



Migrating A FileMaker[®] Solution To Servoy[™]

A Hands-On Primer

Bob Cusick
bob@clickware.com

Migrating A FileMaker Solution To Servoy: A Hands-On Primer

The goal of this document is to help lead you through the process of rebuilding an existing FileMaker Pro solution in Servoy. We will go all the way through the process from exploring the FileMaker solution; to building all the forms and methods; to faithfully reproducing the solution in Servoy.

I'll break down the entire process into Chapters that you can take one chapter at a time; and there will also be versions of the solution that you can import into Servoy at the end of each chapter. So, grab your favorite beverage, and let's get started!

For this tutorial, I'm using FileMaker Pro 6.0 on a PC – although it will work exactly the same way if you're using a Mac. There's a lot of stuff covered in this tutorial, and if you follow the directions exactly - it will take you about 8 hours to go through it all.

I suggest you go through this tutorial twice. The first time, follow all the directions - and see how long it takes you to get through. The second time - you should be able to re-create the solution in about 2-3 hours. It's as they say - practice makes perfect!

Chapter 1 – Exploring Your FileMaker Solution

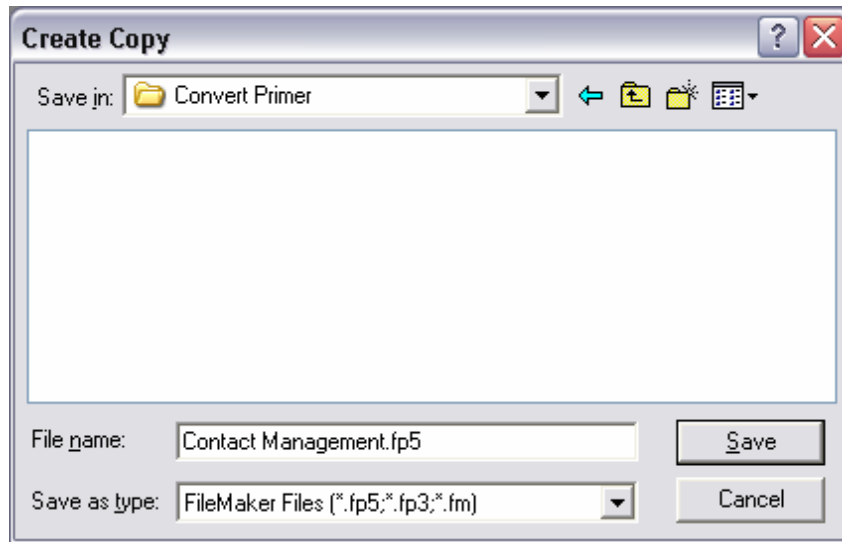
Estimated Time To Complete: 30-45 minutes

For the purposes of this tutorial, I'm using the "Contact Management" template solution that comes with FileMaker Pro 6.0. If you don't have FileMaker Pro 6.0 – I've included a copy of the file with the download package. If you do have FileMaker Pro 6.0 – then start by creating a new database:

Choose "New Database..." from the "File" menu and you should see a list of templates.

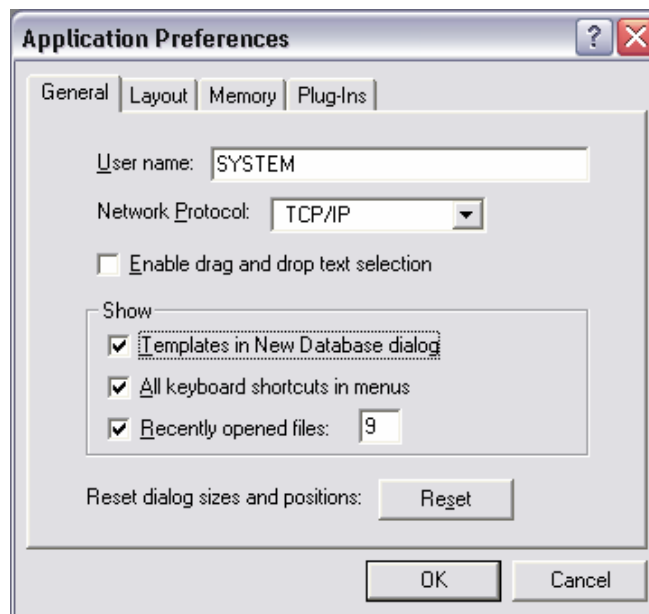


Choose the "Business" category from the pop-up list and click on "Contact Management". Then click "OK".



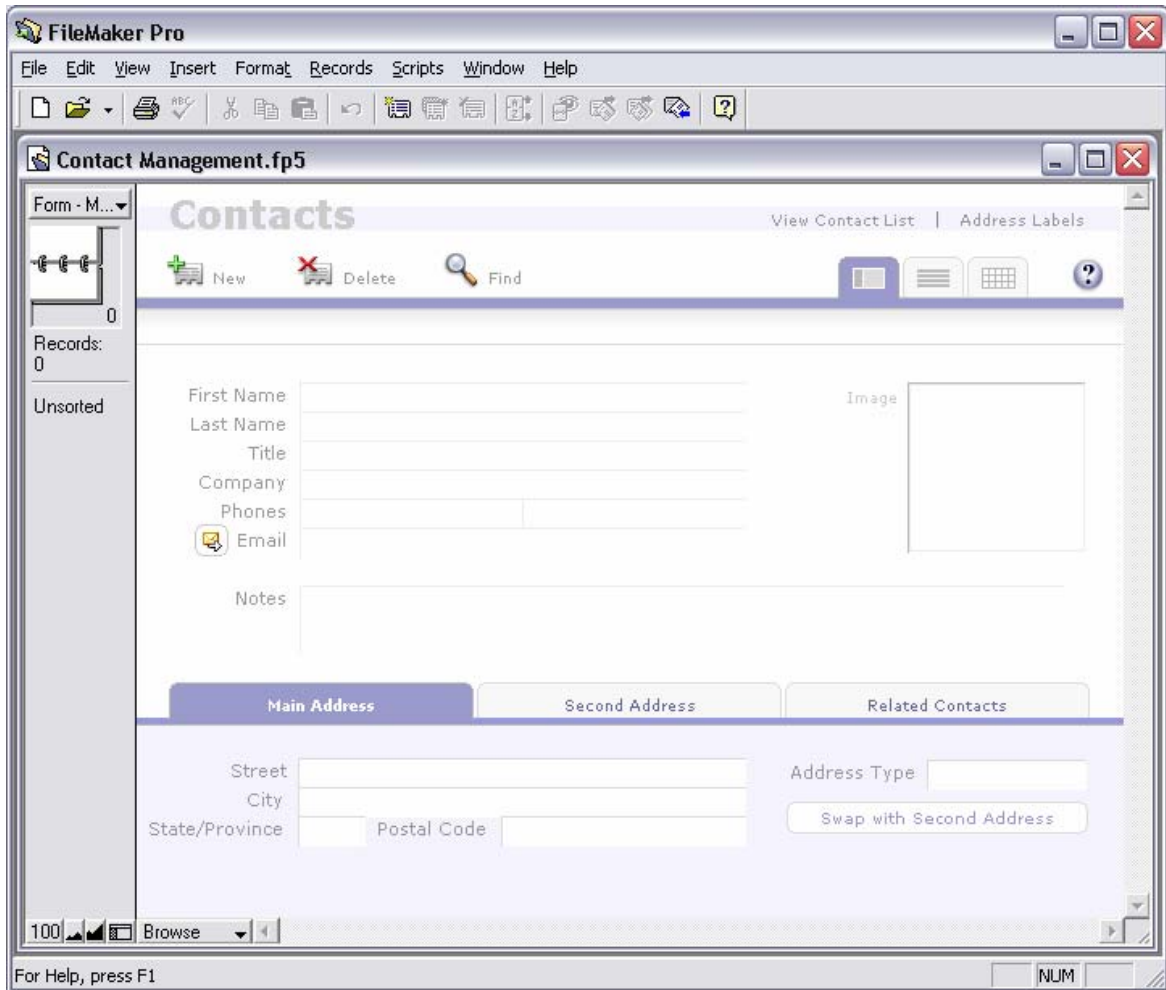
Name the file "Contact Management.fp5" and click "Save".

If you don't see the listing of templates when you choose "New Database..." – then you have the preference turned off. Here's how to make the template list appear: Choose "Preferences -> Application..." from the "Edit" menu and DO check "Templates in New Database dialog".



Click "OK" to close the dialog and choose "New Database..." from the "File" menu.

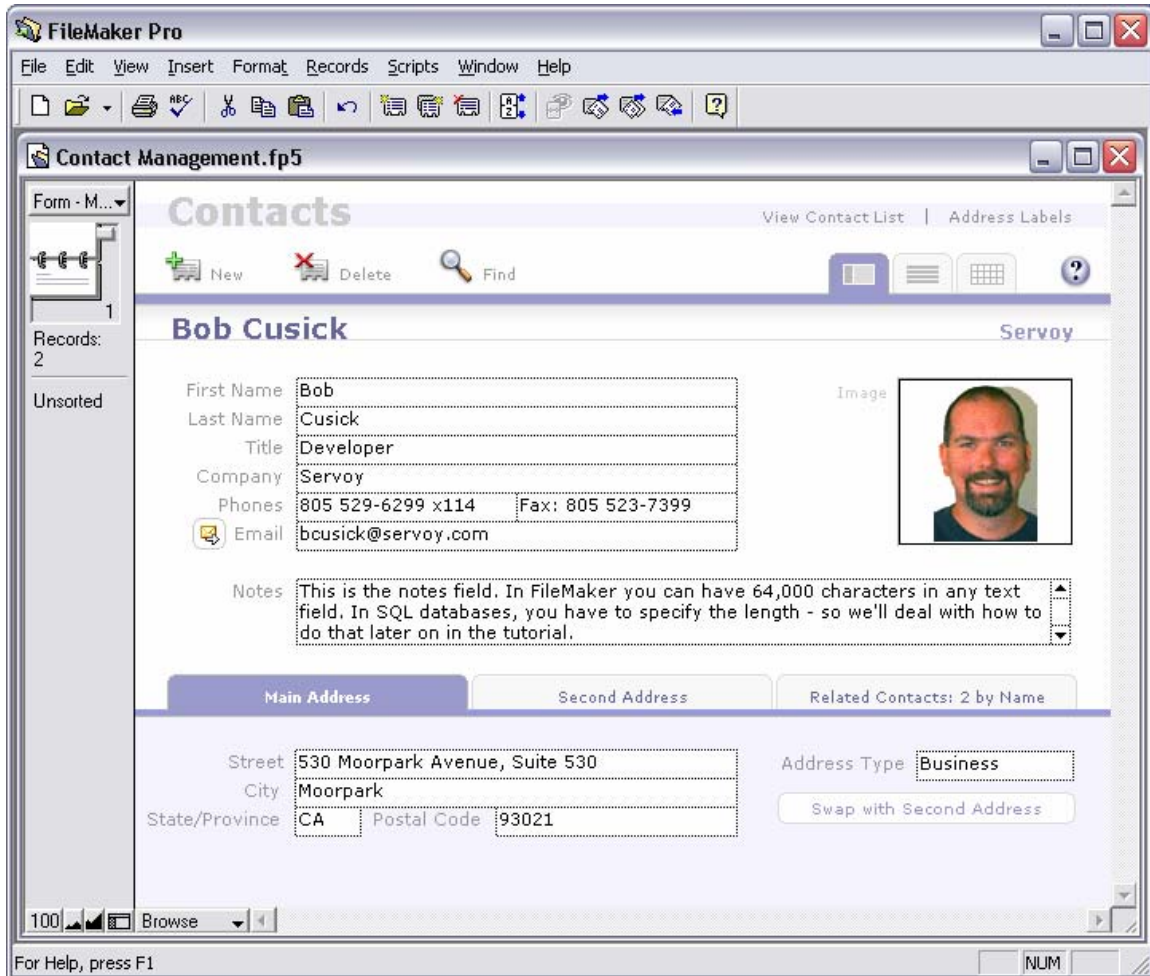
Once you save the template – it should open up right away. Here’s what you should see:



Now that we have the solution we want to convert – let’s examine what’s going on in the template so we can determine what we need to convert to Servoy, and so we can get a better understanding of the overall solution. If you’re the sole developer of a solution, then you probably have a good idea of what data is used where – but let’s go through the process together – it’s good practice.

Step 1 – Sample Data

The first thing that I do – is to create a sample record – filling out all the fields so that we can see where the data appears in the current solution.



Once you have a sample record or two, we'll begin by clicking around the solution to see what it does (we're assuming we didn't write this solution – which we didn't!).

Step 2 – Clicking Around

There are a couple of different ways you can figure out what's going on in a solution :

Go into Layout Mode and click the flip-book icon (or use the layout pop-up menu) to navigate between all the layouts.

Click through the various buttons, tabs, etc. within the solution to get an idea of how it works.

My personal preference is *to do both*. Going into Layout Mode will show you all the physical layouts in the file – and in this case – we will want to migrate all of the layouts. However, when you're duplicating this process in your Servoy solution – you may find all kinds of layouts that aren't needed; or that were created for some long-forgotten purpose.

HOT TIP: Don't bother migrating the stuff you don't need anymore.

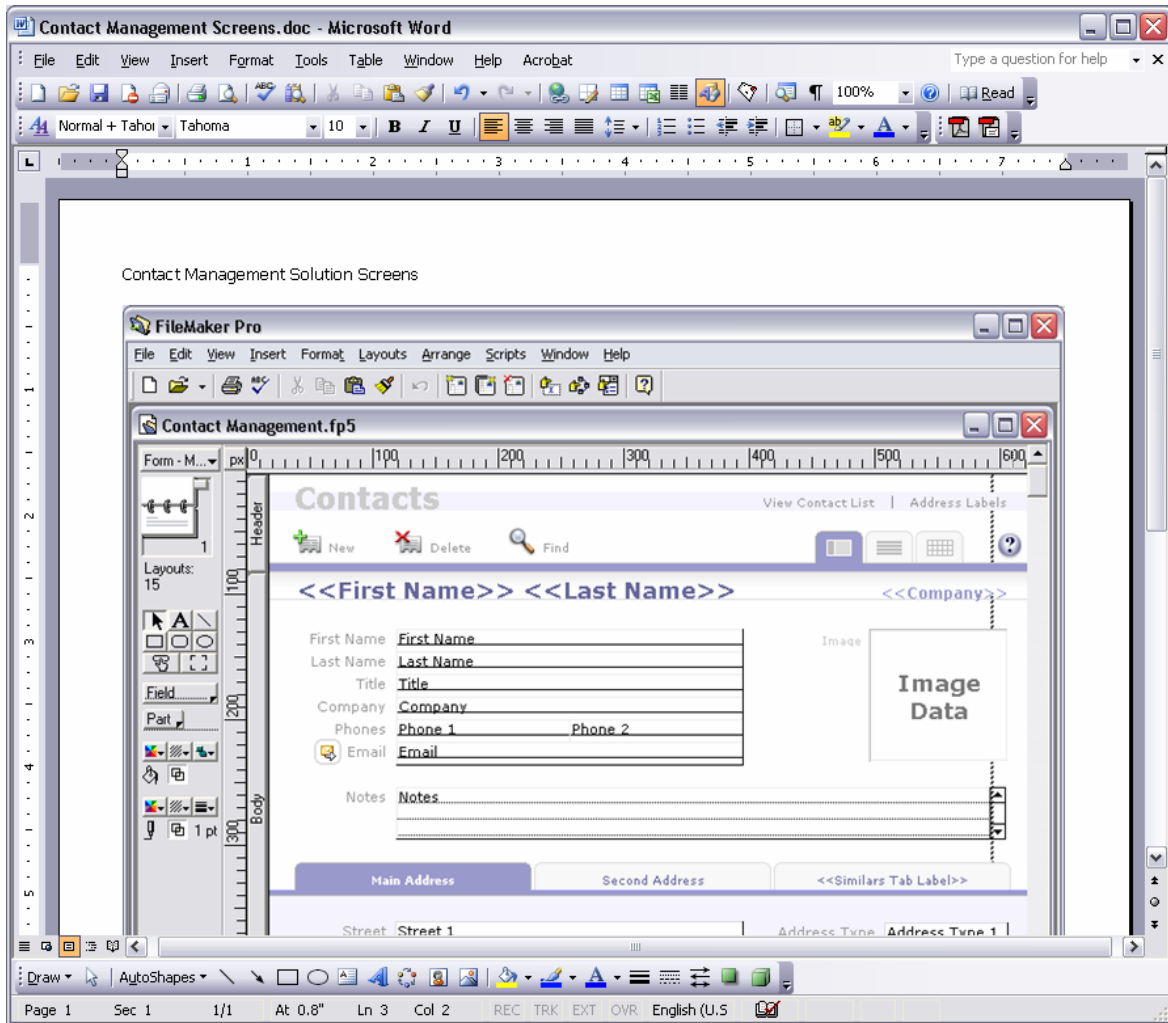
Once you've clicked through the solution to see what the buttons do, and have an idea of how the solution basically functions, it's time to get down to work.

Since we'll be creating a new SQL database for our solution – and one or more data tables to hold the data – we will focus on duplicating only the data we need to TYPE IN at this point. All the other fields that may exist in your solution – calculations, globals, summaries – don't matter. We'll deal with those types of data on the Servoy side. The MOST IMPORTANT fields are the ones that the user has to enter data in.

So, to get a better understanding of the layouts we're going to convert and the fields they contain, we're going to go into Layout Mode and do screen captures of each screen – this also gives us a "journal" of these layouts for later use.

Let's take a step-by-step look at how we're going to accomplish this:

1. Go into Layout Mode.
2. Make sure your window is big enough to see all the elements, and then capture the screen. You can use your favorite screen capture utility – or if you don't have a screen capture utility – then use ALT+Prnt Scrn on the PC or COMMAND+SHIFT+3 on your Mac.
3. Once you have captured the screen, open your favorite word processing program (I'm using Microsoft Word) – and paste the screen capture (PC) – or drag-and-drop the captured file (Mac) into the document.



From looking at the list of layouts in this file (from the layouts pop-up menu) – apply this process for the following layouts:

Form – Main Address
 Form – Second Address
 Form – Similarity
 List
 Table
 Information
 Letter
 Avery 5160
 List Report
 Letter: Address Confirmation

We won't be migrating the "Web" layouts – nor the layouts with the name of "-" (these layouts are just used as the dividing lines when you're looking at the layouts in the pop-up menu). In this tutorial, we'll focus on migrating the "Form" layouts; the List and the Table layout. We'll leave the letters and labels for the next series.

Once you've finished capturing the screens – save your document and print it out (if you have a color printer – DO use the color – it will be helpful later).

Now that we have the printed roadmap of the layouts we want to migrate – and the names of the fields on each layout – let's start planning our data structure for the new table(s) we'll be creating.

Chapter 2 – (Re) Designing The Data Model

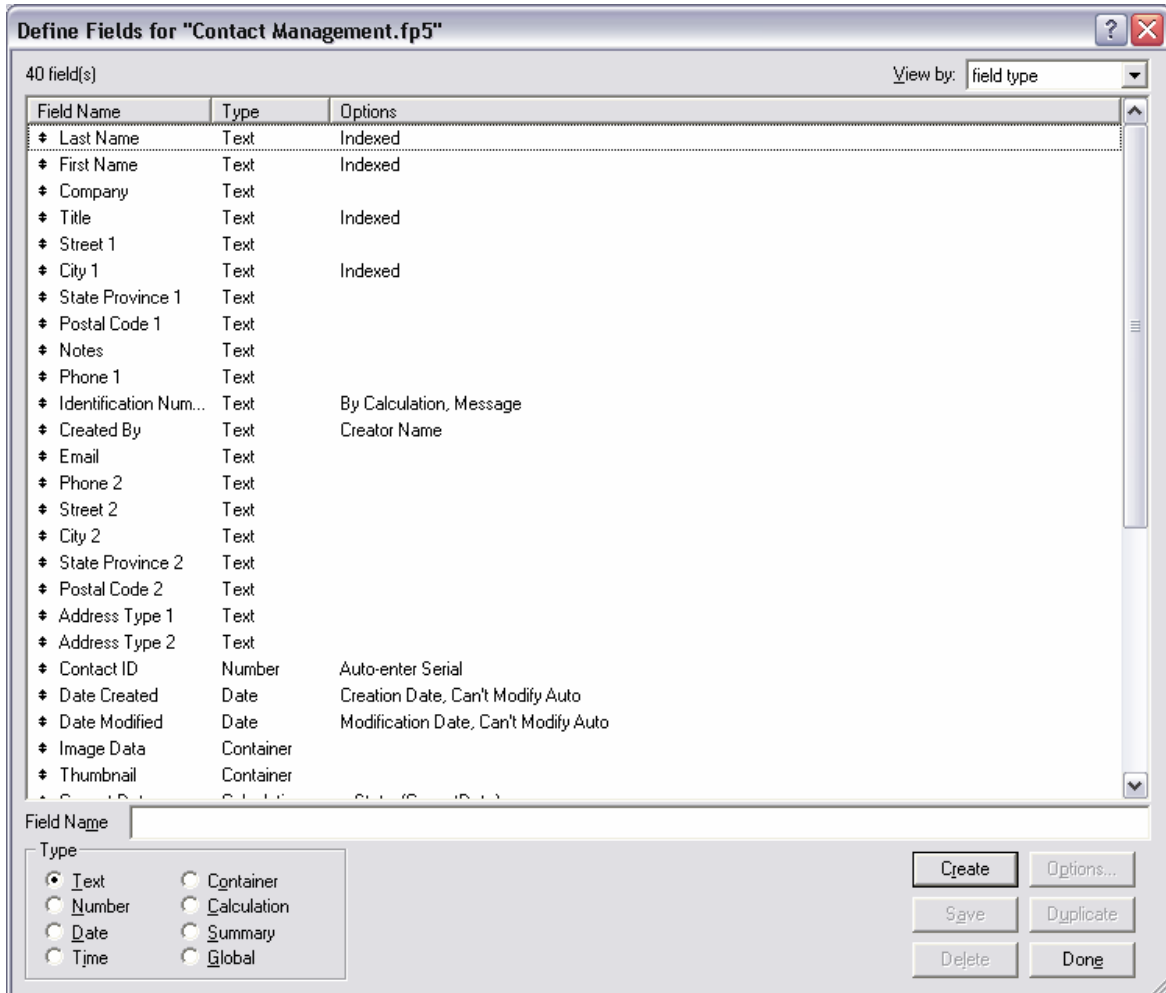
Estimated Time To Complete: 15 minutes

As I mentioned before, we won't concern ourselves with globals, calculations or summary fields in the planning of our data model – because Servoy treats these types of fields differently than FileMaker does.

Global fields in FileMaker are physical columns (fields) in your table (database). In Servoy, globals are in-memory variables and NOT physical fields in your table - and they're different than the concept of using any column as a global (as in FileMaker 7). Likewise, calculation fields and summary fields in Servoy are NOT physical fields – but rather are "virtual" columns that can be displayed or used in other calculations. We'll get into more detail about globals, calculations and summaries a little later when we're creating the Servoy solution.

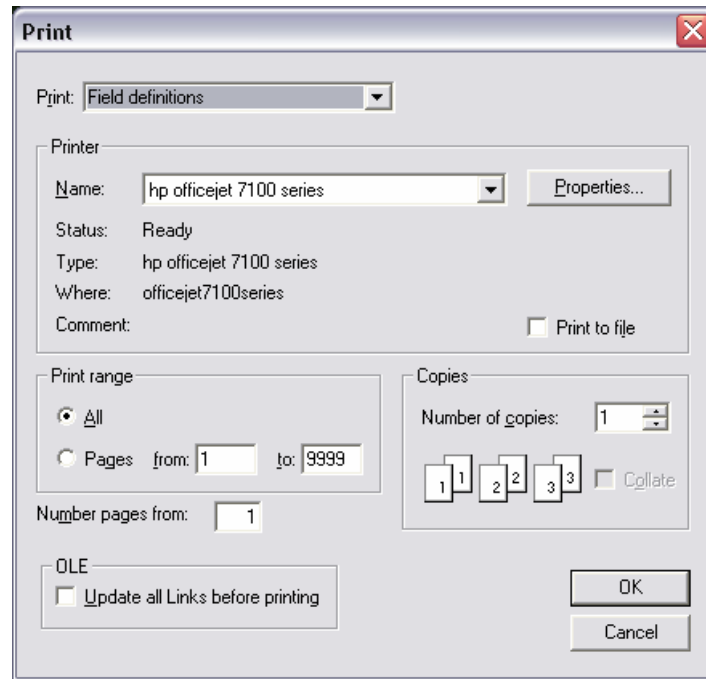
By looking at the printed copy of the layouts, you can get an idea of the data fields used in the solution.

Another thing you'll want to do at this point – is to open up "Define Fields" from the "File" menu in FileMaker, and sort the fields by "Type" (by clicking the word "Type" at the top of the list). This will show you the names of the fields in the existing solution – and give you an idea of the columns we'll have to create.



The reason we're sorting the list by Type rather than by Name is because we're only concerned with the fields that are Text, Number, Date, and Time fields. Now print out the field definitions by clicking the "Done" button and choosing "Print" from the "File" menu.

Choose the option for “Field definitions” from the pop-up menu that currently reads (“Records being browsed” or “Current record”).



We’ll use this list of fields along with our printed layout list to determine the bare minimum of fields that will make up our new data model.

Here’s the list of fields we wind up with:

Field Name	Type
Contact ID	Number
First Name	Text
Last Name	Text
Company	Text
Title	Text
Email	Text
Notes	Text
Image Data	Container
Thumbnail	Container
Address Type 1	Text
Address Type 2	Text
Street 1	Text
Street 2	Text
City 1	Text
City 2	Text
State Province 1	Text
State Province 2	Text
Postal Code 1	Text
Postal Code 2	Text

Phone 1	Text
Phone 2	Text
Date Created	Date
Date Modified	Date

Looking at this list of fields – there are a few things that we need to address in the design of our new data model:

Duplicate field names with numbers after them. Whenever you see fields with a number after them (i.e. Street 1, Street 2, City 1, City 2, etc.) – this should immediately raise a little red flag in your mind. Why did the developer choose to have multiple fields that store similar types of data in the same file instead of creating a related table to hold the information? Is there a good reason to keep this design, or will it limit our new solution? In this case, since the base solution is a template file – I’m pretty sure that the developer wanted to keep everything in an easy-to-use single file design.

