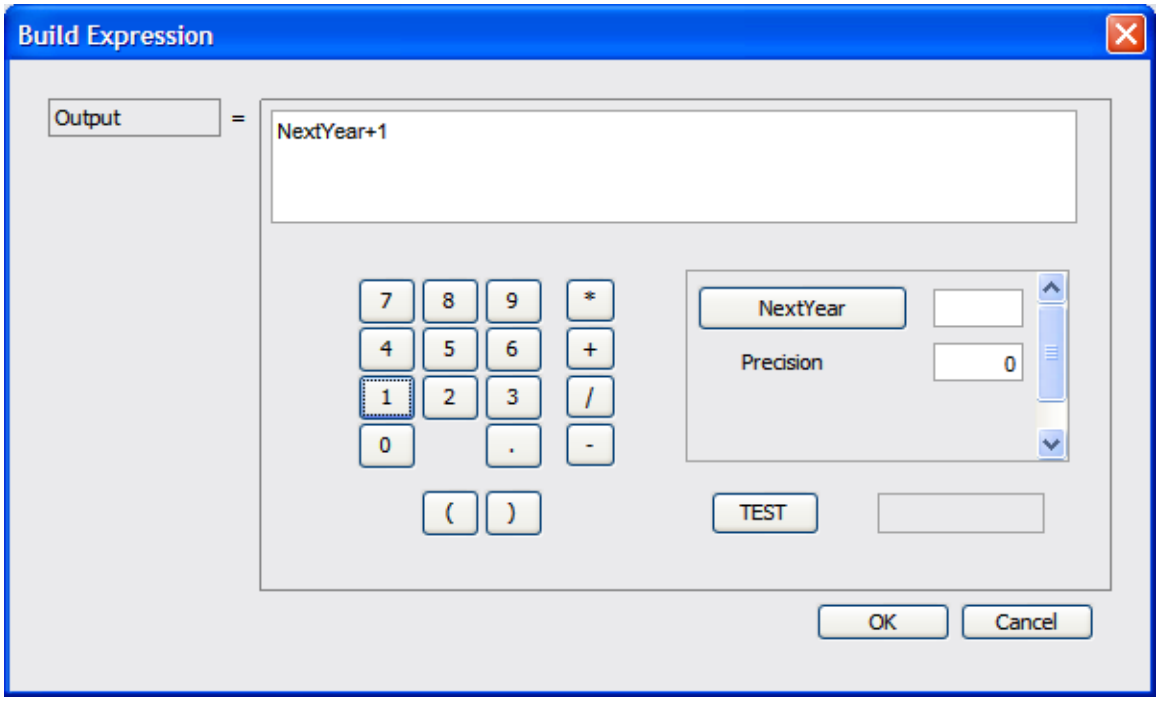
23. In the Select a Field dialog box, enter "Year + 1" in the Name field, then click OK.



24. With the Expression widget you just placed selected, click the Build Expression button from the Properties Palette.



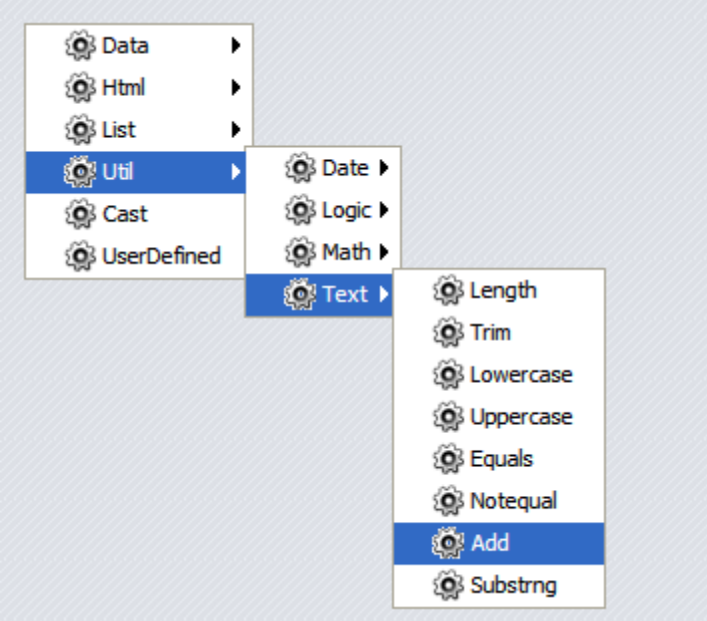
25. In the Build Expression dialog box, complete the following steps:
    a. Click the NextYear button.
    b. Click the + button.
    c. Click the 1 button.
    d. Enter 0 in the Precision field, then click OK.



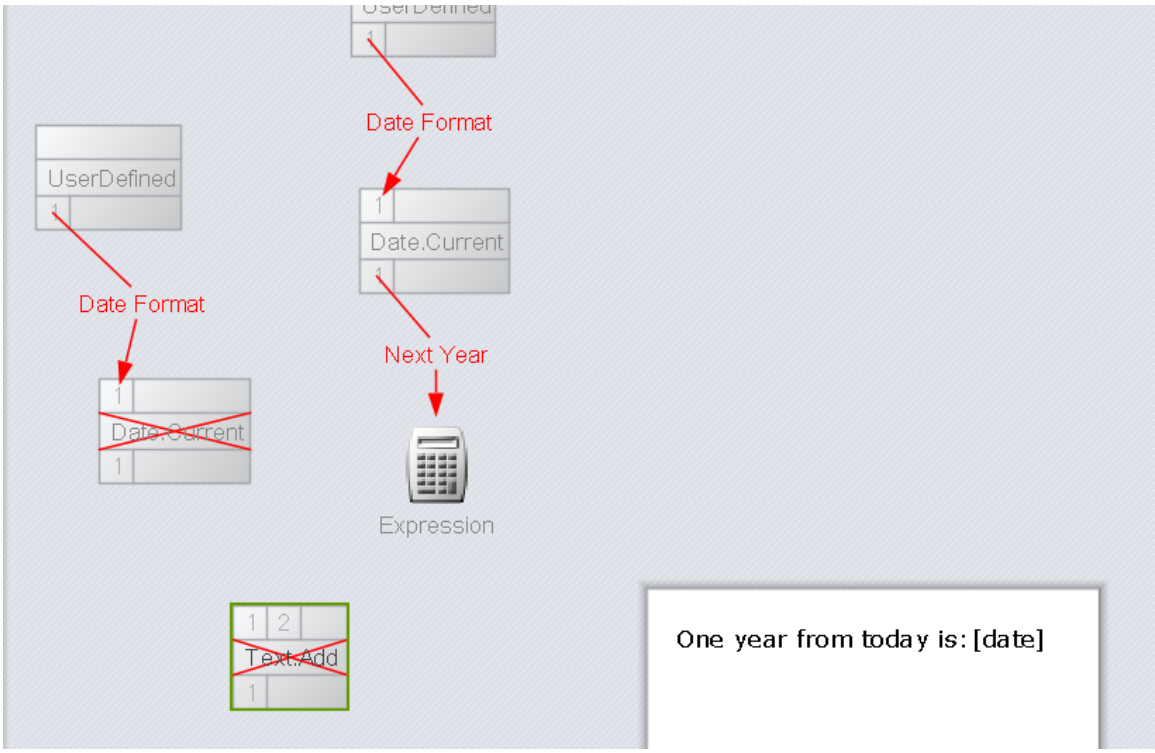
Next we'll be using a new widget to add back together the two portions of the date that you'd previously split apart.

26. Click the Operator toolbar button, then click a region on the drawing canvas slightly below the Expression widget.

27. From the context menu, select Util > Text > Add.



At this point your drawing should look something like this:



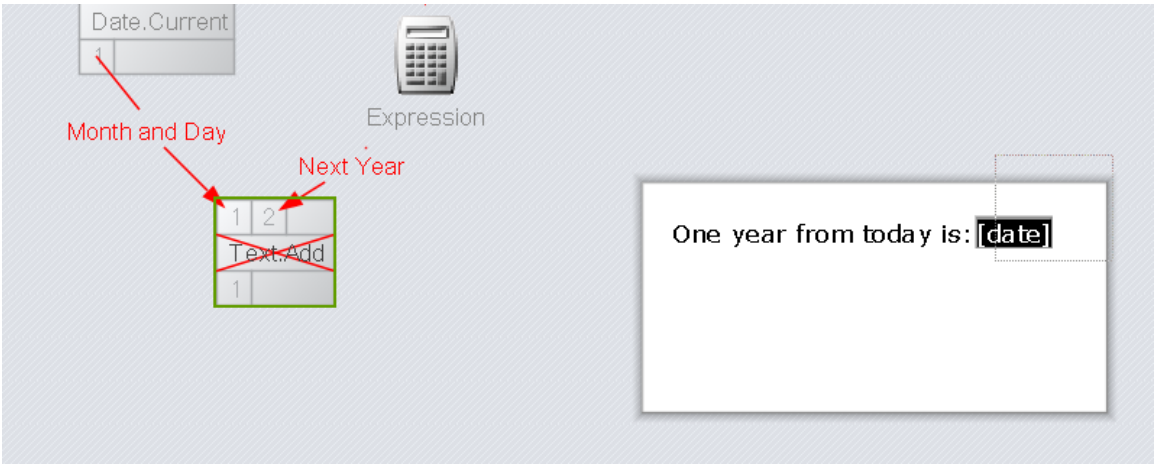
28. Drag and drop the lower Current Date widget to the Text Add widget, then choose Send Data.