This brings up another major difference between FileMaker and Servoy:

In FileMaker 1.x-6.x, each “database” is a single file containing a single “table” of data. In the world of SQL (and FileMaker 7), a “database” is a “bucket” that can have many tables, scripts (stored procedures), triggers, functions, user privileges, and other information inside it. We refer to the objects that hold data as “tables”. It’s not uncommon for a single SQL database to have dozens (or sometimes hundreds) of individual data tables in them.

A quick terminology note:

In Servoy – you create “solutions.” With FileMaker, you create “databases.”

In Servoy – the “wrapper” (the thing you’re creating) is called a “solution” whereas in FileMaker the “wrapper” is a “file” (or “database” or also “solution”) that has a single window.

Servoy solutions can have many different forms that reference many different tables; different physical databases; and/or even remote databases that are running at a different location (such as at a hosting facility on the Internet).

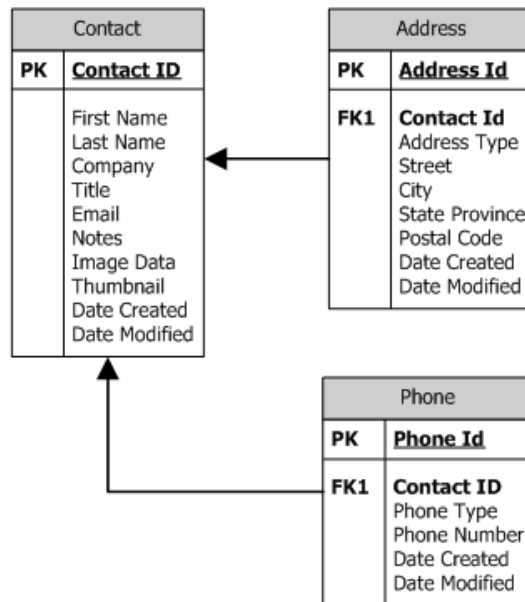
Keeping these differences in mind – we can design our data in a way that makes the most sense in terms of future functionality and enforce data normalization (where similar data is stored in a separate table and referenced by an ID).

Let's take a look at the differences in these two approaches (flat file versus normalized data) and determine the impacts of both designs in terms of complexity and functionality.

De-Normalized Schema

Contact	
PK	Contact ID
	First Name Last Name Company Title Email Notes Image Data Thumbnail Address Type 1 Address Type 2 Street 1 Street 2 City 1 City 2 State Province 1 State Province 2 Postal Code 1 Postal Code 2 Phone 1 Phone 2 Date Created Date Modified

Normalized Schema



A **schema** is the design of your database.

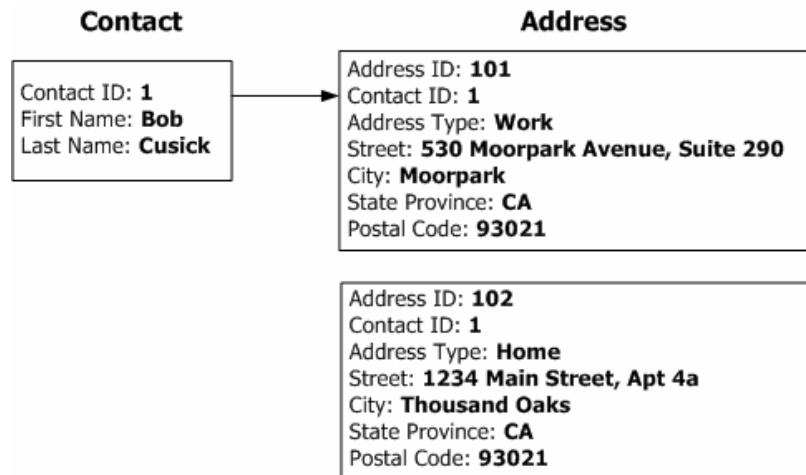
In most cases, you would want to create a *normalized* schema – a design in which there is a separate table that stores each type of distinct data. For example, rather than having fields named "Street 1", "City 1", etc. you would have a separate table called Address that would allow you to add as many addresses as you want – all linked to the contact's primary key (PK – Contact ID) via a foreign key (FK - Contact ID).

There may be times, however, that you would want to create a *de-normalized* structure. Think about an application in which we will only ever have two addresses and two phone numbers. Using the de-normalized structure is an easy way to set up the links for Address 1 and Address 2 tabs – whereas using a normalized structure, we need to come up with a way to display the correct address record on the correct tab. The normalized structure would allow us to add many more addresses and many more phone numbers and not have to add columns to our table.

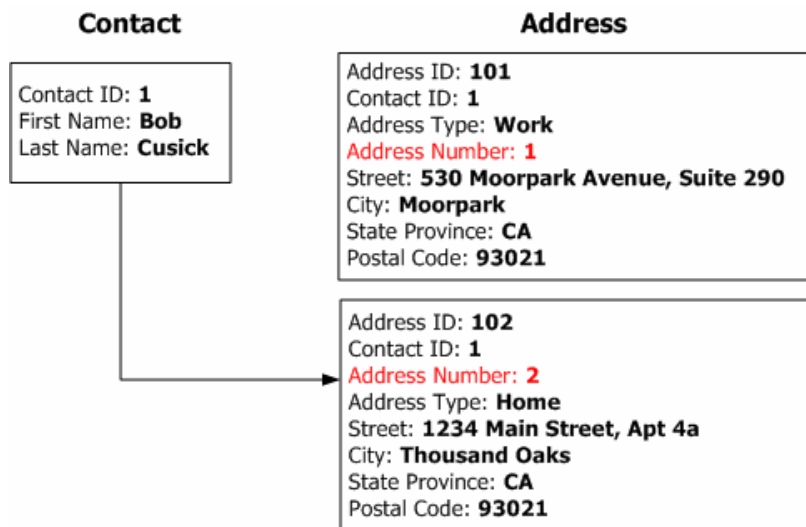
It's also important to consider how you might use the same tables in another solution. Remember – Servoy is simply the user interface – it's NOT the database. You could create this particular Contact Management solution as a standalone – but you may also want to use this same contact data again in a completely different solution where you would require more than one address and/or phone number. Because of this possibility, we're going to use the normalized data approach in our solution – both because it's flexible, and because it follows good database design.

So – how are we going to show data from one address record on one tab – and then another data record on another tab? And what about the phone numbers? What if we want to use a

specific type of phone number (such as “Work” or “Home”) on a letter or a form? All great questions! Let’s look more closely at some of these considerations. We know that the primary key (the unique identifier) for each contact is the Contact ID column – and we know that each address also has a foreign key (Contact ID) that we can use to link the contact and the address together. However, this doesn’t solve our problem – because both tabs would show the same address – the FIRST address for that contact.



We need to create a new identifier that will identify both the Contact ID and also which address number a particular address is – so we can show it on the correct tab. To accomplish this – we’re going to add a column to the Address table called Address Number. This will be an integer – representing the number of the address we want to display. We can then create a relation that looks at both the Contact ID and the Address Number (stored in a global) that will allow us to access the correct record on each Address tab.



We can accomplish the same type of methodology with our phone numbers. We’ll create a relation that looks at both the Contact ID and the Phone Type to display the correct item on our forms. Now that we’ve outlined the data needs of our application – and we’ve come up with a way to display that data – we’ll get busy creating our solution.

Chapter 3 – Creating the SQL Database

Estimated Time To Complete: 20 minutes

In Servoy – the basis of every solution is one or more *named database connections*.

These named database connections simply “point” to a particular database. The database can be on your local hard drive; on your LAN; on your WAN; or hosted at an Internet Provider. The physical location of the database and the database vendor don’t matter. You can have connections to Sybase databases; Firebird databases; MySQL databases; Microsoft SQL Server databases; and Oracle databases – all at the same time in the same solution. You can even display data from more than one different vendor’s database (i.e. Sybase and Oracle) and create portals or reports showing data from completely different databases running on two totally different machines!

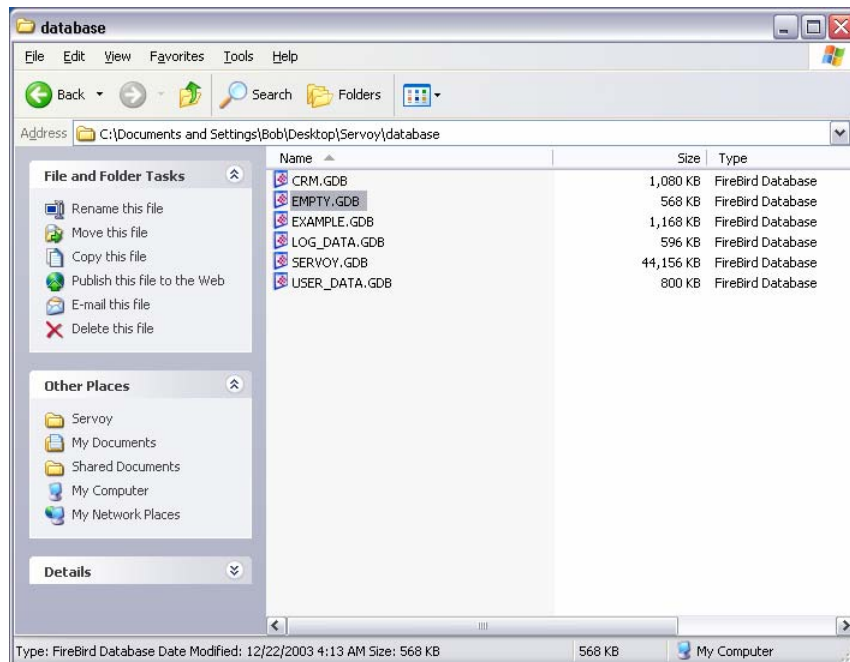
In order for this whole thing to work – you must start with a database. This can be an existing database – or you can create a new database from scratch. Because different database vendors have many, many different options for database settings, if you’re going to create a new database – you need to create it **outside of Servoy**. If your database already exists, all you need is a username and password to connect to it, the IP address (and sometimes the port – if it’s not running on the “default” port), and the name of the database. With these four pieces of information – you’re ready to build your solution.

Since this is a tutorial – we’ll take a look at how to create a new database in two different vendor’s databases: Firebird and Sybase iAnywhere ASA.

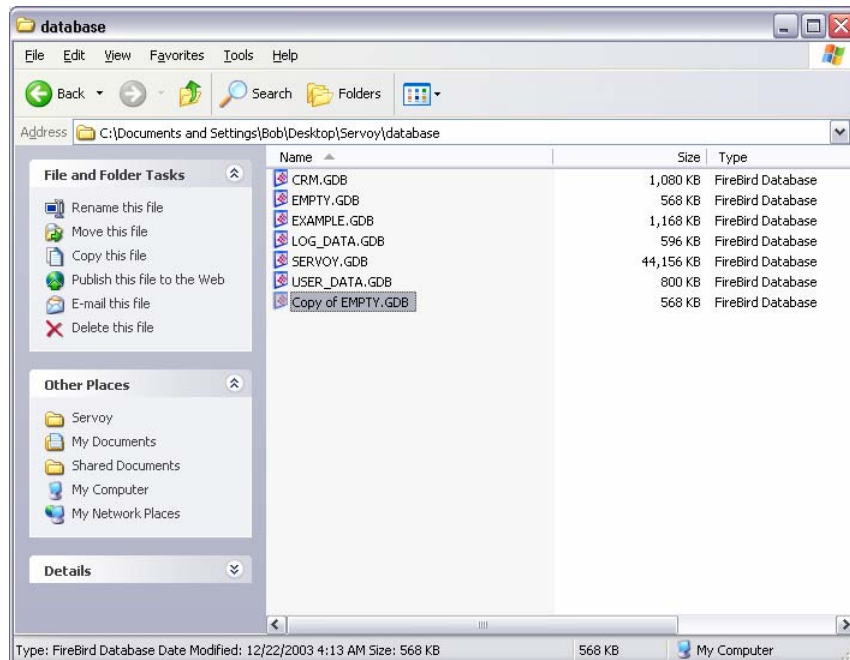
Creating a new Firebird Database

Versions of Servoy prior to 2.0, shipped with an open source database called Firebird (the open source version of Borland’s Interbase). Servoy shipped this “default” database as a way to ensure that users of Servoy could get started “right out of the box” without the need to install and configure their own SQL database. If you’re using Firebird with Servoy (yes, you CAN use Firebird as well as Sybase with Servoy at the same time!) – creating a new database is very straight forward:

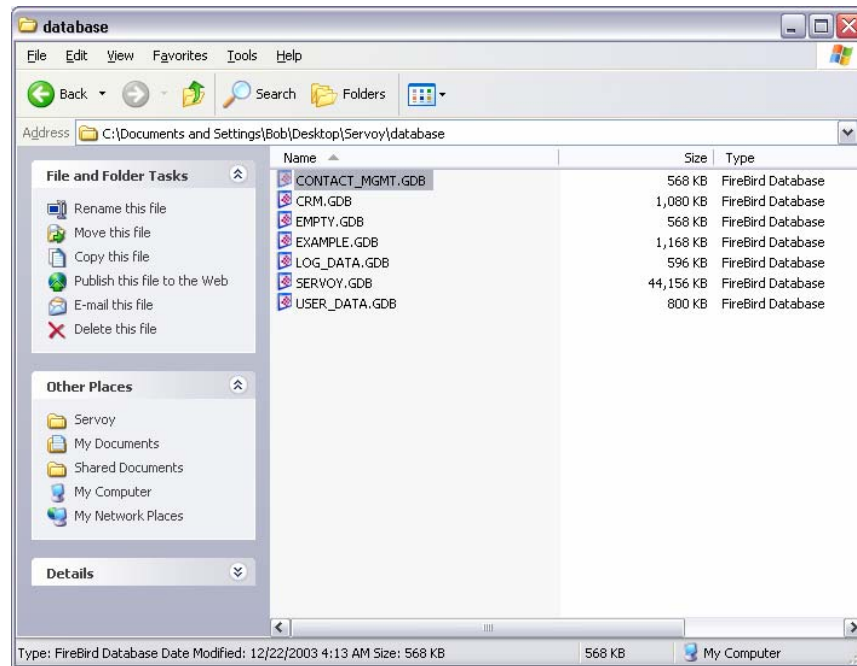
Locate your Servoy folder.
Double-click on the "database" folder.
Copy the file called "EMPTY.GDB".



Paste.



Rename the "Copy of EMPTY.GDB" to a new name. In our case, we will call our database "CONTACT_MGMT.GDB".



If you're using Firebird – that's it! You've now created an empty database without any data tables in it.

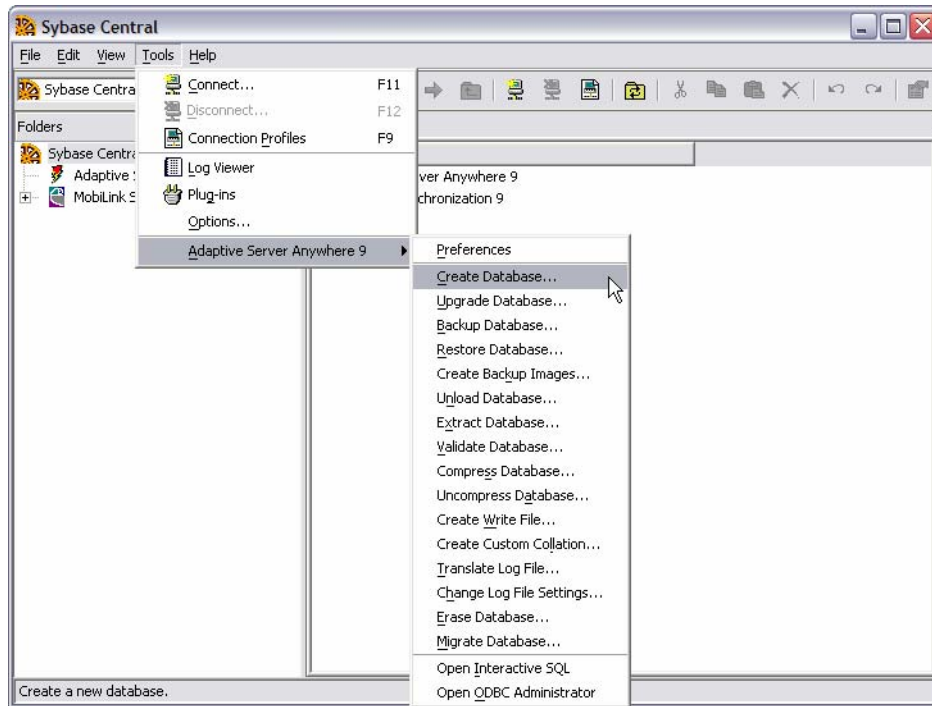
Creating a new Sybase iAnywhere ASA database

If you are using Servoy 1.0 to Servoy 2.0RC9 – USE FIREBIRD and the Instructions above unless you already have a separate license for iAnywhere ASA.

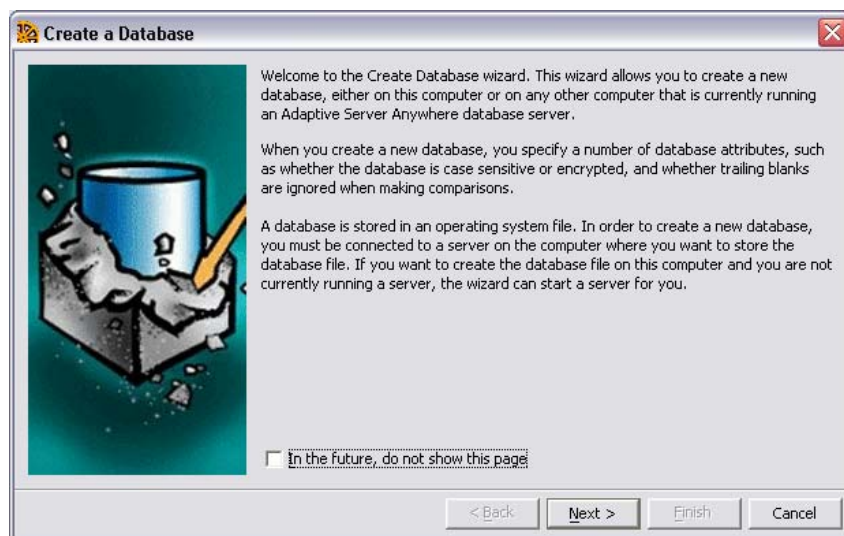
Beginning with Servoy 2.0 – the default database that ships with the product was changed from Firebird to Sybase iAnywhere ASA (Adaptive Server Anywhere). iAnywhere ASA is an enterprise class, scalable and extremely high performance database. The database files are also binary compatible between platforms – which means you can use a database file developed on a Macintosh directly on a PC without any conversions or data imports. Setting up a new Sybase database, and linking it to the Servoy application is a very straight-forward process – but not as simple as copying and pasting!

To create a new Sybase database – we’re going to use the free utility called “Sybase Central”. If you don’t have a copy of Sybase Central – you can download it as part of the “SQL Anywhere Studio v9” from <http://www.ianywhere.com/downloads/index.html>.

Open up Sybase Central and choose “Adaptive Server Anywhere 9” from the Tools menu and then select “Create Database...” from the context menu.

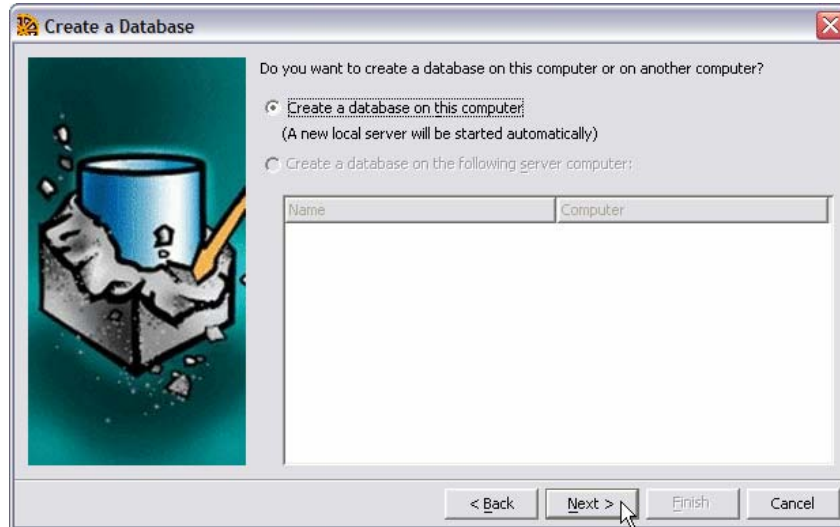


This will start up a wizard that will help you to configure the new database.



There are LOTS of different settings available when creating a new Sybase iAnywhere ASA database – and using this wizard – I'll help you through the choices.

Click "Next" to continue.

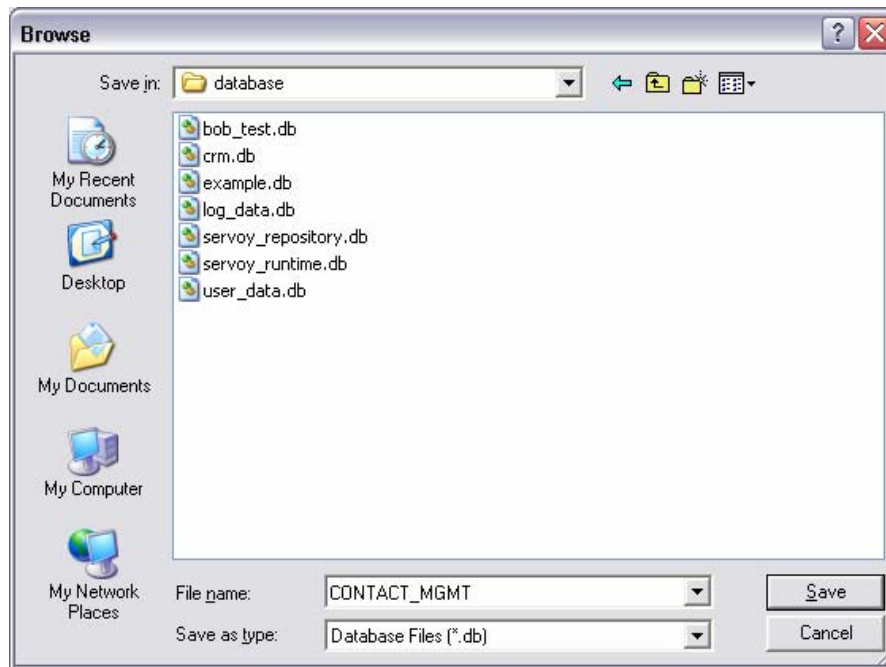


Click "Next".

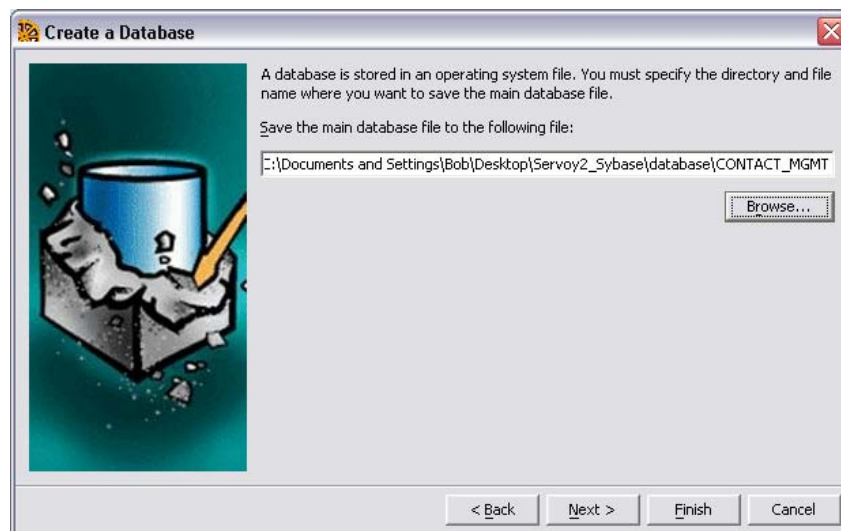


Click the "Browse..." button and locate the "database" folder of your Servoy folder.

Give your database the name of CONTACT_MGMT.



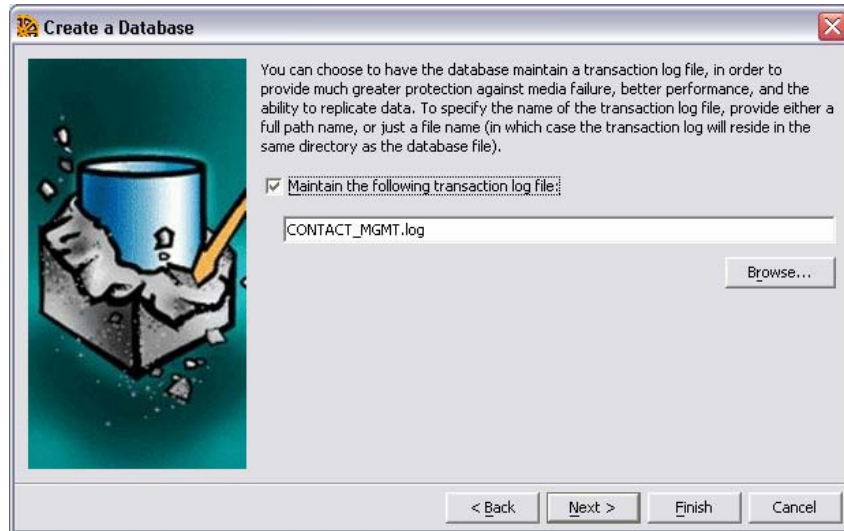
Click "Save" to dismiss the dialog and return to the wizard.



Click "Next" to continue.

You'll be asked whether you want to create a transaction log. A transaction log will *save every action taken to every table*.

So every time you create a record, update a record, or delete a record – the transaction will be saved in a log file. This option can help you re-create your data in case something goes wrong; allow you to replicate your data to a different Sybase database; and provides much better performance of the database itself.

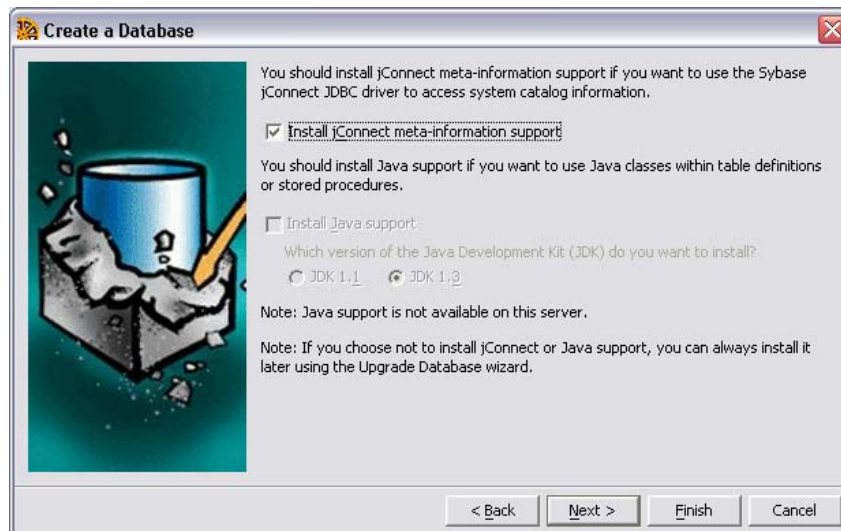


Click "Next". You'll now be asked if you want to create a mirrored log file. This does just what it sounds like – it will keep an identical copy of the log file in a different location. In most cases – you won't want to make a mirror copy of the log file as it will affect the performance of your database.

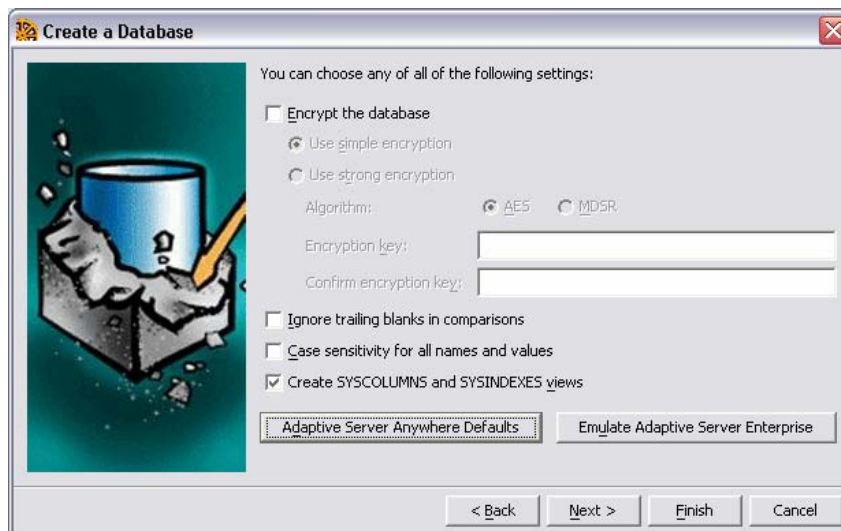


Click "Next" to continue.

On this screen – YOU MUST choose to “Install jConnect meta-information support”. This is because Servoy uses JDBC to connect to the Sybase iAnywhere ASA database. If you do not check this option – YOU WILL NOT BE ABLE TO CONNECT to your database.



Click “Next”. This screen is where you can specify database-specific features.



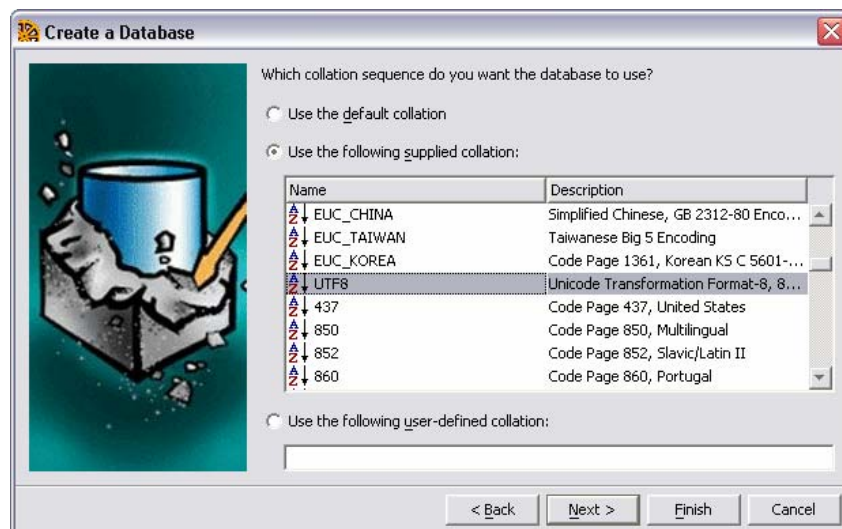
You can choose to natively encrypt the data in your database; ignore trailing blanks when using comparisons (for example, you have “Bob ” and you want the database to ignore the trailing space – similar to “trim” in FileMaker); and case sensitivity for all names and values.

The **default** for Sybase is to use *case in-sensitive* values. This means that when your users perform a search for “Bob” that entering “bob” or “bOb” or “boB” will still find the value “Bob”. This is DIFFERENT than Firebird – which IS case sensitive. Case insensitivity is really a great feature – and I recommend you leave this option Unchecked.

Click "Next". On this screen you can choose the database page size to use (the size, in bytes that the database uses to hold data "pages"). Unless you have a very good reason for changing it – keep the default size of 2048.



Click "Next". This screen allows you choose the "collation" or the sorting options. You can choose from a large selection of foreign languages or even supply your own collation if you want to. The discussion of using different collations is way outside the scope of this tutorial (for more information – check out the iAnywhere web site). For now, click "Use the following supplied collation:" and choose "UTF8". This option will work cross-platform and is also the same setting used when you setup a new Sybase ASA connection from within Servoy.

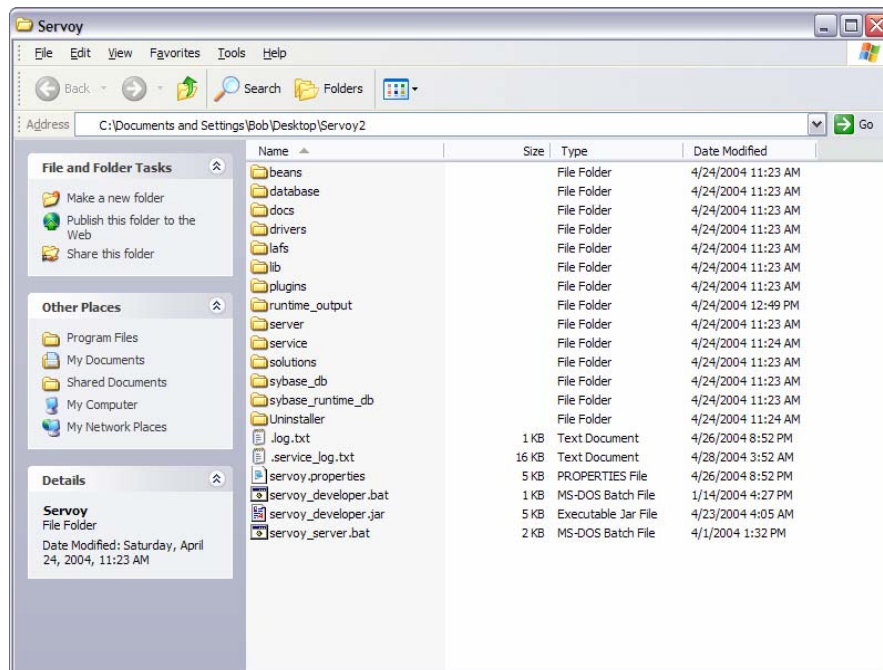


Click "Next". This final screen asks you whether you want to connect to the database (in Sybase Central) after creation. If you choose to connect – you can begin creating your tables and defining your columns right way – within Sybase Central.

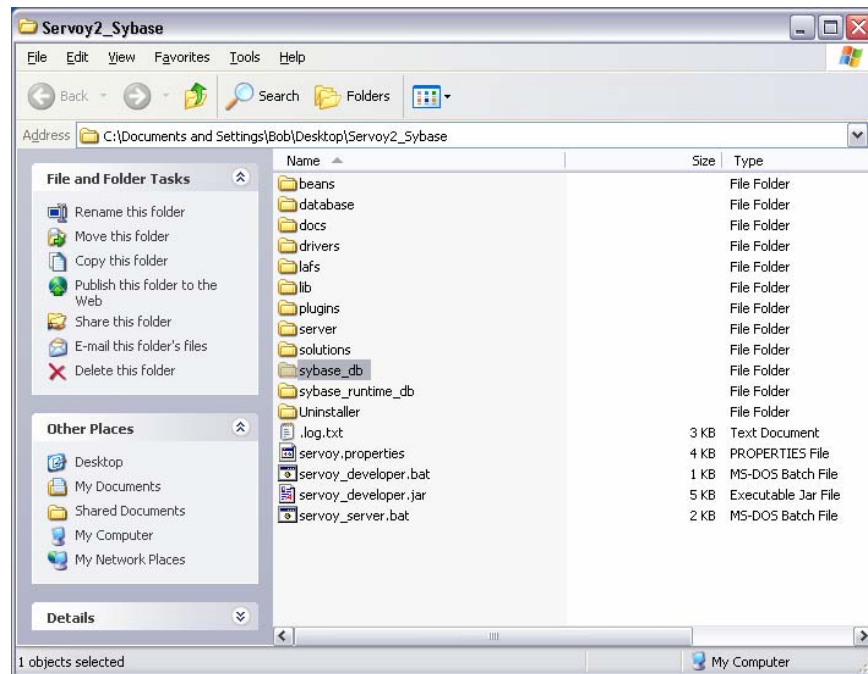
For now, DON'T check the "Connect to the new database" option and click "Finish".



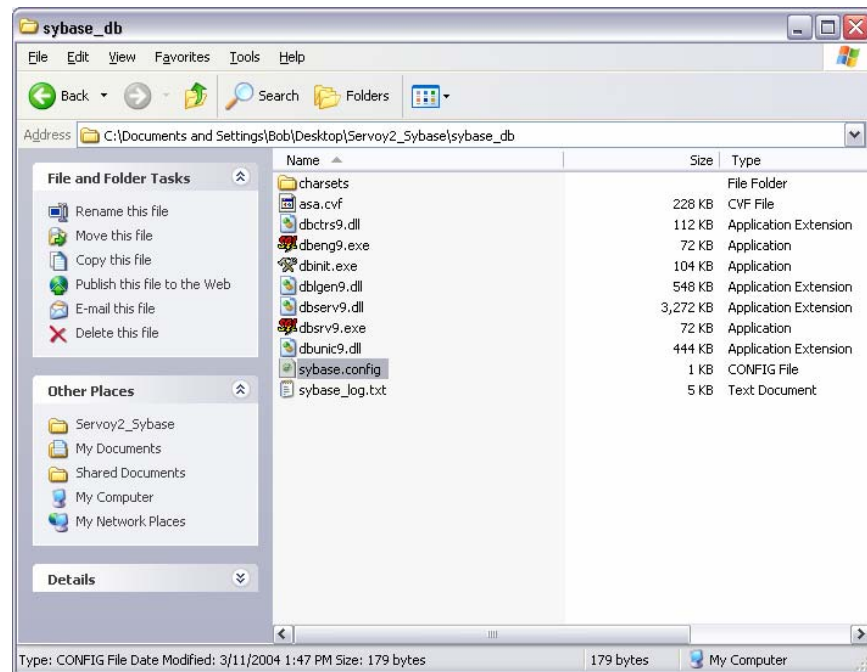
Congratulations! You now have a new, empty database.



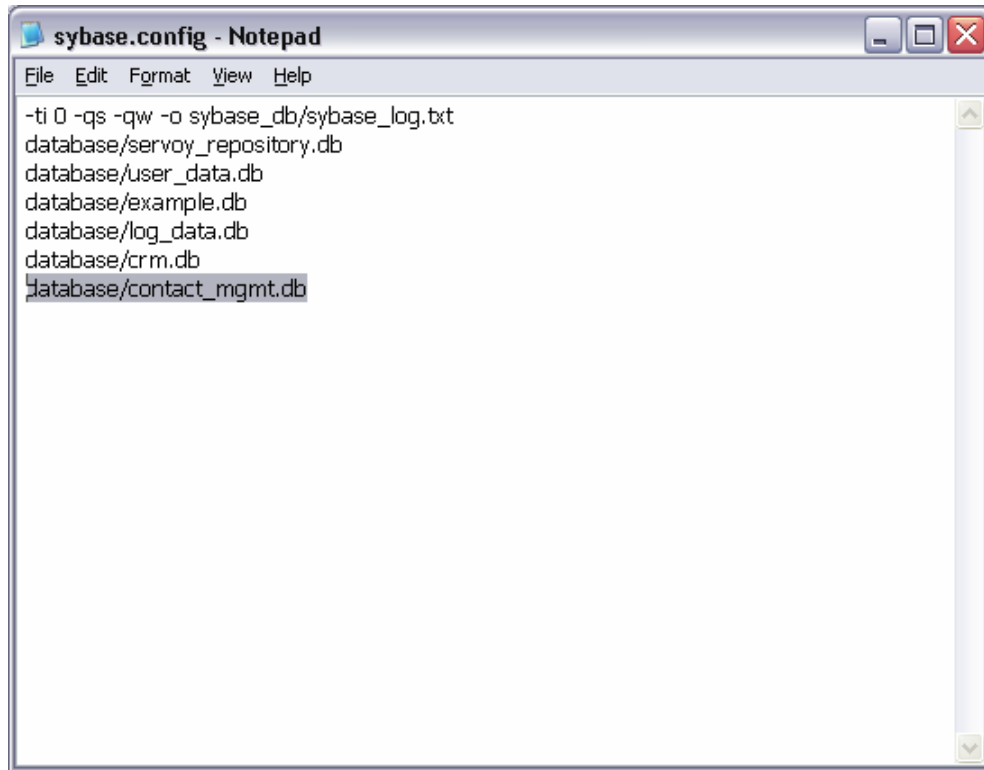
However, we're not quite done with the configuration yet – we need to "tell" the servoy_repository that it should start up our new database when it starts up. To do that – we need to modify one more file. Exit Sybase Central – and locate your "Servoy" folder.



Double-click the "sybase_db" folder and open the file called "sybase.config" with a text editor so we can add our new database "contact_mgmt.db" to the list of databases Servoy should open when launched.



Because we've saved our new database in the "database" folder – don't forget to include "database/" (no quotes) in front of the database name.



Save and close the file.

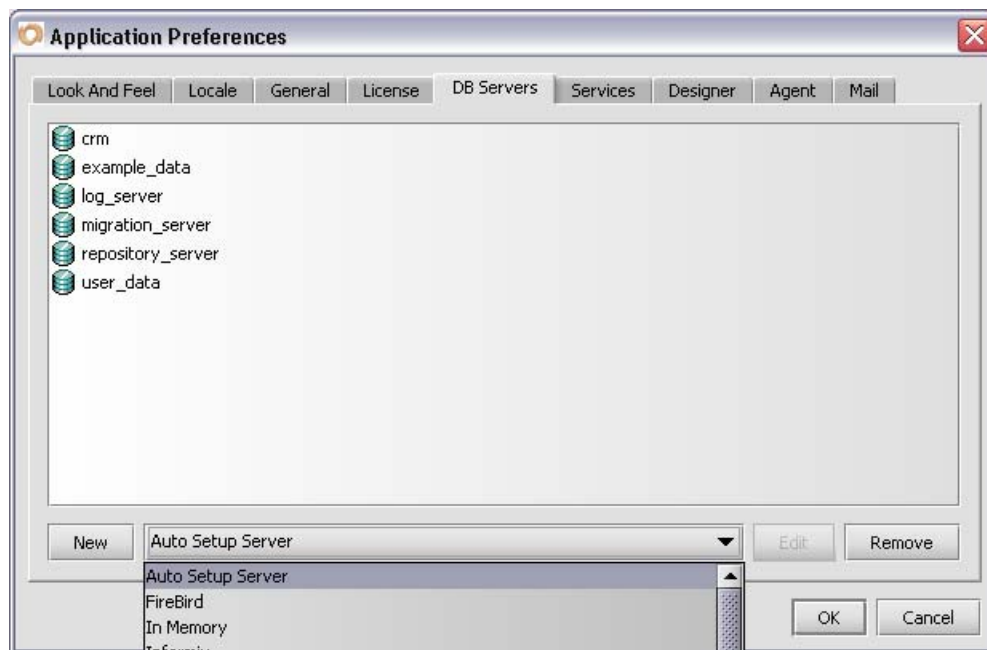
OK – now we're all ready to begin building our Servoy solution.

Chapter 4 – Creating the Servoy Solutions and Data Tables

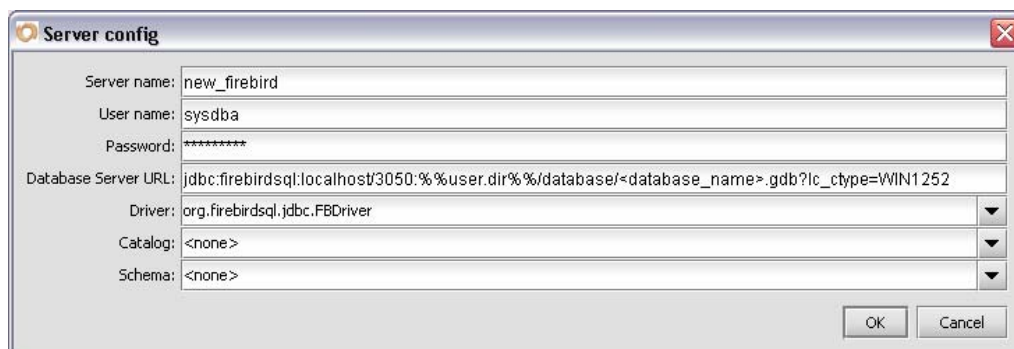
Estimated Time To Complete: 20 minutes

It's time to launch Servoy and get started with our solution. The first thing we need to do is to create a connection to our new database. After that, we'll create our solution and then create the data tables and basic forms for our solution.

Once you've double-clicked on the Servoy icon and Servoy has started – click the "Cancel" button when you see the "Select Solution" dialog – and choose "Preferences" from the "Edit" menu. Click on the tab called "DB Servers".



If you're using a Firebird database – then choose "FireBird" from the popup menu and you'll see:

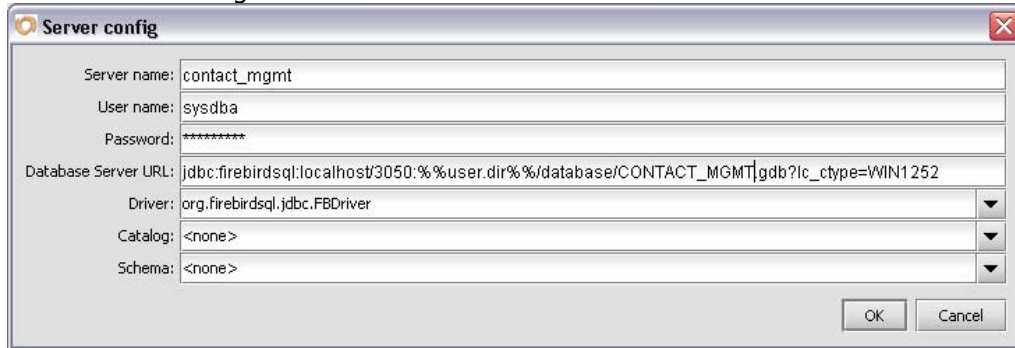


The "Server name" is the name YOU want to use to refer to the connection. Let's use "contact_mgmt" for the Server name.

Leave the default settings for the user name and password (unless you've changed the master password and username). The default User name is "sysdba" and the default Password is "masterkey".

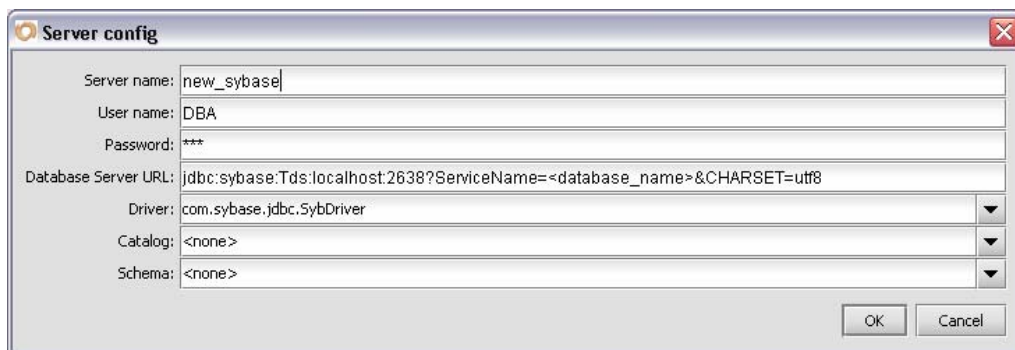
In the "Database Server URL" line, you need to change "localhost" to the IP address of your FireBird server (or leave it "localhost" if you have it on the same machine), and then replace "<database_name>.gdb" with "CONTACT_MGMT.gdb" (no quotes).

You can leave the Catalog and Schema set to "<none>" – and click "OK."



If you didn't get any errors – your connection is now set up and valid. If you did get an error – make sure your User name and Password are correct.

If you're using a Sybase iAnywhere ASA database – then choose "Sybase ASA" from the popup menu and you'll see:

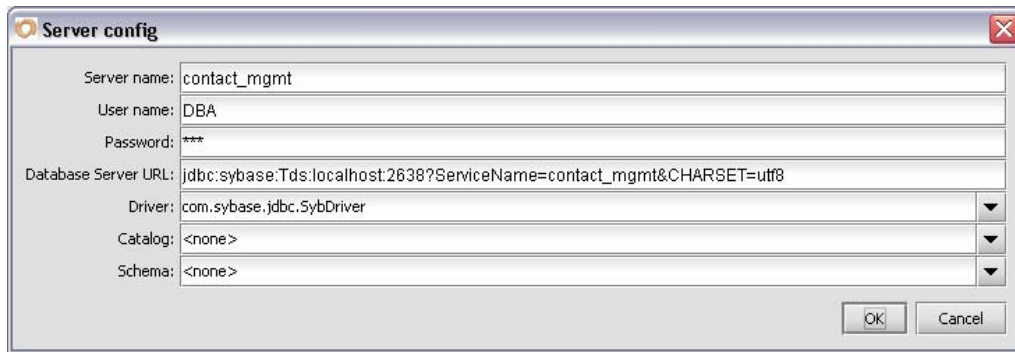


The "Server name" is the name YOU want to use to refer to the connection. Let's use "contact_mgmt" for the Server Name.

Leave the default settings for the user name and password (unless you've changed the master password and username). The default User name is "DBA" and the default password is "SQL".

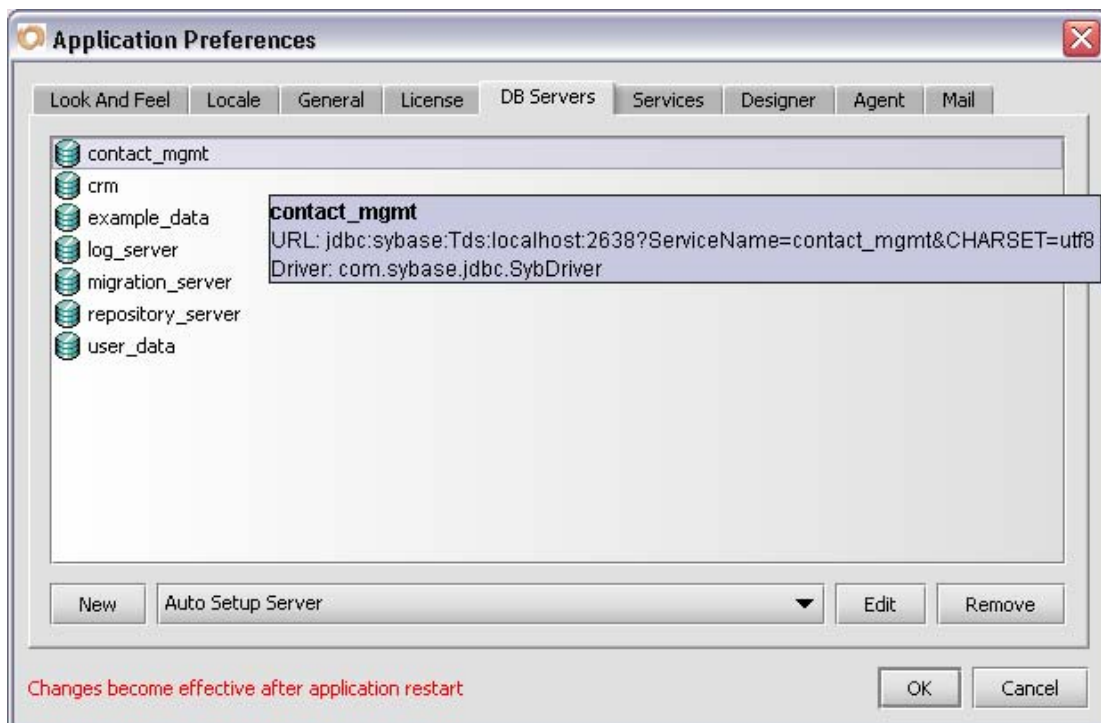
In the "Database Server URL" line, you need to change "localhost" to the IP address of your Sybase server (or leave it "localhost" if you have it on the same machine), and then replace "<database_name>" with "CONTACT_MGMT" (no quotes).