29. In the Select a Field dialog box, enter Month and Day in the Name field, then click OK.

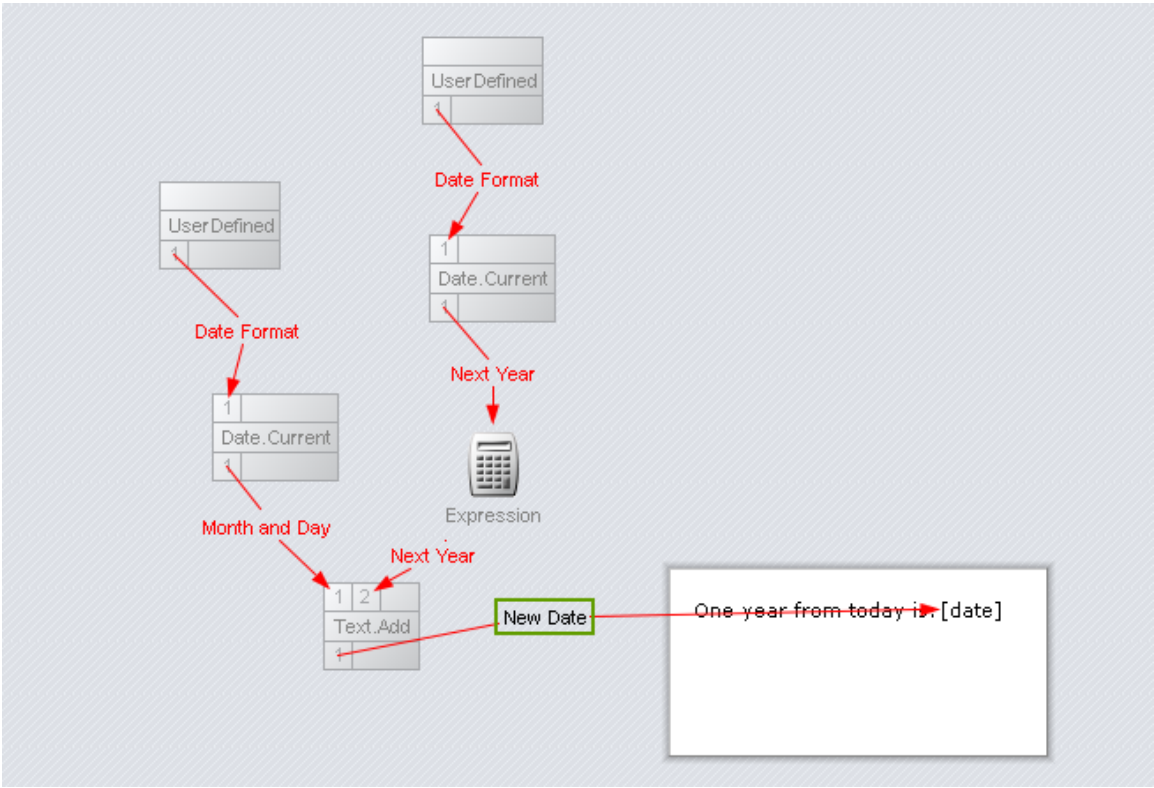
30. Drag and drop the Expression widget onto the Text Add widget, then choose Send Data.