You can leave the Catalog and Schema set to "<none>" – and click "OK."

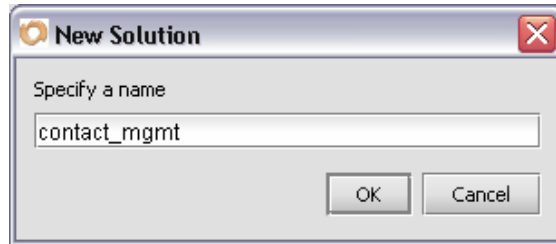


If you didn't get any errors – your connection is now set up and valid. If you did get an error – make sure your User name and Password are correct.

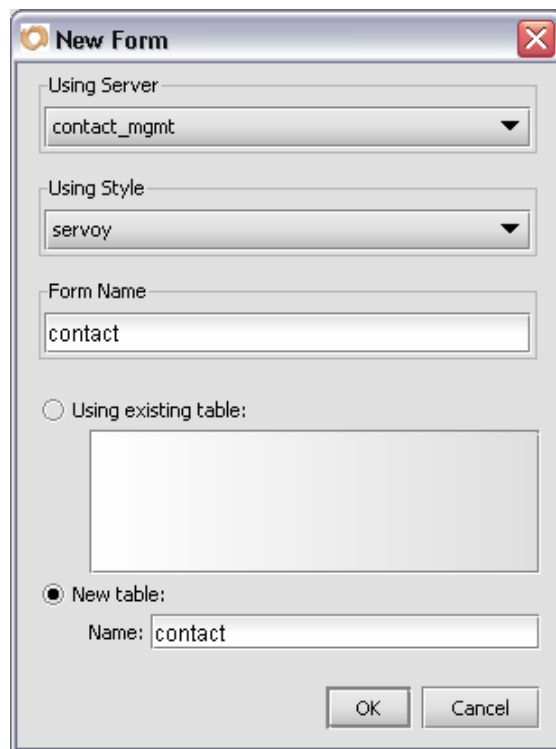
You'll notice that after your connection is added – red type appears at the bottom of the dialog saying that changes are effective only after restart. So, restart Servoy and we'll begin creating our solution.



Once you've restarted Servoy – click the "Cancel" button when you see the "Select Solution" dialog – and then choose "New Solution" from the "File" menu. Enter contact_mgmt (spaces aren't allowed in the solution name) and click OK.

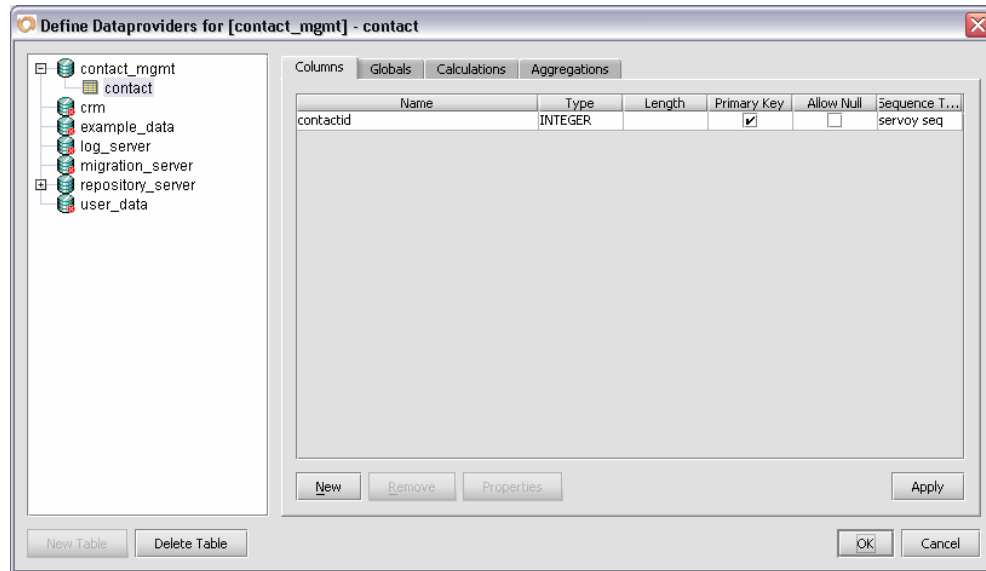


Once you've created the solution – the first thing that Servoy assumes you want to do is to create a form that will display data. Choose your connection called "contact_mgmt" from the "Using Server" pop-up menu. Because we don't have any data tables yet, there will be nothing that appears in the list box below "Using existing table".



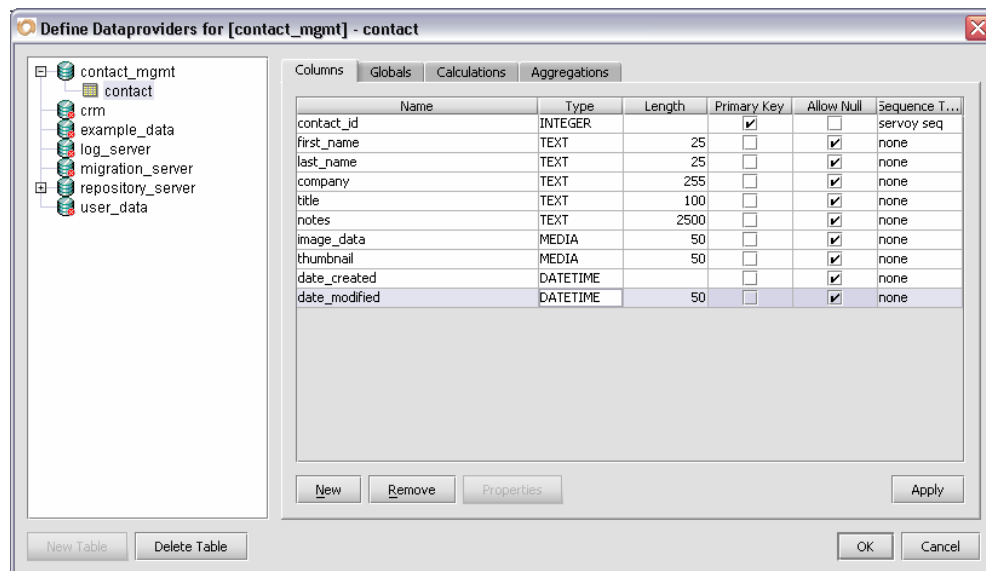
Instead, we'll create our first table – contact. Enter "contact" (no quotes) in the field below "New Table", make sure the radio button next to "New Table" IS selected and click "OK".

The next dialog is called the “Define Dataproviders” dialog and is equivalent to the dialog “Define Fields” in FileMaker. You’ll see a list of all your database connections on the left hand side (those databases that aren’t currently available are marked with a red “x”) – and if you see a “+” next to the connection name – you can expand the connection to show all the data tables in the database pointed to by the connection name.



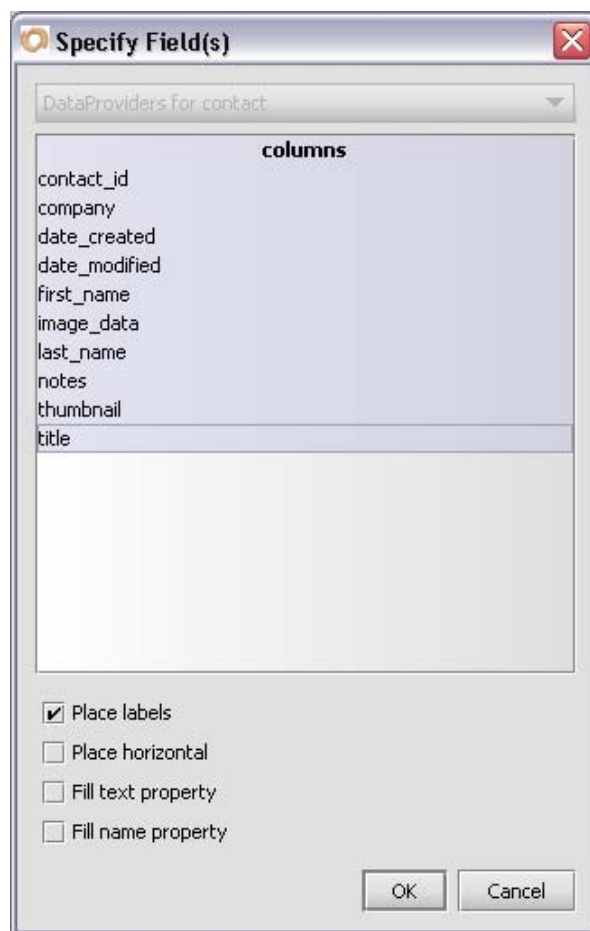
Our connection name is “contact_mgmt” – and you’ll notice that Servoy created a new table for us called “contact” and added a new field called “contactid” – that is an integer and marked as the Primary Key.

Change “contactid” to “contact_id” and add the remaining fields to match this list:



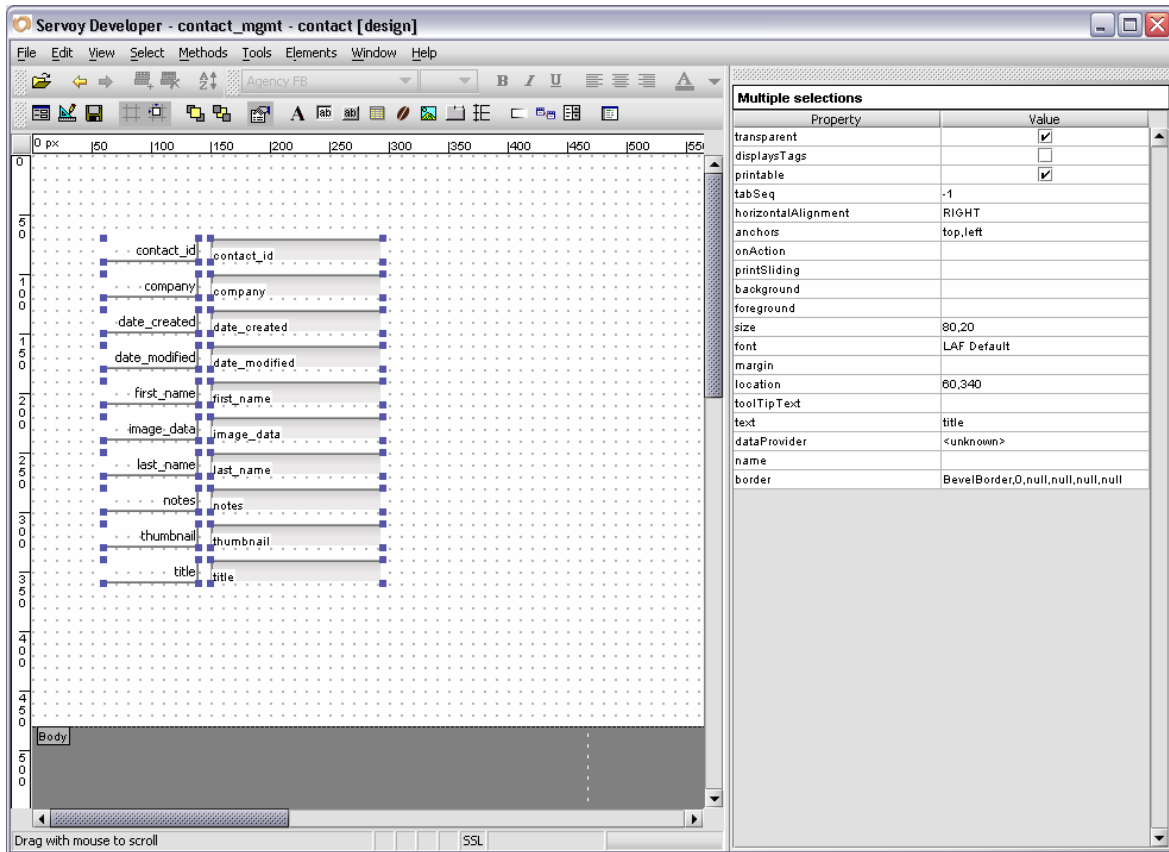
BEFORE YOU CLICK THE "APPLY" BUTTON – MAKE SURE that all the columns are spelled correctly and that you have set the data lengths on the TEXT fields correctly. This is because the Define Dataproviders dialog will NOT let you RENAME columns or change their lengths – you need to use Sybase Central (for Sybase ASA) or another 3rd party tool to access the database directly – once you click the "Apply" button. The reason for this is simple. Suppose you were connecting to a database that you didn't personally create. Now let's also suppose that same database is being used by a PHP application to put data up on the web. What do you think would happen if you just started changing the names of fields?? Yep, the web application would stop working. This is why you need to use 3rd party tools. if you want to make those kinds of changes – you need to "mean it" – because it may affect other applications and/or stored procedures, etc. etc.

Once you are happy with the spellings and data lengths of the fields – DO click the "Apply" button, and Servoy will create a form for you based on the contact table you just created.



DO click the "Fill name property" - I'll explain the functionality of this later. You can choose one or more fields to add to the new form. You can even choose discontinuous fields by holding down the CTRL key. To start with – we're going to add all the fields, so click on a field and press CTRL-A (PC) or COMMAND-A (Mac) to select all the fields, then click "OK."

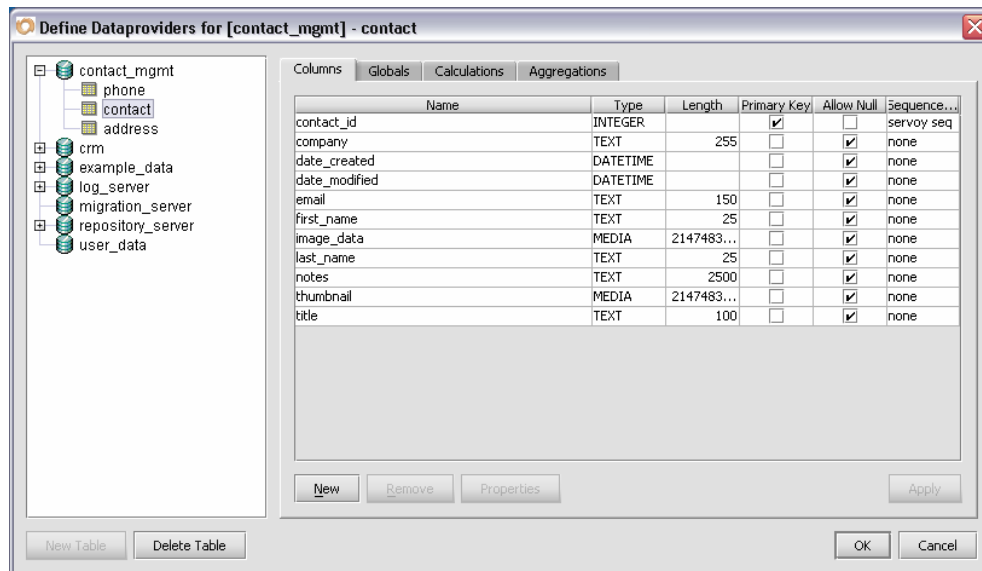
Servoy will place all the fields on the form – and place their labels to the left of the fields.



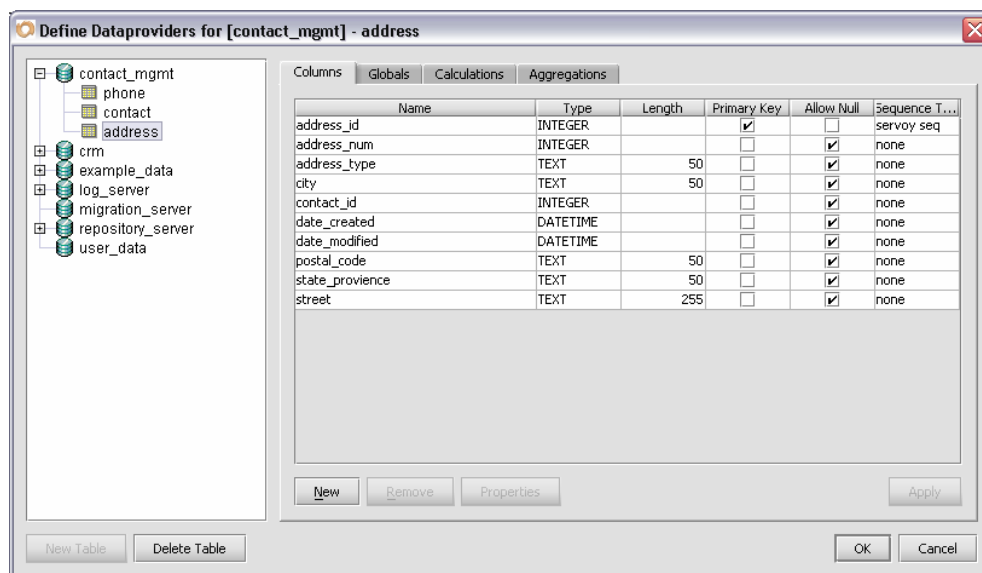
If you exit the Designer mode – press CTRL-L (PC) or COMMAND-L (Mac) – you can start adding and deleting records – all with no SQL programming!

Next, we're going to create the other data tables we need for our solution: Address and Phone.

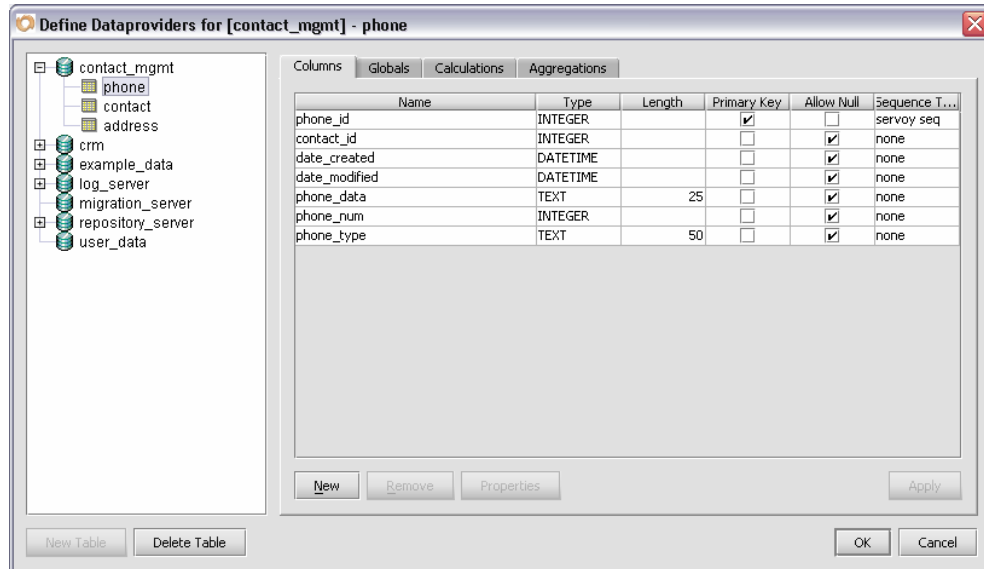
Go back into Designer mode – and choose “Dataproviders...” from the “Tools” menu:



Click on the “contact_mgmt” database connection – and click the “New Table” button on the bottom left of the screen. When you’re prompted to enter the name of the new table – enter “Address” (no quotes). Change “addressid” to “address_id” and add the remaining fields to match this list:



Now we're going to create our last table: "Phones." Click on the "contact_mgmt" database connection again – and click the "New Table" button on the bottom left of the screen. When you're prompted to enter the name of the new table – enter "Phone" (no quotes). Change "phoneid" to "phone_id" and add the remaining fields to match this list:



Now that we're finished defining the tables to hold our data – we can begin to build the user interface.

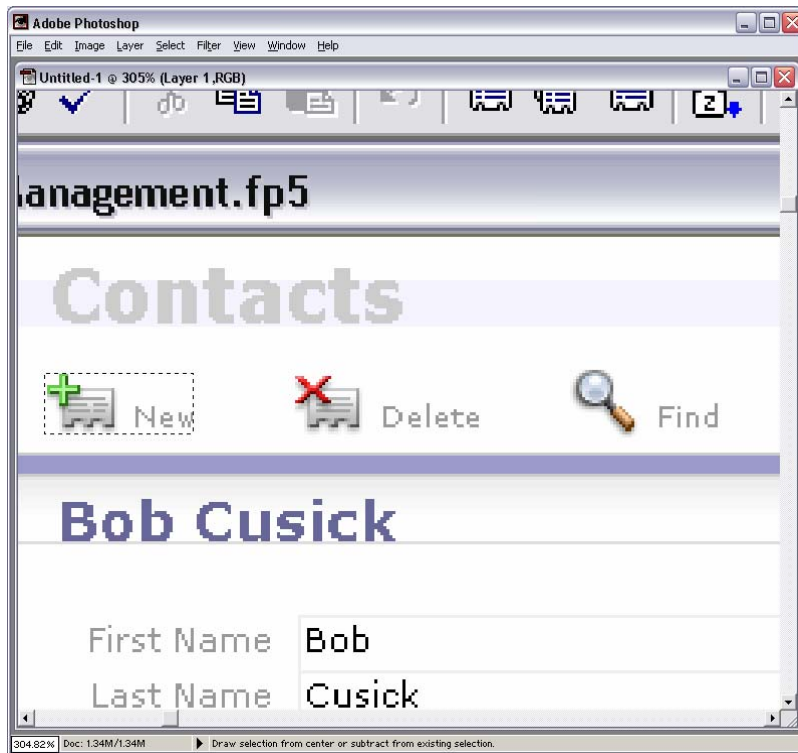
Chapter 5 – Building the User Interface – Part 1

Estimated Time To Complete: 1.5 - 2 hours

The first thing we're going to do in building the user interface, is make sure that we have all of the graphical elements we need. There are native elements we can create in Servoy – all the lines, type, etc. – but there are some elements like the "New", "Delete" and "Find" buttons and the "Form", "List" and "Table" buttons that we'll need to create graphics for.

To create the button graphics, I went into Browse Mode in the FileMaker solution – and captured the screen.

Then I opened the screen capture in Adobe Photoshop (you can use your favorite graphics editor); used the lasso tool to select each of the button areas; copied the graphic; pasted it into a new document; chose "Save For Web" and then saved the pasted graphic in .jpg format. I did the same thing for the "Form", "List" and "Table" graphics (in both grey and purple).



After capturing all the graphics – I saved the .jpg versions to a separate folder called “images”.

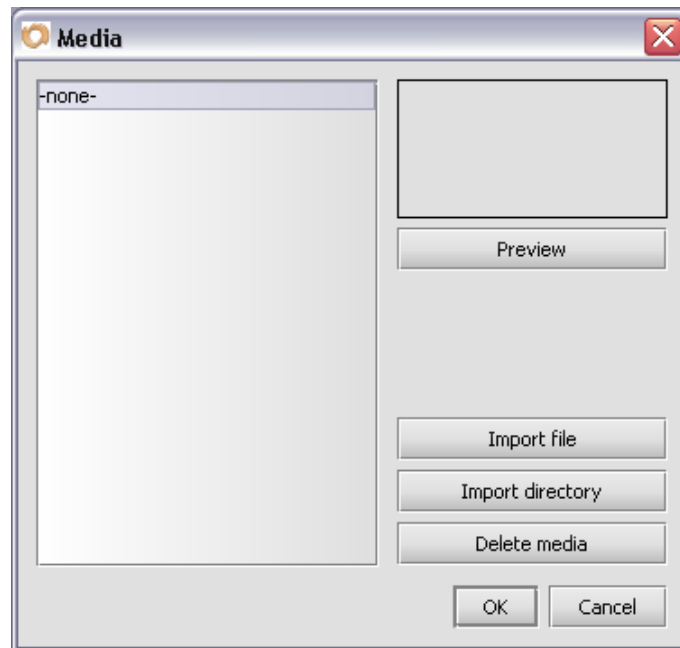
To make the graphics available in my Servoy solution – I use the image tool to import the entire folder of items into the solution. You will need to create images for the “New” button; “Delete” button; “Find” button; “Email” button; as well as the form button, list button and table button – in both grey and purple. Once you’ve captured everything and turned them into .gif or .jpg images – you’re ready to import them into the image library so you can use them in your solution.

Here’s how to import the images:

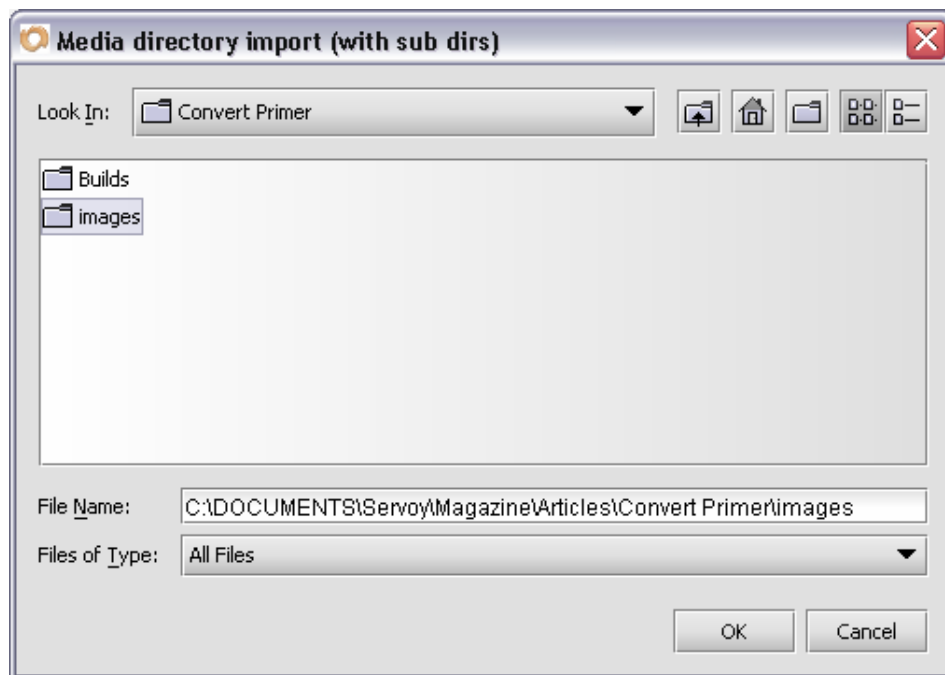
Capture all your images into a folder called “images” – make sure they’re in .gif or .jpg format.

Go into the Designer mode in your Servoy solution.

Click the Image tool in the toolbar – or choose “Place Image” from the “Elements” menu.

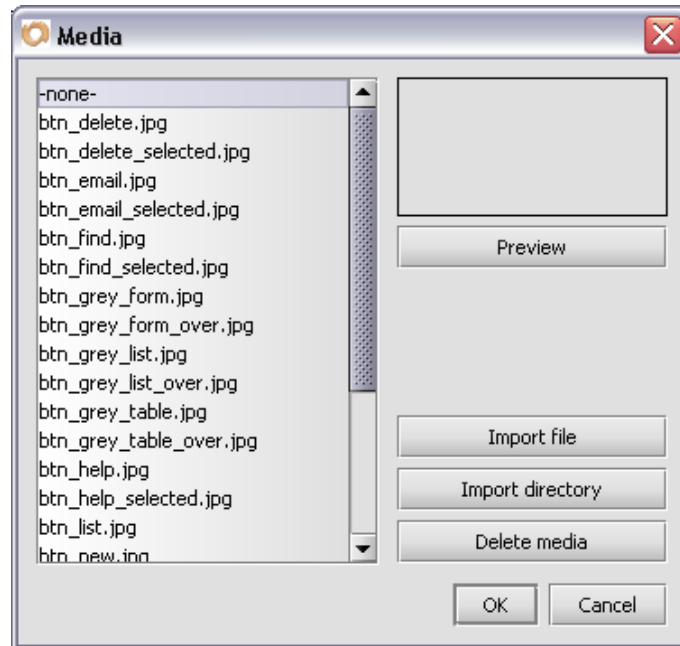


Click the “Import directory” button – and choose your “images” folder.



Click the “OK” button to import all the images into your Servoy solution.

Then click the "Cancel" button to dismiss the Media dialog.



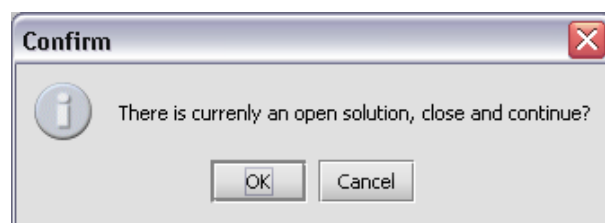
For those of you that don't want to take the time to capture all the elements – you can simply import the "images" folder from the download files. I've included a bunch of other images (like rollover images, etc. – plus all the properties assume graphics of a certain size – so it's a good idea to import the images folder from the download files.

Now we'll create and place all of the elements of the Main Address form. Remember – our goal is to make our Servoy solution resemble the FileMaker solution as closely as possible – so we'll place our objects using the size and location properties as well as the foreground and background (color) properties. If you don't want to create and place all of the objects – there's an easier way. With Servoy you can have multiple "Releases" of a single solution. Although you can only edit the latest (most recent) release, you can easily create many different releases as you go.

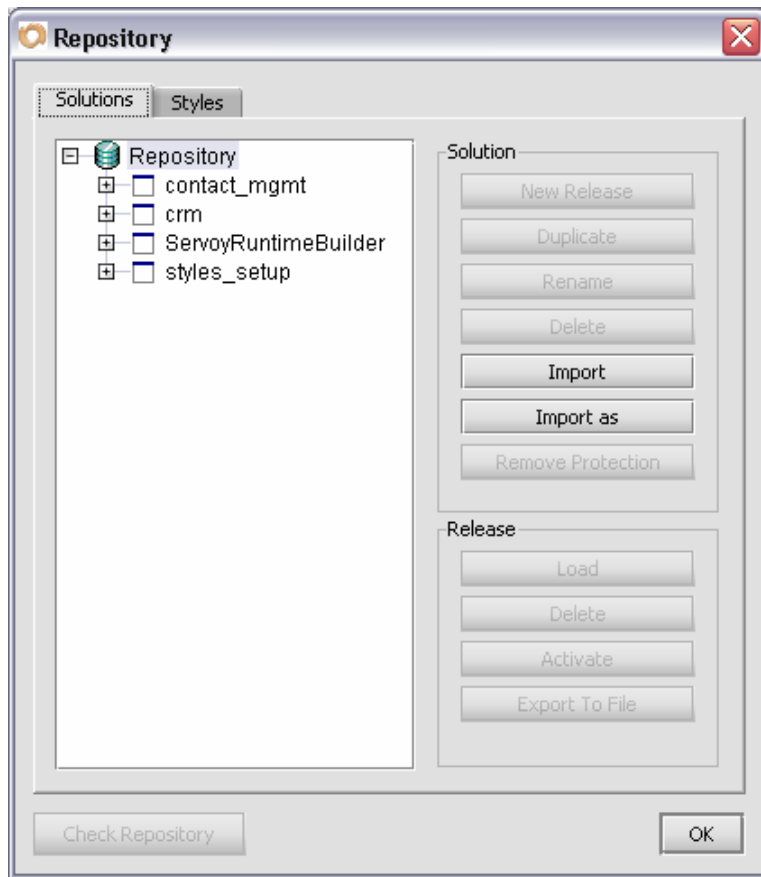
If you want "instant" results – you can import the file called "contact_mgmt_02.servoy" (in the "Builds" folder of the download files). Here's how to do it:

Choose "Repository..." from the "File" menu

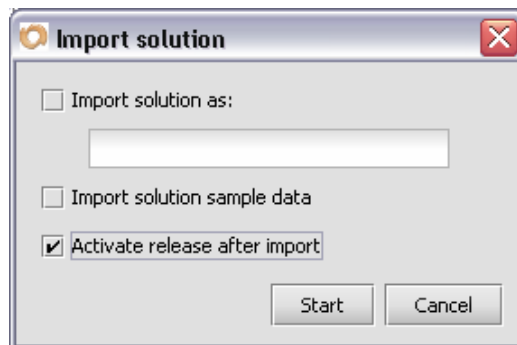
When you see the dialog "There is currently an open solution, close and continue?" – click the "OK" button.



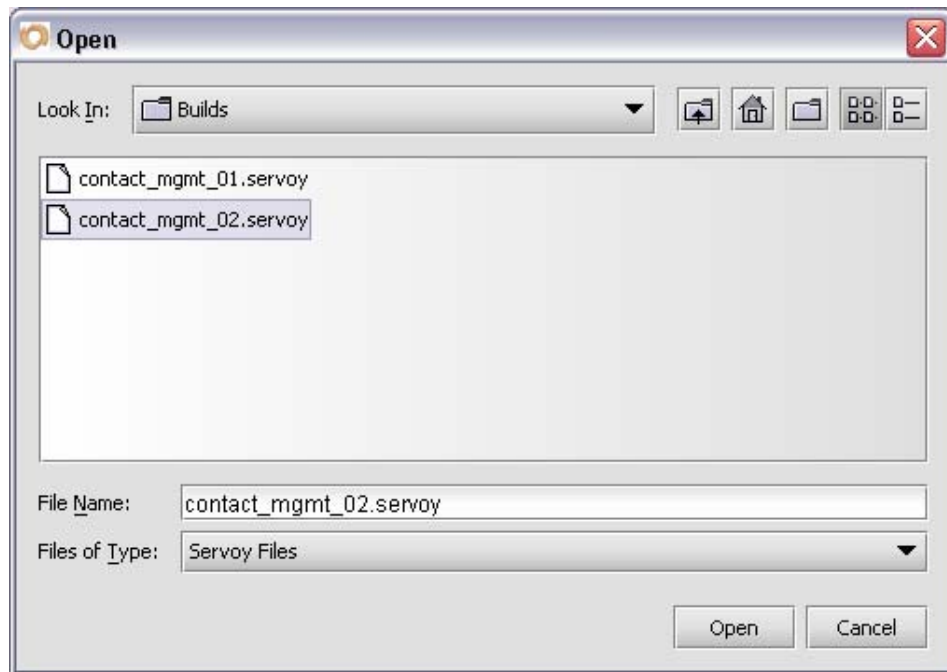
You will see the Repository screen appear – click on the “Repository” at the top – and then click the “Import” button.



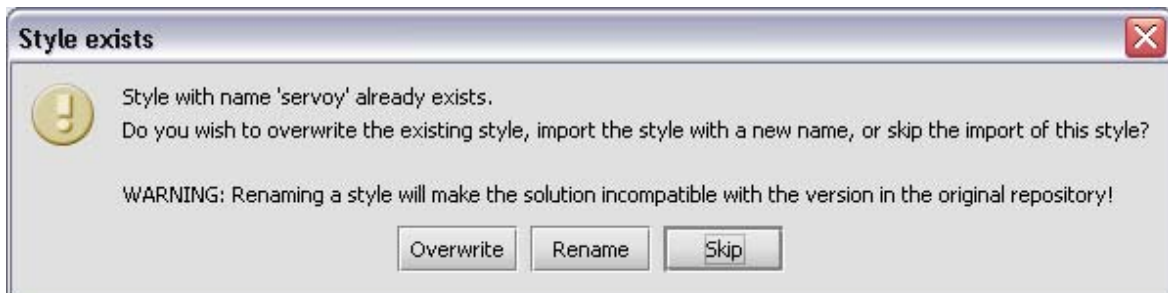
In the “Import” dialog – CHECK the option for “Activate release after import” and click “Start”.



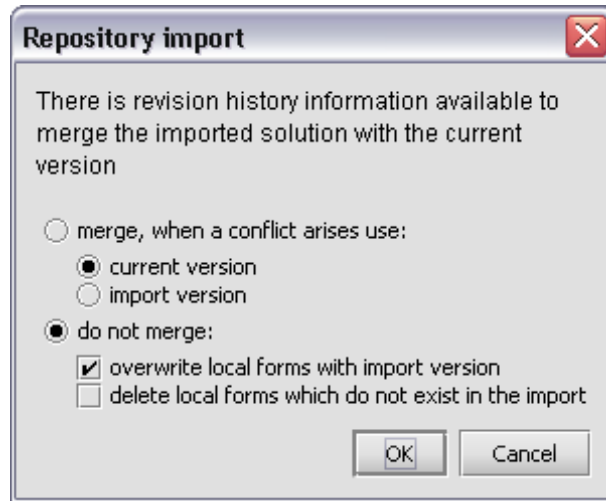
Locate the "Builds" folder in the download and double-click the file "contact_mgmt_02.servoy" (or select the file and click "Open").



You will see a dialog that tells you the style "servoy" already exists – and asks if you want to overwrite it – click the "Skip" button.



The next dialog is very interesting. It says that there is "version information" available – and asks if you want to "merge" the imported solution with the imported version.

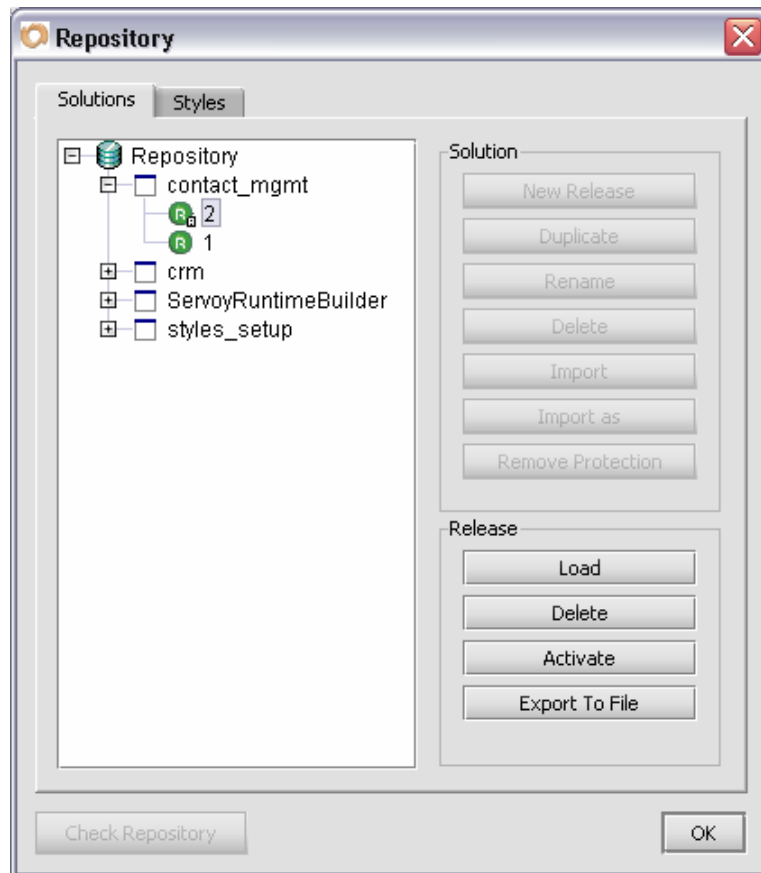


This is a very powerful feature of Servoy. Since a Servoy solution is not a database – just the user interface – you can have multiple people working on the same solution (even offline) – and then merge all the versions together. For example – if developer A worked only on forms 1-3 and developer B worked on forms 4-7 – they could "merge" their versions together to create a new release with BOTH of their work combined together.

For now, just accept the defaults – "do not merge" and "overwrite local forms with import version" – and click the "OK" button.

In a few seconds, you'll see "successfully completed". Click the "OK" button to dismiss the dialog and return to the Repository screen.

When you click on the “+” next to your contact_mgmt solution on the Repository screen – you’ll see that there is a new release (R2) with a little black “A” on it – signifying it’s the “active” release.



You can now click the “OK” button and then choose “Open Solution” from the “File” menu to open up the new release of the contact_mgmt solution.

For those of you that want to place the elements manually – follow the chart of elements and properties below to build and place all the elements:

Object Type	Property	Value
Form	Width	612
	View	Record view (locked)
	StyleName	<none>
	TitleText	Contact Main
	Background	color= Red:255, Green:255, Blue:255
	Name	contact_main
Body Part	Height	454
Rectangle Top line behind “Contacts”	Size	612,15
	Location	0,19
	Anchors	Top, Left, Right
	Border	Line: size=1, color= Red:245, Green:243, Blue:255

	Foreground	color= Red:245, Green:243, Blue:255
	Background	color= Red:245, Green:243, Blue:255
Label Contacts Label	Size	130,20
	Location	19,14
	HorizontalAlignment	Left
	Font	Verdana, 24pt, Bold
	Transparent	True (checkbox)
	Foreground	Color = Red:204, Green:204, Blue:204
	Anchors	Top, Left
	Text	Contacts
Label View Contact List text button	Size	95,20
	Location	386,16
	HorizontalAlignment	Left
	Font	Verdana, 9pt, Plain
	Transparent	True (checkbox)
	Foreground	Color = Red:153, Green:153, Blue:153
	Anchors	Top, Right
	Text	View Contact List
Label Dividing line between text buttons	Size	14,20
	Location	486,16
	HorizontalAlignment	Center
	Font	Verdana, 9pt, Plain
	Transparent	True (checkbox)
	Foreground	Color = Red:153, Green:153, Blue:153
	Anchors	Top, Right
	Text	
Label Address Labels text button	Size	84,20
	Location	509,16
	HorizontalAlignment	Left
	Font	Verdana, 9pt, Plain
	Transparent	True (checkbox)
	Foreground	Color = Red:153, Green:153, Blue:153
	Anchors	Top, Right
	Text	Address Labels
Label "New" button	Size	56,26
	Location	19,45
	HorizontalAlignment	Left
	VerticalAlignment	Center
	ImageMedia	btn_new.jpg
	RollOverImageMedia	btn_new_selected.jpg
	MediaOptions	Crop
	Transparent	False (checkbox)
	Background	Color = Red:255, Green:255, Blue:255
	Anchors	Top, Left
	Border	Empty – 0,0,0,0
Label "Delete" button	Size	66,26
	Location	108,48
	HorizontalAlignment	Left
	VerticalAlignment	Center
	ImageMedia	btn_delete.jpg
	RollOverImageMedia	btn_delete_selected.jpg

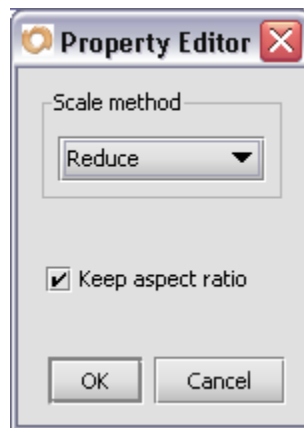
	MediaOptions	Crop
	Transparent	False (checkbox)
	Background	Color = Red:255, Green:255, Blue:255
	Anchors	Top, Left
	Border	Empty – 0,0,0,0
Label "Find" button	Size	60,22
	Location	201,48
	HorizontalAlignment	Left
	VerticalAlignment	Center
	ImageMedia	btn_find.jpg
	RollOverImageMedia	btn_find_selected.jpg
	MediaOptions	Crop
	Transparent	False (checkbox)
	Background	Color = Red:255, Green:255, Blue:255
	Anchors	Top, Left
	Border	Empty – 0,0,0,0
Label Purple "form" button	Size	38,26
	Location	435,51
	HorizontalAlignment	Center
	VerticalAlignment	Center
	ImageMedia	btn_purple_form.jpg
	MediaOptions	Crop
	Transparent	False (checkbox)
	Background	Color = Red:255, Green:255, Blue:255
	Anchors	Top, Right
	Border	Empty – 0,0,0,0
Label Grey "list" button	Size	38,26
	Location	475,51
	HorizontalAlignment	Center
	VerticalAlignment	Center
	ImageMedia	btn_grey_list.jpg
	MediaOptions	Crop
	Transparent	False (checkbox)
	Background	Color = Red:255, Green:255, Blue:255
	Anchors	Top, Right
	Border	Empty – 0,0,0,0
Label Grey "table" button	Size	515,51
	Location	38,26
	HorizontalAlignment	Center
	VerticalAlignment	Center
	ImageMedia	btn_purple_table.jpg
	MediaOptions	Crop
	Transparent	False (checkbox)
	Background	Color = Red:255, Green:255, Blue:255
	Anchors	Top, Right
	Border	Empty – 0,0,0,0
Label "help" button	Size	23,24
	Location	578,49
	HorizontalAlignment	Left
	VerticalAlignment	Center
	ImageMedia	btn_help.jpg

	RollOverImageMedia	btn_help_selected.jpg
	MediaOptions	Crop
	Transparent	False (checkbox)
	Background	Color = Red:255, Green:255, Blue:255
	Anchors	Top, Right
	Border	Empty – 0,0,0,0
Rectangle Purple rectangle below buttons	Size	612,7
	Location	0,76
	Anchors	Top, Left, Right
	Border	Line: size=1, color= Red:204, Green:204, Blue:204
	Foreground	color= Red:153, Green:153, Blue:204
	Background	color= Red:153, Green:153, Blue:204
Rectangle Grey line below purple dividing line	Size	612,1
	Location	0,103
	Anchors	Top, Left, Right
	Border	Line: size=1, color= Red:221, Green:221, Blue:221
	Foreground	color= Red:221, Green:221, Blue:221
	Background	color= Red:221, Green:221, Blue:221
Label Merge text for first and last name	Size	420,20
	Location	13,85
	HorizontalAlignment	Left
	Font	Verdana, 18pt, Bold
	Transparent	True (checkbox)
	Foreground	Color = Red:102, Green:102, Blue:153
	Anchors	Top, Left
	Text	%%first_name%% %%last_name%%
Label Merge text for company name	Size	220,20
	Location	371,88
	HorizontalAlignment	Left
	Font	Verdana, 12pt, Bold
	Transparent	True (checkbox)
	Foreground	Color = Red:153, Green:153, Blue:204
	Anchors	Top, Right
	Text	%%company%%
Label "Image" field label	Size	40,19
	Location	440,126
	HorizontalAlignment	Right
	Font	Verdana, 9pt, Plain
	Transparent	True (checkbox)
	Foreground	Color = Red:204, Green:204, Blue:204
	Anchors	Top, Right
	Text	Image
Field Image	Size	111,109
	Transparent	True (checkbox)
	Location	483,126
	HorizontalAlignment	Center
	Transparent	True (checkbox)
	Foreground	Color = Red:0, Green:0, Blue:0
	Anchors	Top, Left, Right
	DisplayType	ImageMedia
	Border	Line: size=1, color= Red:238, Green:238, Blue:238

Label "First Name" field label	Size	80,19
	Location	20,126
	HorizontalAlignment	Right
	Font	Verdana, 11pt, Plain
	Transparent	True (checkbox)
	Foreground	Color = Red:153, Green:153, Blue:204
	Anchors	Top, Left
	Text	First Name
Field First Name	Size	277,19
	Transparent	True (checkbox)
	Location	107,126
	HorizontalAlignment	Left
	Font	Verdana, 11pt, Plain
	Margin	1,3,1,1
	Foreground	Color = Red:0, Green:0, Blue:0
	Anchors	Top, Left, Right
	DisplayType	TextField
	Border	Line: size=1, color= Red:238, Green:238, Blue:238
<i>For the rest of the fields – just copy/paste the first name field and label and change the location and the label text.</i>		
Label "Last Name" field label	Location	20,144
	Text	Last Name
Field Last Name	Location	107,144
	DataProvider	last_name
Label "Title" field label	Location	20,162
	Text	Last Name
Field Title	Location	107,162
	DataProvider	title
Label "Company" field label	Location	20,180
	Text	Company
Field Company	Location	107,180
	DataProvider	company
Label "Phones" field label	Location	40,198
	Text	Phones
Field Phones	Location	107,198
	DataProvider	company (FOR NOW – will change later!)
Label "Email" field label	Location	40,216
	Text	Email
Field Email	Location	127,216
	DataProvider	email
Label "Notes" field label	Location	20,254
	Text	Notes
Field Notes	Location	107,254
	DataProvider	notes
	Size	467,57
	DisplayType	TextArea
	Scrollbars	Vertical: when needed, Horizontal: never

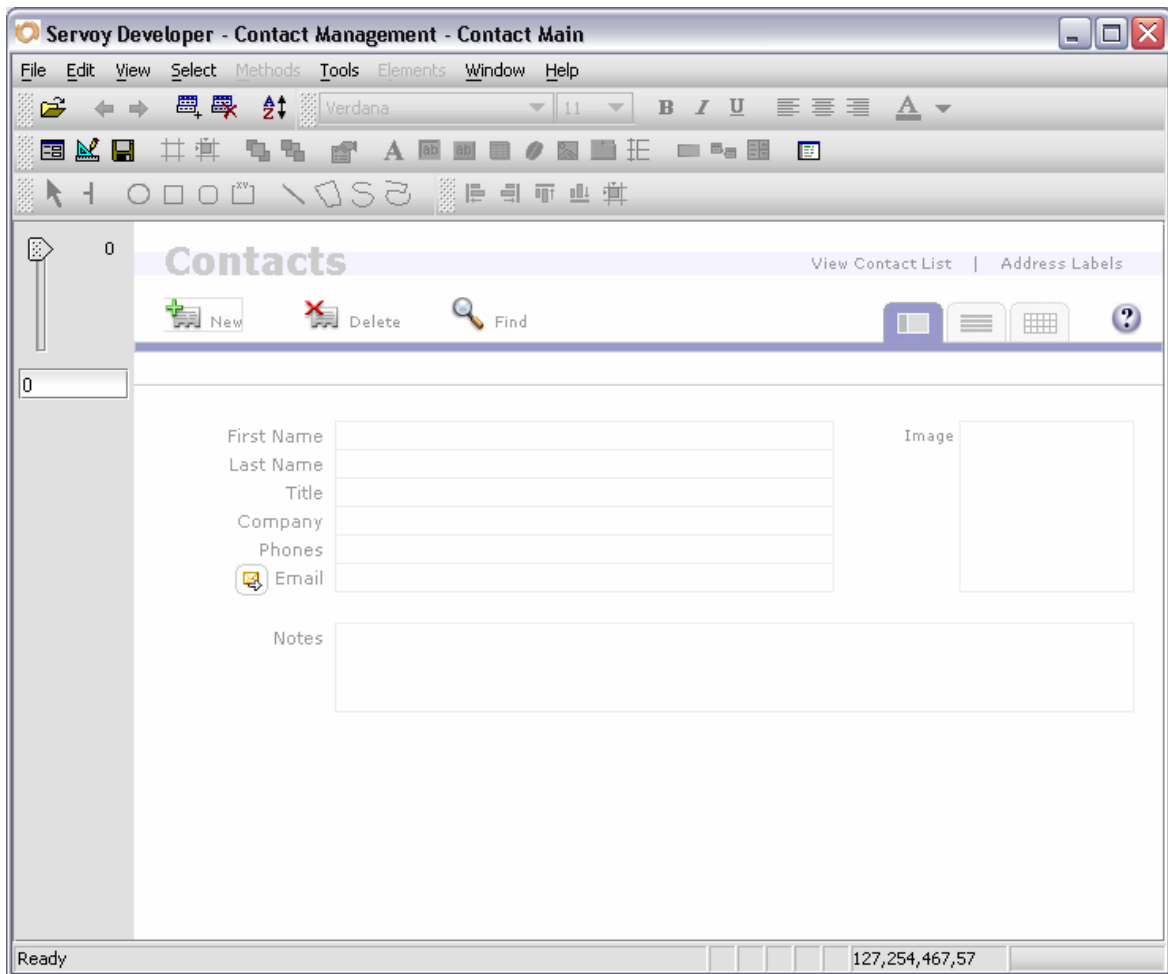
Label "email" button	Size	23,24
	Location	43,215
	HorizontalAlignment	Center
	VerticalAlignment	Center
	ImageMedia	btn_email.jpg
	RollOverImageMedia	btn_email_selected.jpg
	MediaOptions	Crop
	Transparent	False (checkbox)
	Background	Color = Red:255, Green:255, Blue:255
	Anchors	Top, Left
	Border	Empty – 0,0,0,0

NOTE: On each of the graphic buttons - double-click the property mediaOptions and set the Scale method "Reduce" and DO check the checkbox "Keep aspect ratio":



If you chose to place all the elements yourself – GOOD JOB! Take a breather. We'll continue in the next section by creating the forms that will make up the tabpanel at the bottom of the screen to display the Main Address, Second Address, and Related Contacts.