31. In the Select a Field dialog box, choose Next Year from the list, then click OK.

32. Drag and drop the Text Add widget onto the [date] text widget on the page.

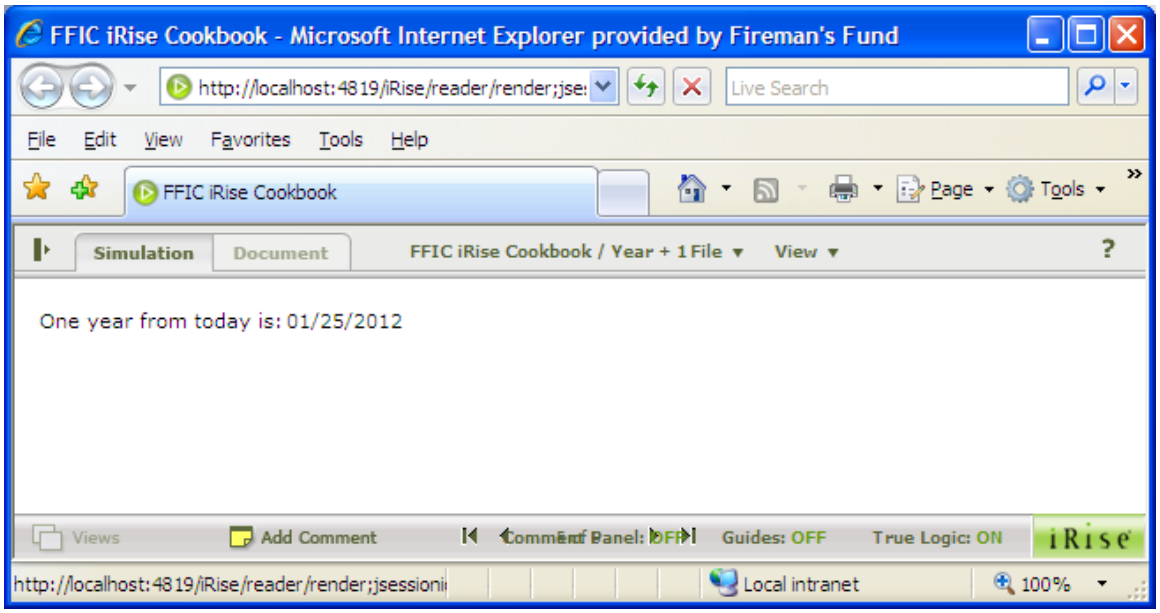


33. Enter New Date in the Name field of the Select a Field dialog box, then click OK.



34. Click the Launch toolbar button to load your completed web page into your browser.

At this point, your web page should render, displaying the current date with one year added to it.



Miscellaneous Routines

Using Show/Hide Widgets as an Alternative to Views

Views are a powerful mechanism within iRise for toggling between a variety of different visual representations of a set of controls, but they are not without drawbacks or limitations. They tend to work best when toggling state information for a set of identical controls and are most difficult and problematic to work with when switching from an empty view to a view that dynamically displays controls that were previously hidden. When working with empty views, it is quite easy to forget to "reset" the hidden view to a height of 0, resulting in pages within your scenario with large amounts of empty space.

Fortunately, however, iRise often provides a multitude of different mechanisms for accomplishing a given task, and the Show/Hide Widgets capabilities work very well for a variety of dynamic view interactions.

The example described here utilizes named table rows, but you can also experiment with this interaction for different types of widgets.

To use show/hide widgets as a replacement for alternative views

1. Create a simple page layout using a Table widget as the basic building block.

2. Identify a row within this table that you would like to hide when the page is initially loaded:
    a. Select the row from within iRise.
    b. From the Properties inspector, give a meaningful name to this row: you will reference this name in a subsequent step to make the row visible.
    c. From the Properties inspector, check the Hide in Browser check box.
3. Place a Check Box widget

Preserving State of Radio Button Settings

Oftentimes within a simulation, users will submit a form from a given page and return to that same page to perform further edits. For example, both the Search by Payor and Search by Policy work-flows require the user to enter a search string, and any matching results are returned to the calling search page. It is desirable under this circumstance to maintain the state of any radio button settings, such that they can be restored when the user is returned back to the page.

To preserve the state of radio button settings on a page

1. Create a clipboard, and pass data to it from one of the options from within the radio button group[1].
2. Create a Submit Form action and associate it with a form widget (e.g., a button).
3. Create a Navigation link from the Submit Form action that returns you to the current page.
4. Create a second clipboard which passes the data variable defined in step 1 back to the radio button group.

[1] While somewhat counter-intuitive, you do not need to pass data from each of the radio button options within a radio button group – in fact, iRise blocks you from doing this.