If everything came out correctly – your data entry screen should look like this:

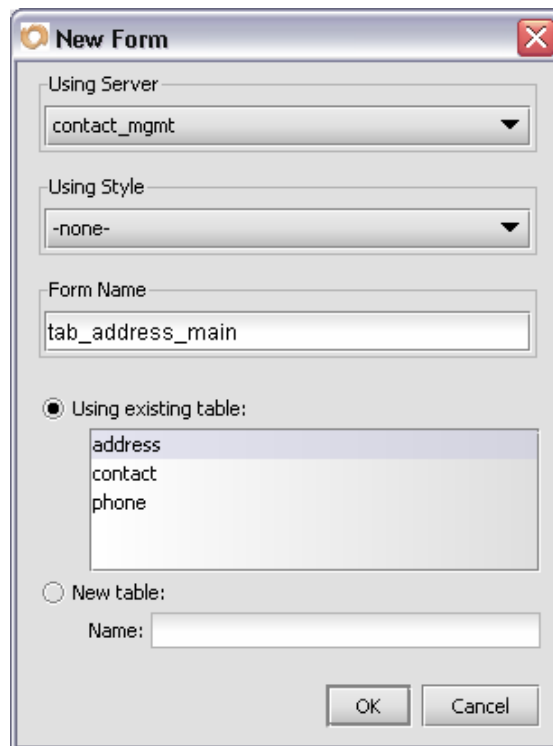


Chapter 6 – Building the User Interface – Part 2

Estimated Time To Complete: 1.5 hours

We're now going to build the forms that will show up in the tabpanel control at the bottom of the main screen. In Servoy, you don't have to duplicate the contacts screen and draw "fake" tabs – Servoy has a *native tabpanel object* that we can use to show different forms – and still have a SINGLE "Contacts" screen.

Let's start by making a new form – select "New Form" from the "File" menu – choose the "contact_mgmt" server, none for the style, and choose the "Address" table.



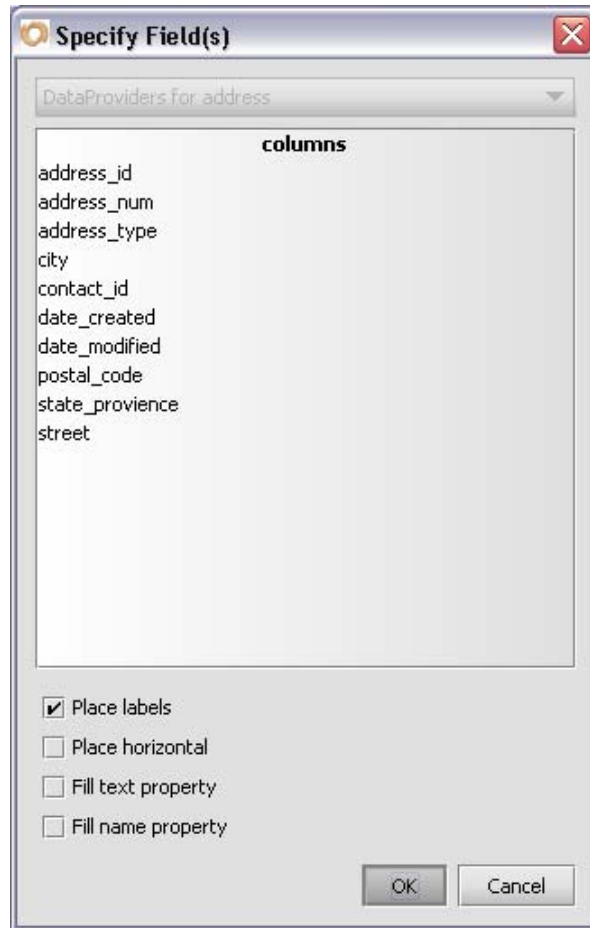
The screenshot shows the "New Form" dialog box with the following settings:

- Using Server:** contact_mgmt
- Using Style:** -none-
- Form Name:** tab_address_main
- Using existing table:** (selected) address
- New table:** (unselected) Name:

Buttons: OK, Cancel

Name the form "tab_address_main" (no quotes) – and click the "OK" button.

When you see the "Specify field(s)" dialog – DO NOT CHOOSE ANY FIELDS and just click the "OK" button.



On the new form you created, set the following properties:

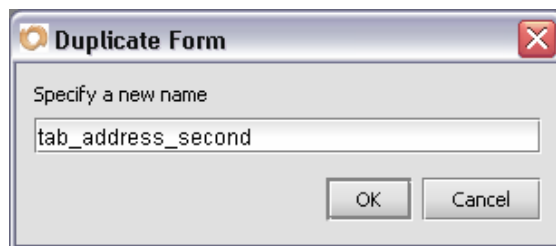
Object Type	Property	Value
Form	Width	612
	ShowInMenu	FALSE (checkbox – UNCHECKED)
	View	Record view (locked)
	Scrollbars	Vertical: never, Horizontal: never
	StyleName	<none>
	Controller	<none>
	TitleText	Address Main
	Name	tab_address_main
Body Part	Height	120
	Background	Color = Red:245, Green:243, Blue:255

Place the Street and City fields (copy from the contact_main screen and change the dataProvider property to the correct fields. Here's the exact settings if you're trying to set them yourself:

Object Type	Property	Value
Label "Street" field label	Size	80,19
	Location	22,22
	Transparent	TRUE (CHECK checkbox)
	HorizontalAlignment	Right
	Font	Verdana, 11pt, Plain
	Transparent	True (checkbox)
	Foreground	Color = Red:153, Green:153, Blue:204
	Anchors	Top, Left
	Text	Street
Field Street	Size	277,19
	Transparent	FALSE (UNCHECK checkbox)
	Location	109,22
	HorizontalAlignment	Left
	Font	Verdana, 11pt, Plain
	Foreground	Color = Red:0, Green:0, Blue:0
	Background	Color = Red:255, Green:255, Blue:255
	Anchors	Top, Left
	DisplayType	TextField
	Name	street
	Margin	1,3,1,1
	DataProvider	street
	Border	Line: size=1, color= Red:238, Green:238, Blue:238
<i>For the rest of the fields – just copy/paste the street field and label and change the location, size and the label text.</i>		
Label "City" field label	Location	40,198
	Size	80,19
	Text	City
Field City	Location	109,40
	Size	277,19
	Name	city
	DataProvider	city
Label "State/Province" field label	Location	2,58
	Size	100,19
	Text	State/Province
Field state_province	Location	109,58
	Size	117,19
	Name	state_province
	DataProvider	state_province
Label "Postal Code" field label	Location	232,58
	Size	70,19
	Text	Postal Code
Field postal_code	Location	309,58
	Size	77,19
	Name	postal_code
	DataProvider	postal_code
Label	Location	407,22

"Address Type" field label	Size	90,19
	Text	Address Type
Field address_type	Location	492,22
	Size	101,19
	Name	address_type
	DataProvider	postal_code
	Anchors	Top, Left, Right
Round Rectangle "Swap" button	Size	187,19
	RoundedRadius	10
	Location	407,45
	Background	Color = Red:255, Green:255, Blue:255
	Foreground	Color = Red:221, Green:221, Blue:221
	Name	btn_swap
Label "Swap" button	Location	411,45
	Size	180,19
	Text	Swap with Second Address
	Font	Verdana, 9pt, Plain
	Name	lbl_swap

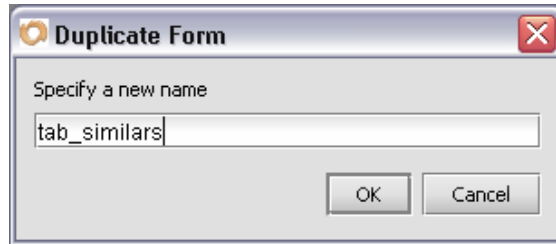
Now we have the "Main Address" tab – and now to make the "Second Address" tab – we're simply going to DUPLICATE this existing form and rename it. Choose "Duplicate Form" from the "File" menu – and call the new form "tab_address_second" (no quotes):



On the new form you created, set the following properties:

Object Type	Property	Value
Form	ShowInMenu	FALSE (checkbox – UNCHECKED)
	View	Record view (locked)
	Scrollbars	Vertical: never, Horizontal: never
	StyleName	<none>
	Controller	<none>
	TitleText	Address Second
	Name	tab_address_second

Now we have the two address tabs – and now we’re going to make the “Similar” tab. Again, we’re simply going to DUPLICATE this existing form and rename it. Choose “Duplicate Form” from the “File” menu – and call the new form “tab_similars” (no quotes):



On the new form you created, set the following properties:

Object Type	Property	Value
Form	ShowInMenu	FALSE (checkbox – UNCHECKED)
	View	Record view (locked)
	Scrollbars	Vertical: never, Horizontal: never
	StyleName	<none>
	Controller	<none>
	TitleText	Similar Tab
	TableName	contact
	Name	tab_similars

You can DELETE all the fields and labels on this form (for now). We’re going to depart from the FileMaker functionality here – and we’re going to create another tabpanel that has another form (in list view) to display all the matching similar contacts.

We could just use a portal here – but then we’d have to do things the FileMaker way – create calculations that trap for each value possible in the radio button list, etc. Since this is Servoy – and we can “tell” list views to find and sort any way we want – this is going to be a LOT easier to implement, much more flexible, and *very easy to code* (as we’ll see in the next section).

Let's create another new form called "list_similars." Choose "New Form" from the "File" menu – with the following settings:

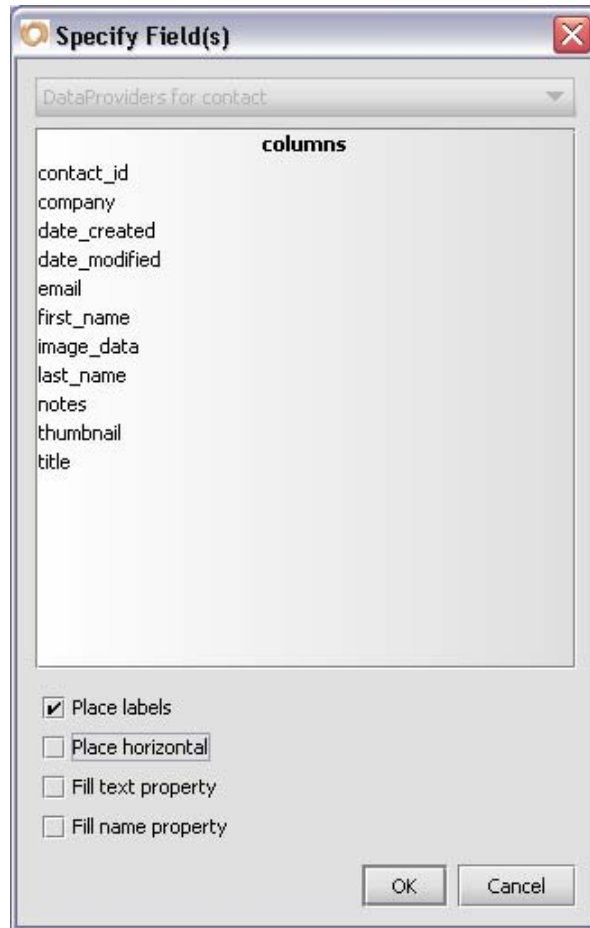
The screenshot shows the 'New Form' dialog box with the following settings:

- Using Server:** contact_mgmt
- Using Style:** -none-
- Form Name:** list_similars
- Using existing table:** (selected)
 - address
 - contact (highlighted)
 - phone
- New table:** (unselected)
 - Name:

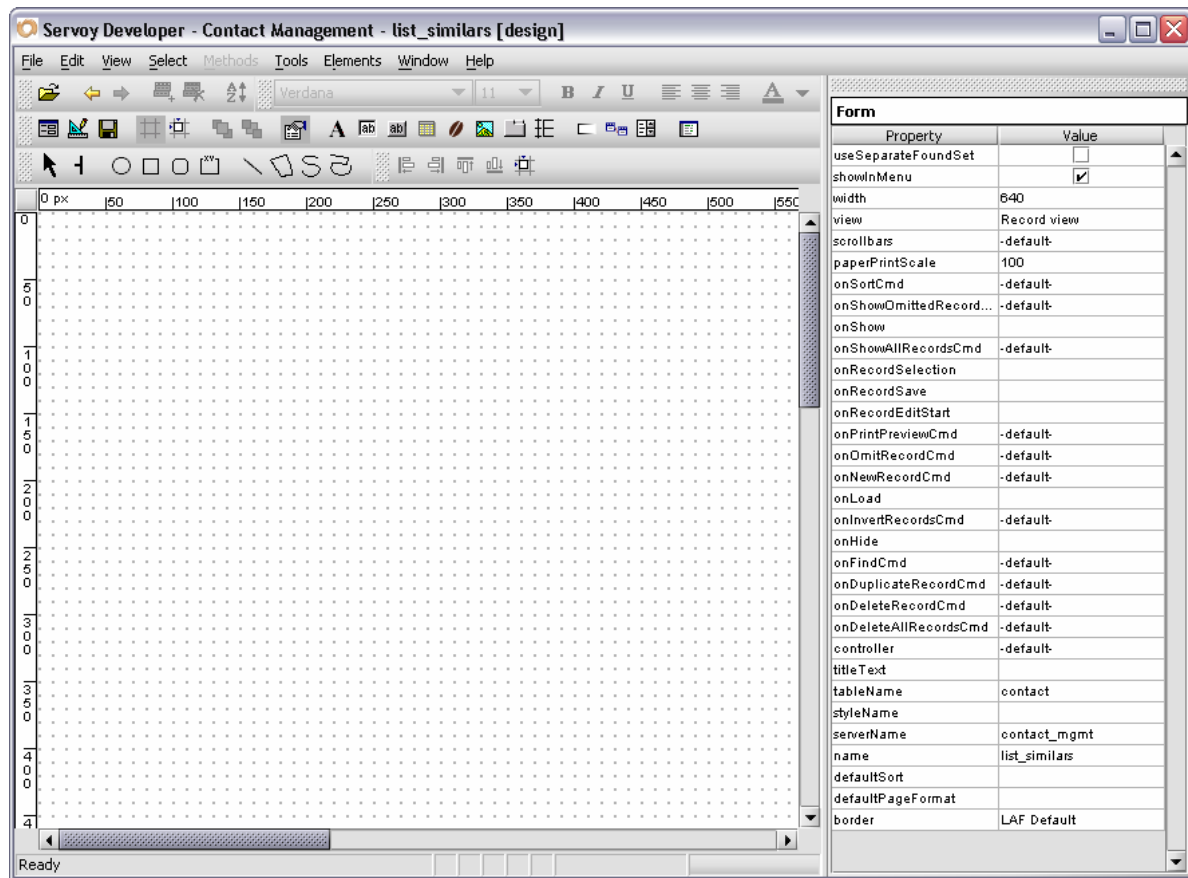
Buttons: OK, Cancel

Click the "OK" button.

Once you've created the form – don't choose any fields – click the "OK" button:



Great! You should see something like this:



On the new form you created, set the following properties:

Object Type	Property	Value
Form	useSeparateFoundset	TRUE (checkbox –CHECKED)
	ShowInMenu	FALSE (checkbox –UNCHECKED)
	Width	612
	View	List view (locked)
	Scrollbars	Vertical: when needed, Horizontal: never
	StyleName	<none>
	Controller	<none>
	TitleText	Similar List
	TableName	contact
Body Part	Name	list_similars
	Height	20
	Background	Color = Red:255, Green:255, Blue:255

Now we're going to add two labels to display the data.

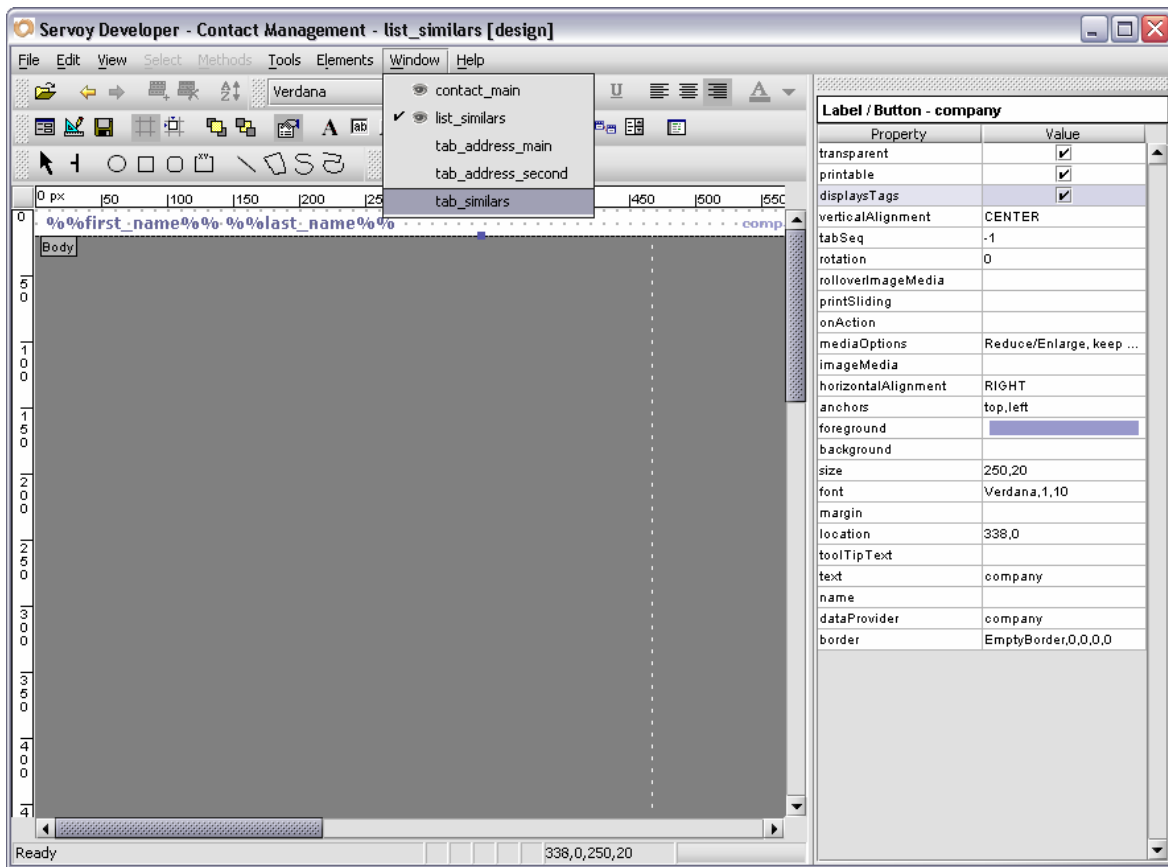
Yep, labels!

In Servoy, you can assign a dataProvider *to any object* – so we're going to use labels. With labels (as opposed to fields) – if the text is too long to fit in the width of the label – it will automatically display an ellipse (...) at the end – without any programming. Because we've set the vertical scrolling to "when needed" – we need to allow for a 20 pixel wide scrollbar – that will appear when necessary. So we should keep the "company" label at least 20 pixels from the right side of the form.

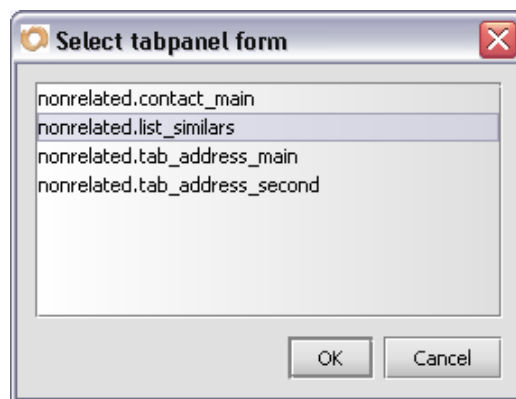
Add a label (or copy/paste from another form) and set the following properties:

Object Type	Property	Value
Label Contact Name	Size	350,20
	Location	8,0
	HorizontalAlignment	Left
	Font	Verdana, 12pt, Bold
	Transparent	True (checkbox CHECKED)
	DisplayTags	True (checkbox CHECKED)
	Foreground	Color = Red:102, Green:102, Blue:153
	Anchors	Top, Left
	Text	%%first_name%% %%last_name%%
Label Contacts Label	Size	250,20
	Location	338,0
	HorizontalAlignment	Right
	Font	Verdana, 10pt, Bold
	Transparent	True (checkbox CHECKED)
	DisplayTags	True (checkbox CHECKED)
	Foreground	Color = Red:153, Green:153, Blue:204
	Anchors	Top, Right
	Text	%%company%%

Now that we have this form created – we can go back to the “tab_similars” form and add the tabpanel to display it.



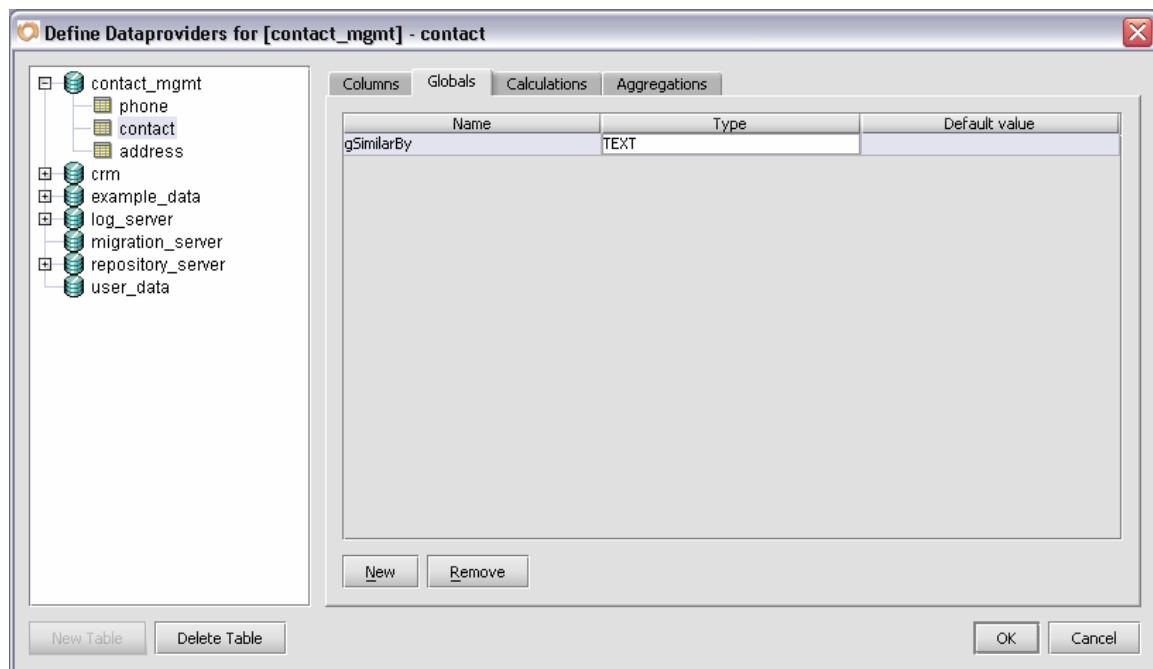
From the “Window” menu – choose “tab_similars”. Click on the tabpanel tool – and when the “” dialog displays – choose the form “nonrelated.list_similars”:



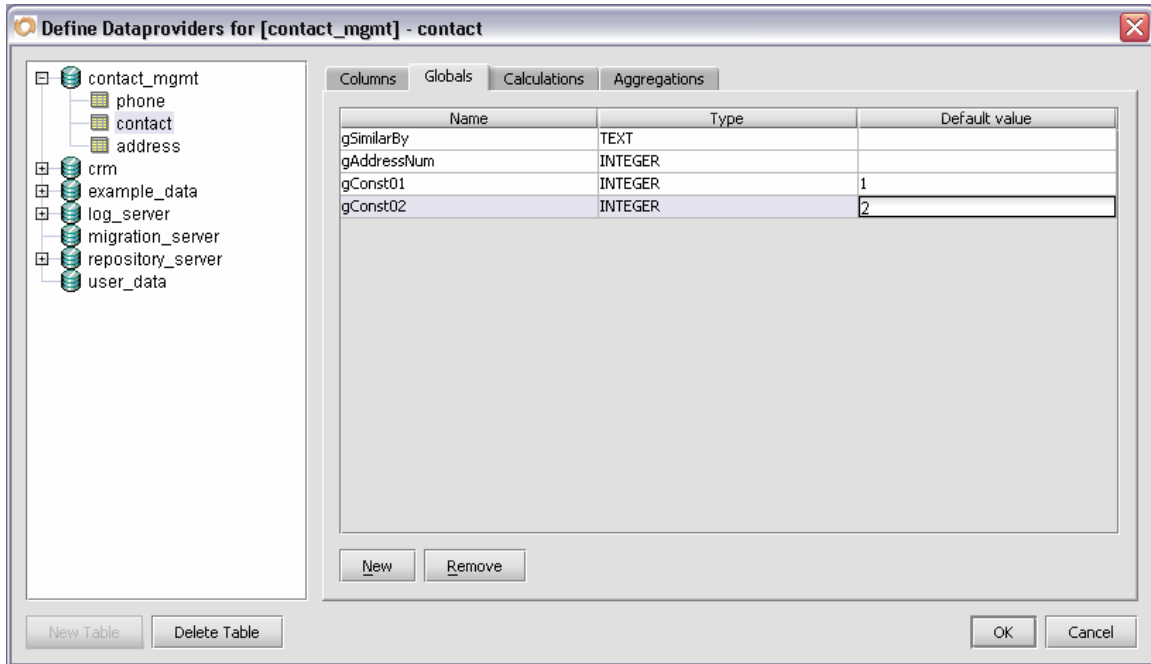
Once the tabpanel is placed – change the properties of the tabpanel to the following:

Object Type	Property	Value
Tabpanel list_similars	Size	612,86
	Location	0,34
	TabOrientation	HIDE
	Font	Verdana, 10pt, Plain
	Transparent	FALSE (checkbox UNCHECKED)
	Background	Color = Red:255, Green:255, Blue:255
	Anchors	Top, Left, Right, Bottom

Now we need to add the label and the field (with valuelist) for the options on the tab. First, choose "Dataproviders..." from the "Tools" menu – and click on the "Globals" tab. Globals in Servoy are IN-MEMORY VARIABLES and NOT physical columns in your database. You can define Globals in the Dataproviders dialog, OR in the Method Editor (we'll do that later). Once you've clicked on the "Globals" tab – click the "New" button and enter "gSimilarBy" (no quotes) and choose the return type of "TEXT" (leave the "Default value" empty):



Now add another global called "gAddressNum" with an "INTEGER" type; one more global called "gConst01" with a type of "INTEGER" and a Default value of "1"; and a "gConst02" also with a type of "INTEGER" and a Default value of "2" (no quotes):



Click "OK" and add a label with the following properties:

Object Type	Property	Value
Label Explanation text	Size	169,19
	Location	197,6
	Font	Verdana, 9pt, Plain
	Transparent	TRUE (checkbox CHECKED)
	Foreground	Color = Red:153, Green:153, Blue:204
	Anchors	Top, Right
	Text	Show Contacts With the Same:

Next, we're going to add the global field we just created – and attach a value list to it. Add a field by choosing "Place field..." from the "Elements" menu – and choose the global field "globals.gSimilarBy".

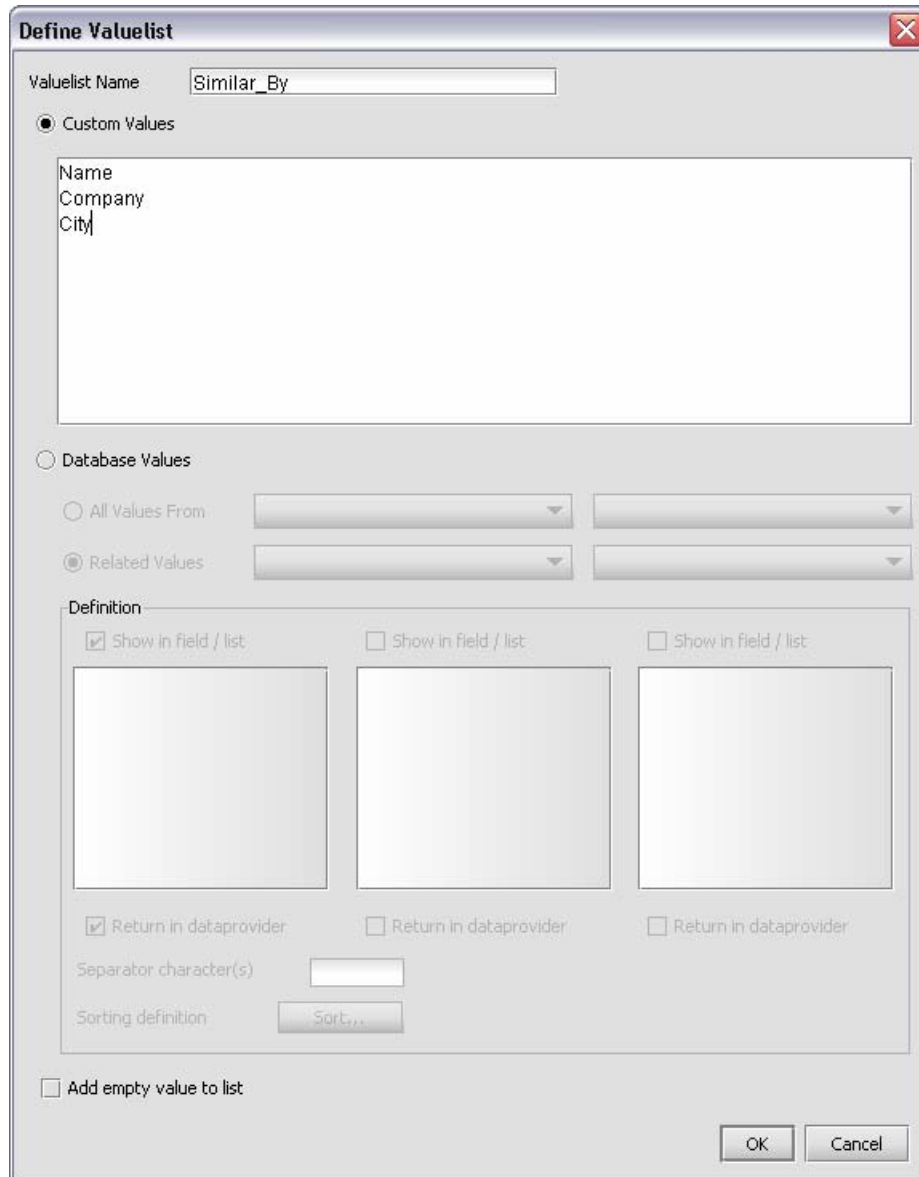


Set the properties as follows:

Object Type	Property	Value
Label Explanation text	Transparent	TRUE (checkbox CHECKED)
	Scrollbars	Vertical: never, Horizontal: never
	DisplayType	CHECKBOXES
	Anchors	Top, Right
	Font	Verdana, 9pt, plain
	Foreground	Color = Red:153, Green:153, Blue:204
	Size	228,20
	Location	371,5
	Border	Empty

Once you've set the properties, it's time to attach the value list. Double-click the blank space next to the "valueList" property – and click the "New" button. Name the value list "Similar_By" and in the "Custom Values" enter:

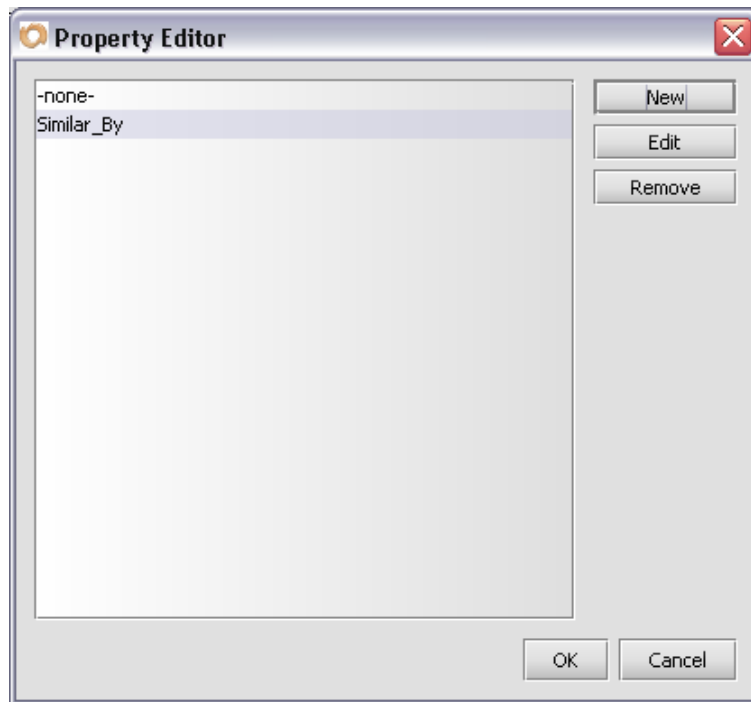
Name
Company
City



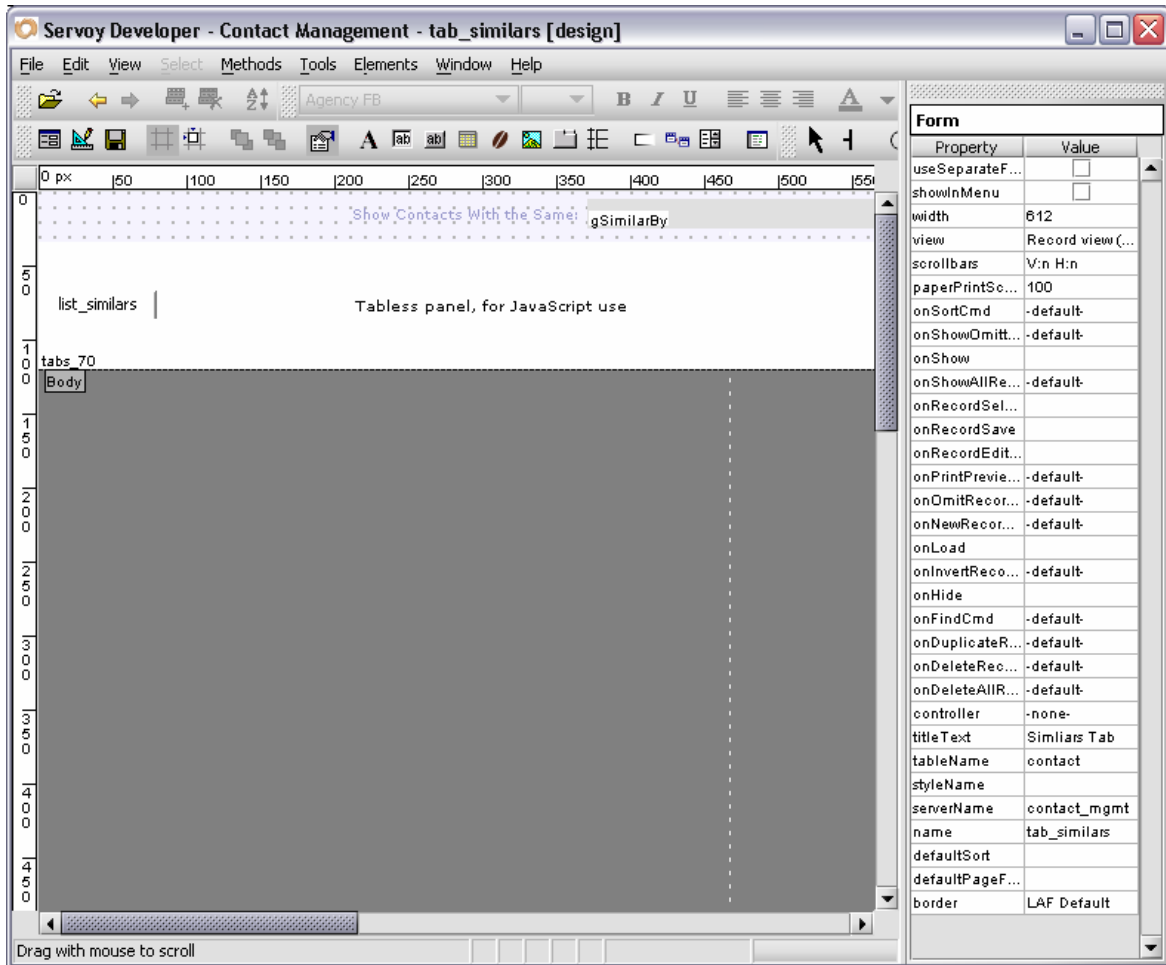
The "Define Valuelist" dialog box is shown. The "Valuelist Name" field contains "Similar_By". The "Custom Values" radio button is selected. The text area below it contains "Name", "Company", and "City". The "Database Values" radio button is unselected. Below it, the "All Values From" and "Related Values" options are shown with dropdown menus. The "Definition" section has three checkboxes: "Show in field / list" (checked), "Show in field / list" (unchecked), and "Show in field / list" (unchecked). Below these are three empty text boxes. Further down, there are three checkboxes: "Return in dataprovider" (checked), "Return in dataprovider" (unchecked), and "Return in dataprovider" (unchecked). Below these are a "Separator character(s)" field and a "Sorting definition" button labeled "Sort...". At the bottom, there is an "Add empty value to list" checkbox and "OK" and "Cancel" buttons.

Then click the "OK" button.

You'll see your new value list show up:

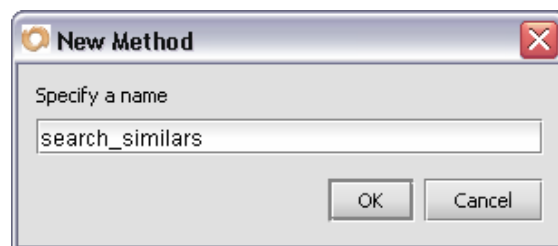


Your form should look like this:



Now that we have the form created – let’s add some code so that our solution will work like the FileMaker solution. We’re going to create a new method so that when the user goes to the “Related Contacts” tab and changes one of the radio button criteria – the search will be performed and the text on the label will show the number of matches.

Open up the Method Editor (CTRL+SHIFT-M or choose it from the “Tools” menu). Click on the tab_similars form and create a new method named search_similars:



We're going to do a search on the `list_similars` form – but we're creating the method on the `tab_similars` form because that's where the control is.

Here's the code:

```
//set up variables with each piece of potential data

var contactId = forms.contact_main.contact_id;
var name = forms.contact_main.last_name;
var company = forms.contact_main.company;
var city = forms.contact_main.gconst01_to_address.city;

//search the contact_main
forms.list_similars.Controller.find();

if(globals.gSimilarBy == 'Name')
{
    forms.list_similars.last_name = '#' + name + '%';
}
else if(globals.gSimilarBy == 'Company')
{
    forms.list_similars.company = '#' + company + '%';
}
else if(globals.gSimilarBy == 'City')
{
    forms.list_similars.gconst01_to_address.city = '#' + city + '%';
}

forms.list_similars.contact_id != contactId;

var numFound = forms.list_similars.controller.search(true, false);

if(numFound > 0)
{
    //a match was found - update the label text

    forms.contact_main.elements.tabs_70.setTabTextAt(3, 'Related Contacts: '
+ numFound + ' by ' + globals.gSimilarBy );
}
else
{
    //none found - reset the label text

    forms.contact_main.elements.tabs_70.setTabTextAt(3, 'Related Contacts' );
}
```

This method (or script) looks at the data in the global `gSimilarBy` and then sets the find criteria (like a Set Field command would in a FileMaker script). The search is performed – and then based on the number of records found – we update the text on the tab.

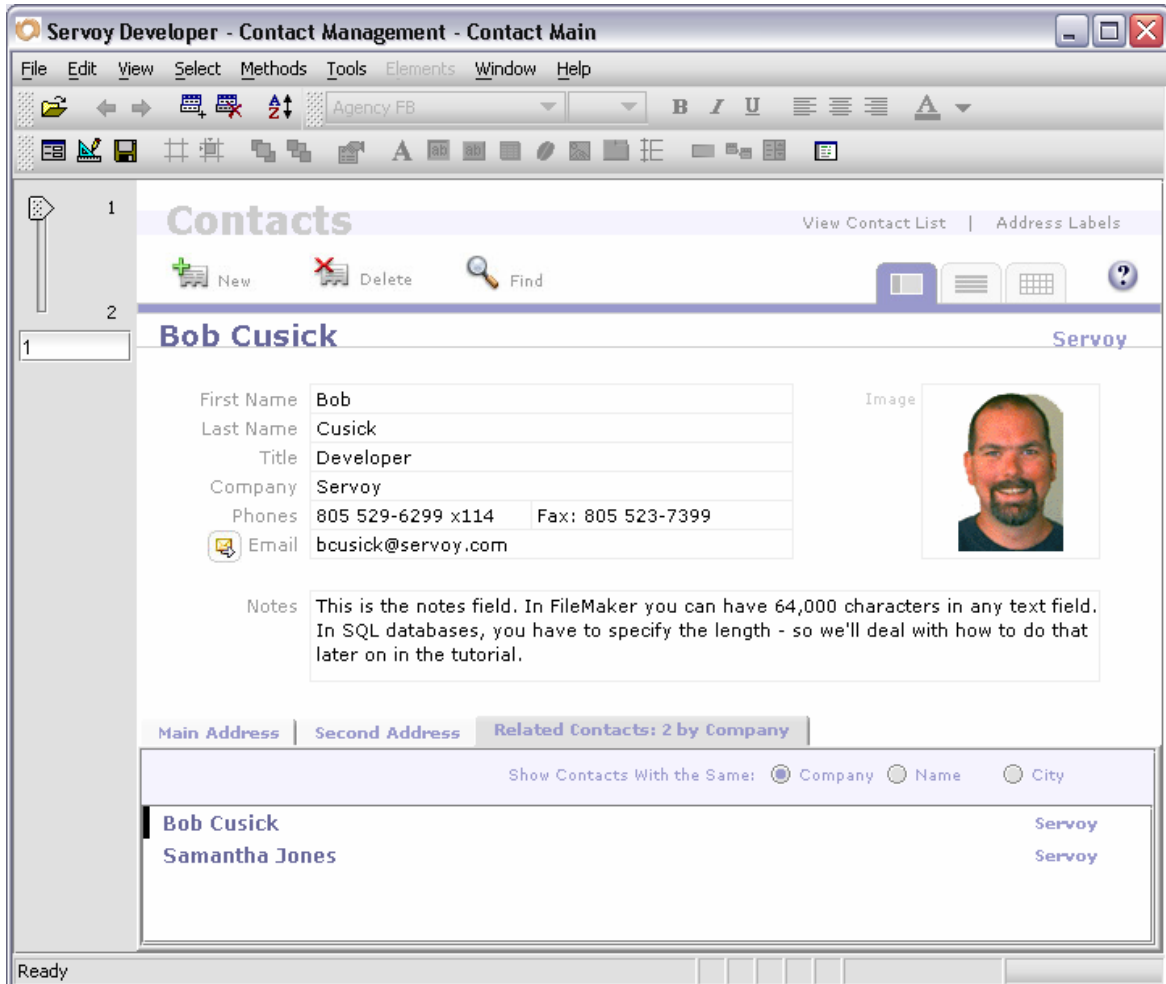
IMPORTANT NOTE: JavaScript IS case-sensitive! Make sure you reference objects and call functions with the **exact**, case-sensitive spellings!

FYI: The `"#"` used above makes the search case IN-sensitive and the `"%"` before and after the string is a wildcard character that indicates a "contains" type of search. If you wanted to do a case-sensitive search, omit the `"#"`; and if you wanted to do a "begins with" search (the default in FileMaker) then remove the first `"%"`.

Next we're going to hook up the script we just created to the radio button list. Navigate to the radio button control. Select the "tab_similars" form from the "Window" menu – and click on the `gSimilarBy` field. Then double-click on the `onDataChange` property and choose the method `search_similars`.



Now you can go into Data mode and add a record of a person with the same last name or from the same company. Then click on the "Related Contacts" tab, and click on the "Name" or "Company" option – and you should see the results appear – and the text on the tab will change as well.



A little later on, we'll add a few more methods so that this tab will update automatically when the person's last name is entered/changed or the company is entered/changed.

We're done creating forms for the moment. Next up: creating relations and adding the forms we created to tabpanels to display the data.

Chapter 7 – Building the User Interface – Part 3

Estimated Time To Complete: 30-45 minutes

Now that we have all the forms we'll need to display in the tabpanels we've previously created, we'll create some relations (like relationships in FileMaker) between the files – and then place the forms into tabpanel objects in the various forms.

Since we want the "Main Address" and "Second Address" to show the address for the currently selected contact – we'll create two different, but similar relations. Click the Relations tool in the toolbar – or choose "Relations..." from the "Tools" menu and click the "New" button in the lower left of the Relations dialog.

By manipulating the source and destination named server connections - Servoy allows you to create relations between two different databases – even two different databases from two different vendors residing on two different servers in two different parts of the world!

For our solution – we are going to be using the same source and destination servers – our `contact_mgmt` server. We're going to relate the `contact_id` and the global field `globals.gConst01` from the `contact` table to the `address` table – so we can display the main address. Set up your "Define Relation" dialog like this:

The "Define Relation" dialog box is shown with the following configuration:

- Define source and destination server:** Both dropdowns are set to `contact_mgmt`.
- Define source and destination table:** The source table is `contact` and the destination table is `address`.
- Relationship name:** `gConst01_to_address`
- Primary key (1) / op / Foreign key (N):**

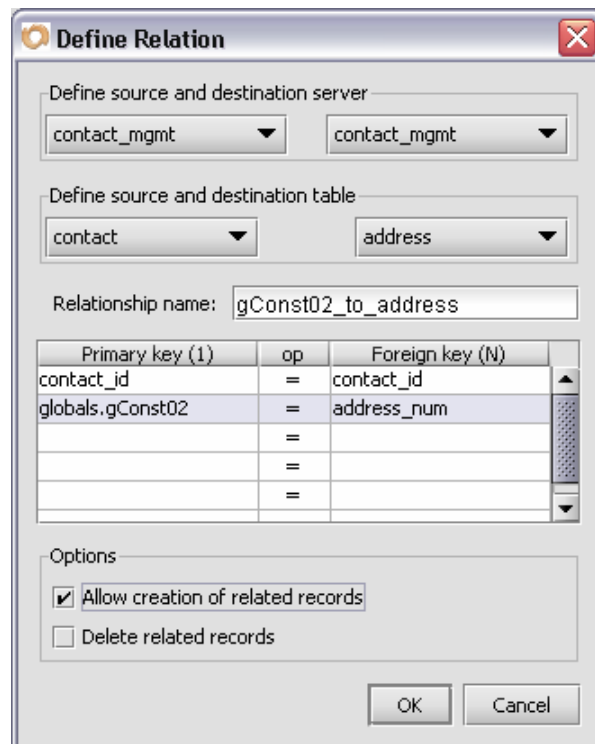
Primary key (1)	op	Foreign key (N)
<code>contact_id</code>	<code>=</code>	<code>contact_id</code>
<code>globals.gConst01</code>	<code>=</code>	<code>address_num</code>
	<code>=</code>	
	<code>=</code>	
- Options:**
 - ☒ Allow creation of related records
 - ☐ Delete related records
- Buttons:** OK and Cancel

Now we're going to add another relation that's very similar – we're going to call it "gConst02_to_address" and rather than specifying globals.gConst01 on the left hand side – we're going to use "globals.gConst02".

Notice that we're setting up multiple *predicates* (match fields) – WITHOUT having to create concatenated calculations (like we would have to do in FileMaker).

Also, the matching doesn't have to be an *equijoin* (equals); it can be >, <, >=, <=, != as well. This capability adds tremendous functionality and power to your relations – and all without coding SQL statements!

Setup the new relation like this:



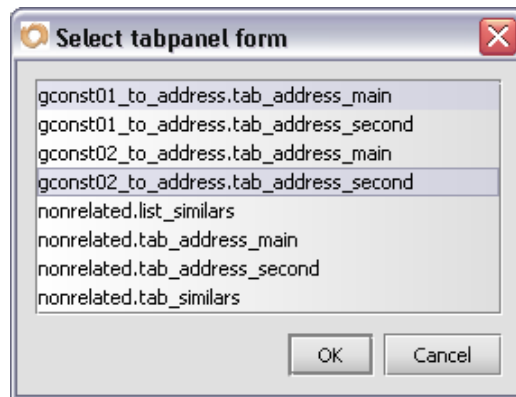
The "Define Relation" dialog box is shown. It has a title bar with a gear icon and a close button. The dialog is divided into several sections:

- Define source and destination server:** Two dropdown menus, both set to "contact_mgmt".
- Define source and destination table:** Two dropdown menus, set to "contact" and "address".
- Relationship name:** A text field containing "gConst02_to_address".
- Table structure:** A table with three columns: "Primary key (1)", "op", and "Foreign key (N)".

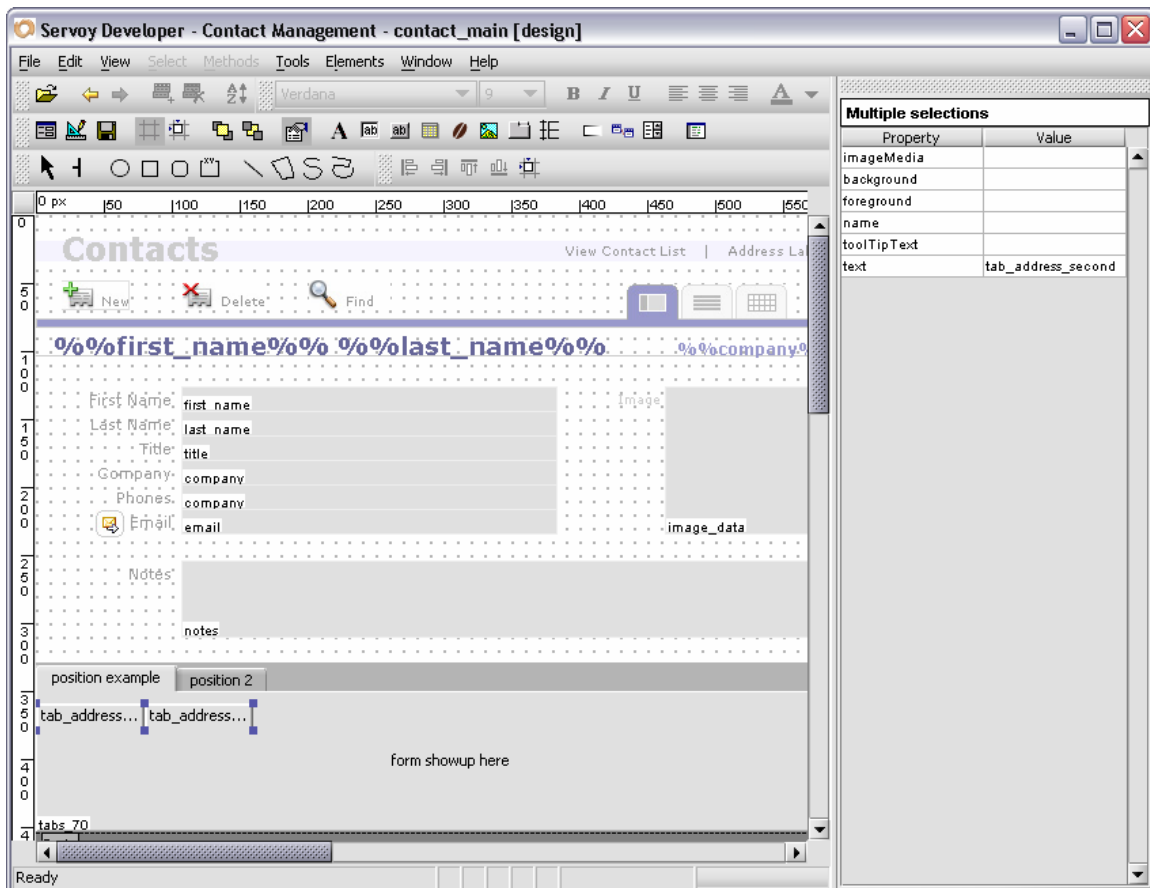
Primary key (1)	op	Foreign key (N)
contact_id	=	contact_id
globals.gConst02	=	address_num
	=	
	=	
	=	
- Options:** Two checkboxes: "Allow creation of related records" (checked) and "Delete related records" (unchecked).
- Buttons:** "OK" and "Cancel" buttons at the bottom right.

OK – we're ready to add our forms (in the Designer mode) to a tabpanel on the contact_main screen. Go to the "Window" menu and choose the "contact_main" form. Create a new tabpanel by clicking on the Tabpanel tool from the toolbar or choosing "Place Tabpanel" from the "Elements" menu.

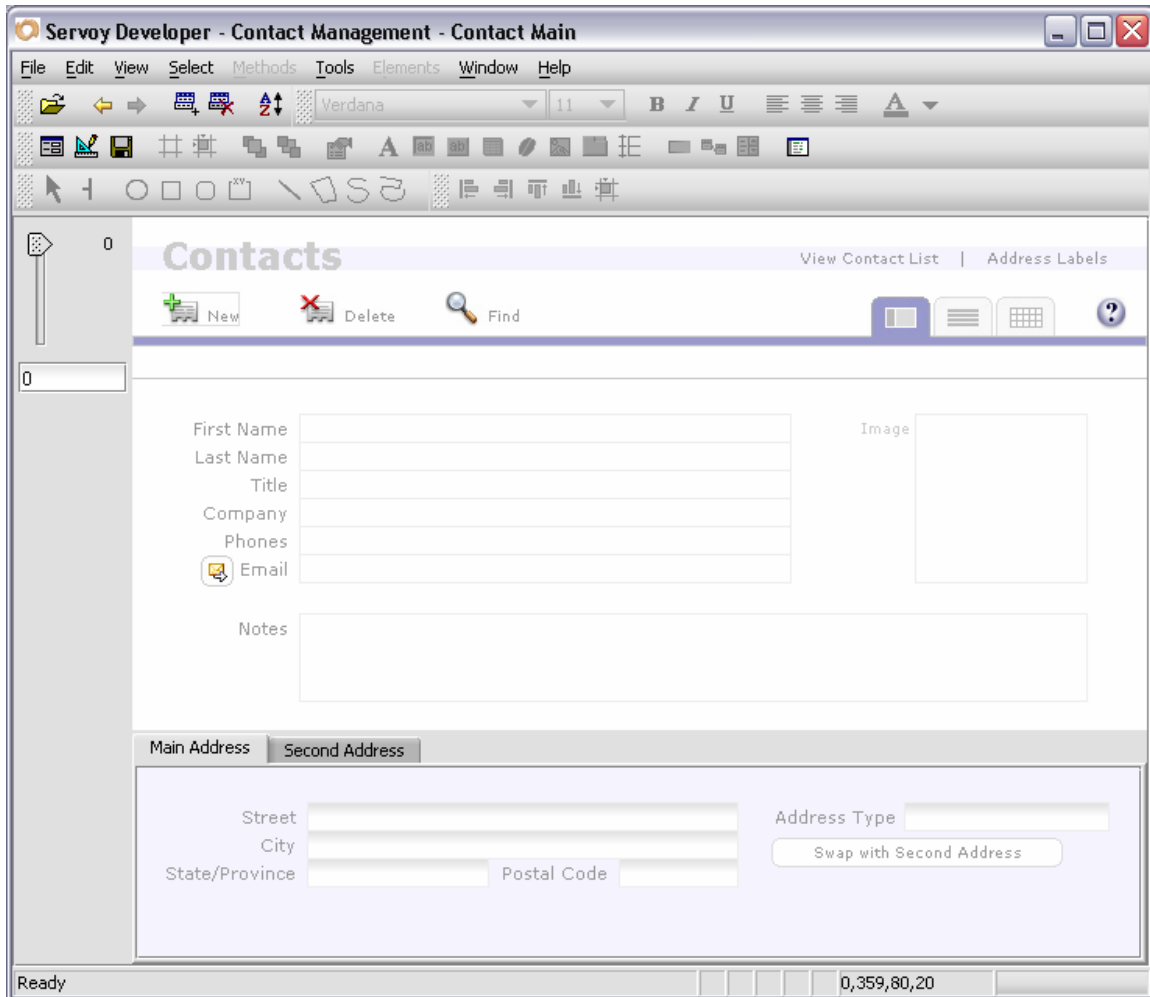
Once the "Select tabpanel form" dialog appears – choose the form "gConst01_to_address.address_main" form and the "gConst02_to_address.address_second" form (hold down the CTRL key to select discontinuous multiple forms):



Notice that each form is listed a couple of times. Why is that? That's because there are two valid relations for our base table (contact) – and there are forms based on those relations (the form name is after the "."). Once you've selected your forms – you'll see two little tabs appear inside the tabpanel.



Click on the individual tabs to change the type that is displayed. Click on the first tab – and change the text property to “Main Address”. Click on the second tab and change the text property to “Second Address”. Exit the Designer mode to see the results of your work:

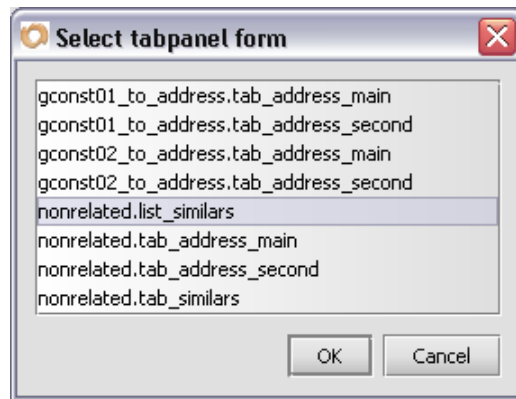


You now have a working solution with a tab for “Main Address” and another tab for “Second Address” – WITHOUT having to create “fake” tabs.

To clean up the tabpanel a bit – set the following properties:

Object Type	Property	Value
Tabpanel Shows addresses	Transparent	TRUE (checkbox CHECKED)
	TabOrientation	TOP
	Anchors	Top, Right, Bottom, Left
	Font	Verdana, 10pt, bold
	Size	612,125
	Location	0,329

Once you've sized the tabpanels – I'll show you how to add our "Similar" form to this existing tabpanel. First, **CLICK ON THE TABPANEL** to "activate" it. Then click on the tabpanel tool, and select the form called "nonrelated.tab_slimilars":



You should have three tabs within the tabpanel. In Servoy, these tabs don't have to be anyplace special (like within the borders of the tabpanel object itself, etc.).

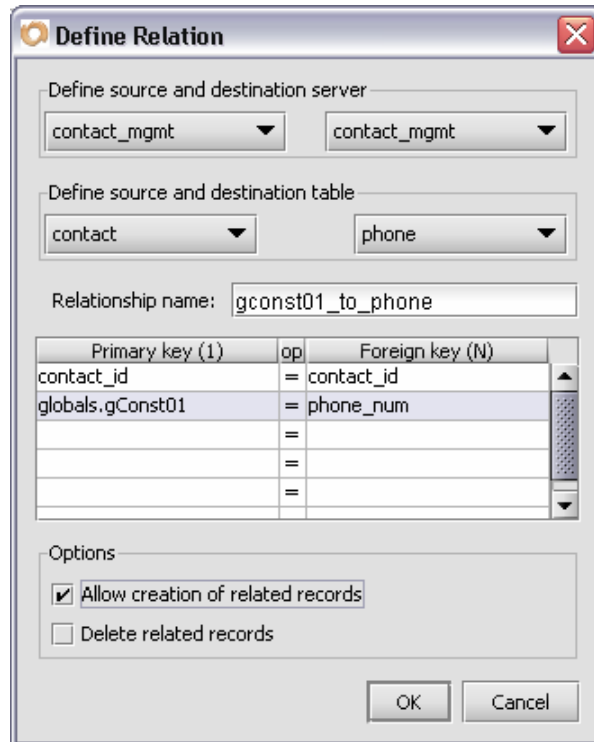
Select all three tabs and change the following properties:

Object Type	Property	Value
Tab Objects Controls the look and feel	Background	Color = Red: 255, Blue: 255, Green: 255
	Foreground	Color = Red: 153 Blue: 153, Green: 204

Once you've done that – you can rename the tab you just placed (nonrelated.tab_similars) – to "Similar" (don't worry – we'll change the text at runtime to display what's being viewed – just like in the FileMaker solution).

We now have to setup our phone numbers to display correctly – and automatically add the related records in our "phone" table.

To do this, we'll need to set up two more relations:



Define Relation

Define source and destination server
contact_mgmt contact_mgmt

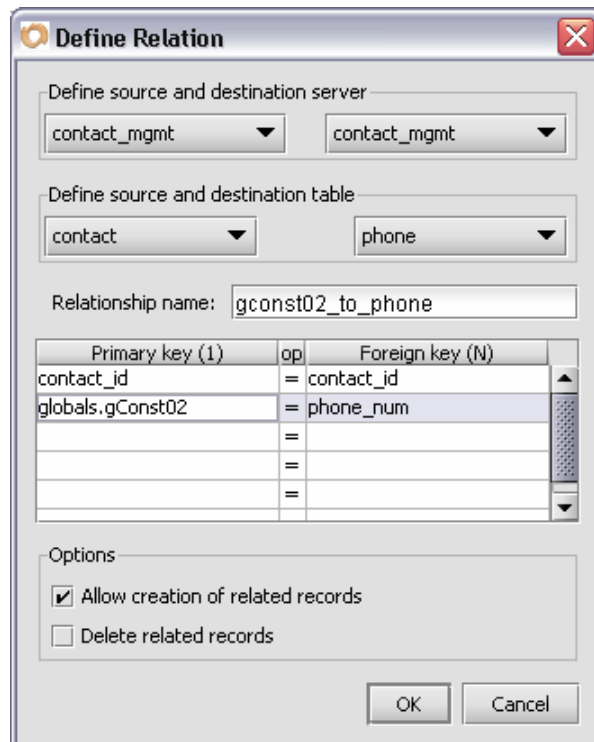
Define source and destination table
contact phone

Relationship name: gconst01_to_phone

Primary key (1)	op	Foreign key (N)
contact_id	=	contact_id
globals.gConst01	=	phone_num
	=	
	=	
	=	

Options
☒ Allow creation of related records
☐ Delete related records

OK Cancel



Define Relation

Define source and destination server
contact_mgmt contact_mgmt

Define source and destination table
contact phone

Relationship name: gconst02_to_phone

Primary key (1)	op	Foreign key (N)
contact_id	=	contact_id
globals.gConst02	=	phone_num
	=	
	=	
	=	

Options
☒ Allow creation of related records
☐ Delete related records

OK Cancel

Earlier, we just placed the “company” field where the phone numbers would go. So, let’s go ahead and change that.

Click on the “company” field (next to the “Phones” label) – and change the properties of the field:

Object Type	Property	Value
Phone Field	Size	138,19
	Location	107,198
	Anchors	Top, Left
	DataProvider	gconst01_to_phone.phone_data

To change the dataprovider, double-click on the box next to the “DataProvider ” property and you’ll see the Property Editor dialog. Choose “gconst01_to_phone” from the pop-up list of relations at the top, then choose the “phone_data” field:



Click OK – and that’s it. Now, copy and paste that phone field – and set the following properties:

Object Type	Property	Value
Phone Field	Size	140,19
	Location	244,198
	Anchors	Top, Left, Right
	DataProvider	gconst02_to_phone.phone_data

To change the dataprovider, double-click on the box next to the "DataProvider " property and you'll see the Property Editor dialog. Choose "gconst02_to_phone" from the pop-up list of relations at the top, then choose the "phone_data" field:



Now, our phone records will be created for us automatically when we enter a value for either or both phone numbers!

Once we have the basics in place, let's start to fill in some of the missing pieces.

For example, on the address forms – there is a field for "Address Type". Let's create a value list for that field. From the "Window" menu – choose the form "tab_address_main". Go into Designer mode and click on the field next to "Address Type".

From the Properties panel – double-click the field next to "valuelist" and click the "New" button.

Name the value list "Address_Types" and fill out the dialog like this:

Define Valuelist

Valuelist Name:

☒ Custom Values

Home
Business
Home Office
Vacation

☐ Database Values

☒ All Values From:

☐ Related Values:

Definition

☒ Show in field / list ☐ Show in field / list ☐ Show in field / list

☒ Return in dataprovider ☐ Return in dataprovider ☐ Return in dataprovider

Separator character(s):

Sorting definition:

☒ Add empty value to list

Then change the "displayType" property to "COMBOBOX" – DO check the "editable" property and set the border to "Empty." When you have a field formatted as a combobox – and you DO have the "editable" property checked – the user can choose one of the values in the value list or type their own. If you have the "editable" property set to false (UNCHECKED) – then the user is limited to using ONLY values from the value list and they cannot type their own value (like the pop-up menu in FileMaker).

Use the "Window" menu to navigate to the "tab_address_second" form and click on the field "address_type". Set the valuelist property to the "Address_Types" value list and change the "displayType" to "COMBOBOX". Also change the border to "Empty."

In the next chapter, we'll start doing some scripting on the solution – and explore the various events that Servoy supports to further enhance our solution.

Chapter 8 – Building the User Interface – Part 4

Estimated Time To Complete: 1 hour

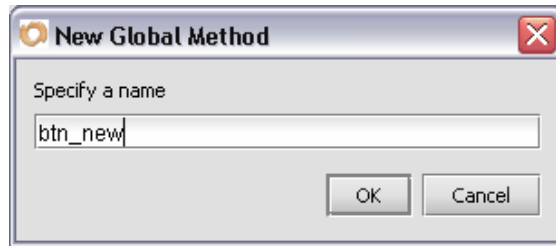
In this section, we'll add some scripting to the buttons and use some of Servoy's cool events to do some neat user interface stuff.

The first thing we're going to do is hook up the "New", "Delete" and "Find" buttons at the top of the screen. So if you don't have the solution open – go ahead and open it up and go into Designer mode.

Click on the "New" button and double-click the white space next to the "onAction" property. You'll see a listing of methods (scripts) – which should be empty (since we haven't created any yet).



We're going to make a new "global" method – because we are going to have the same buttons on other forms – and this way we can make a *single method that we can re-use on multiple forms*. Click the "New global method" button and name the method "btn_new":



You should see that a new global method was created:



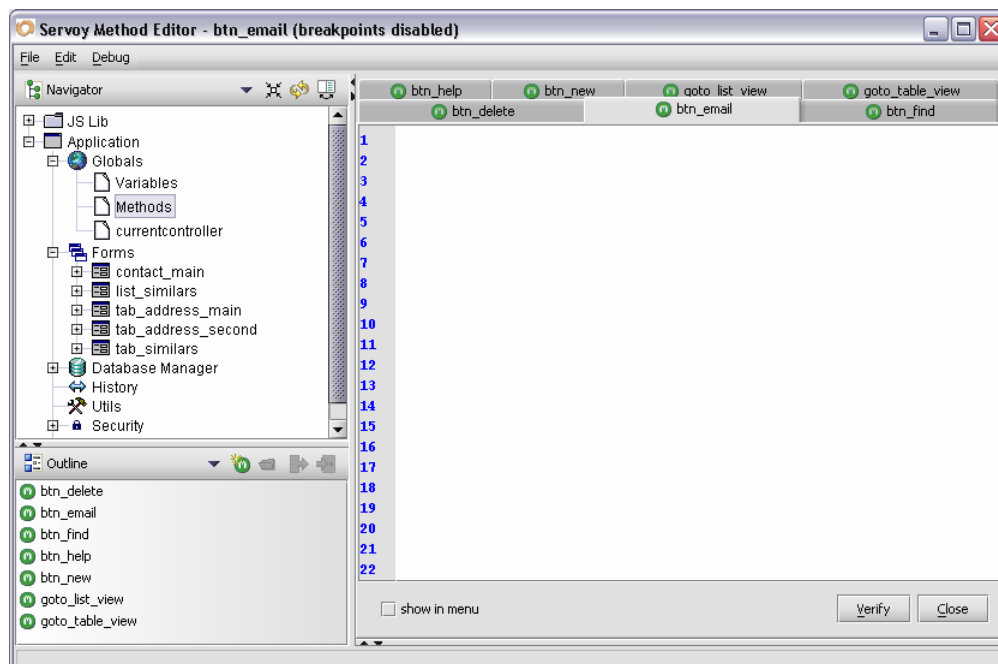
Servoy has created a new, empty script for you – and linked the script to the button. Don't worry about actual code for the method. We'll add the code in a few minutes. Click the OK button to dismiss the dialog.

Repeat the process for: the "Delete" button (call the global method "btn_delete"); "Find" button (call the global method "btn_find"); the Form button (the purple "form" tab at the top – call the global method "goto_form_view"); the List button (the grey one next to the purple "form" tab at the top – call the global method "goto_list_view"); the Table button (the grey one next to the grey "list" tab at the top – call the global method "goto_table_view"); the help button (call the global method "btn_help"). Then there's one last button that we need to define a new method for – and that's the "Email" button next to the "Email" label. Repeat the above sequence and call that global method "btn_email".

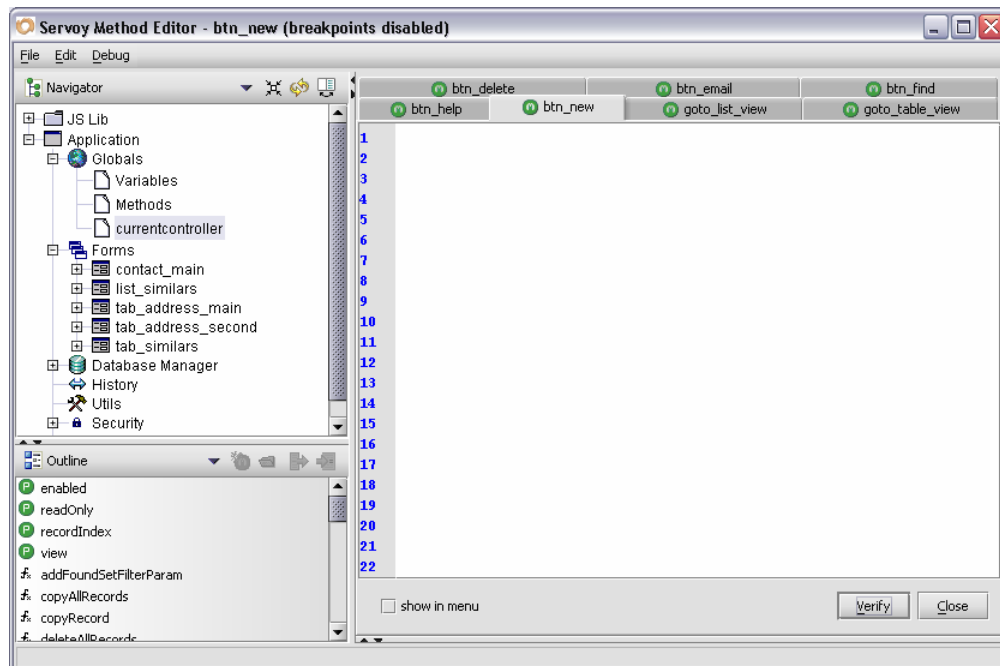


OK, now we'll define what each of those scripts will do. From the "Tools" menu – choose "Methods..." (CTRL-M) and you'll see that all of the global methods we've created are opened up and ready for us to enter the code.

You can see a list of all the global methods by clicking the "+" next to the "Globals" tree on the far left hand side of the dialog – and then clicking on "Methods."

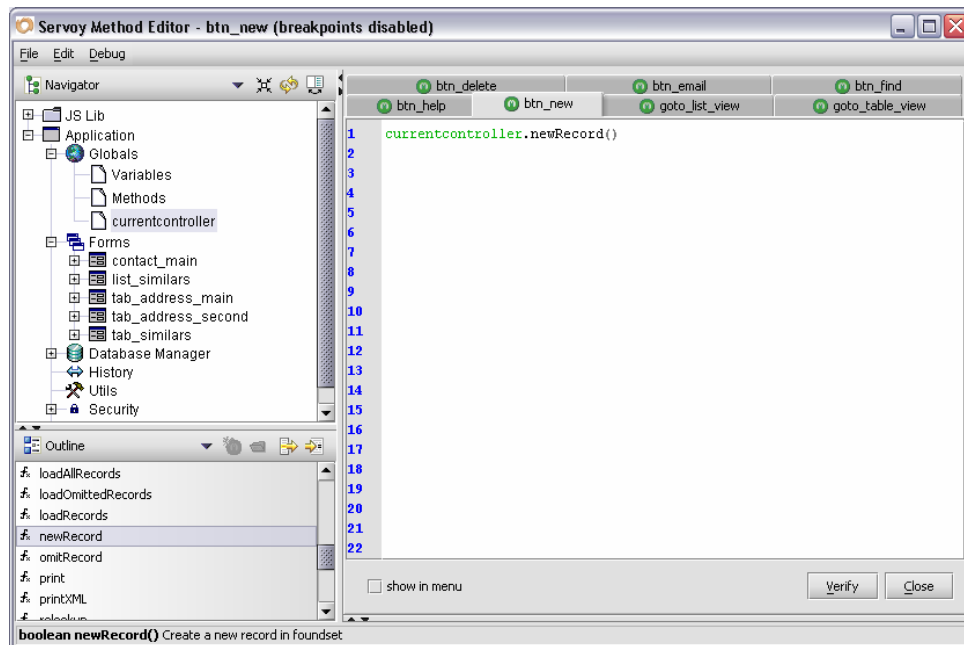


Click on the Script editor tab "btn_new" and then click on the "currentcontroller" node (below "Methods" in the "Globals" section). You'll see a list of functions available where the list of global methods used to be:



Scroll until you can see "newRecord" in the List section – and double-click it. You should see: `currentcontroller.newRecord()` appear in your script. Notice that there is now a little asterisk (*) next to your method name in the Script editor tab. This means you've made changes to the script – and those changes have not yet been compiled and saved.

Click the “Verify” button in the lower left side of the screen – this will compile and save your method – and will also remove the asterisk after the name.



While we’re on the subject of controllers and currentcontroller – let me explain what’s going on.

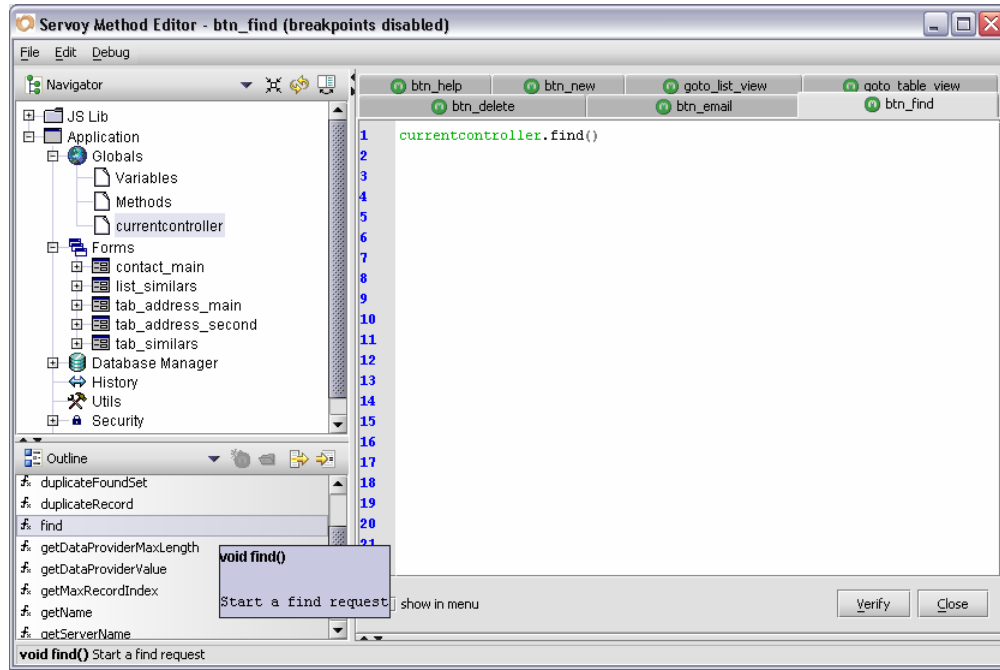
The *controller* object of a form is the object that “knows” how to create records, delete records, duplicate records, find, print, sort, etc. But a controller is tied to a specific form. Each form has a controller object.

The “*currentcontroller*” is a global object that will perform an action regardless of which form you’re on at the time. So, if you’re on the `contact_main` form – then it will make a new record in the contact table, if you’re on a form based on the address table – it will create a new record in the address table and so on.

Since we are using a global method called “`btn_new`” – it also makes sense to use the `currentcontroller` – so we don’t have to re-create the same method on a form-by-form basis.

Next, click on the Script editor tab for "btn_find" and we'll create the code for that method. From the currentcontroller list of functions – double-click on the "find" function.

You should see `currentcontroller.find()` appear in your method.



There are two parts to executing a find – just like in FileMaker. In Servoy – you first issue the `controller.find()` function (or `currentcontroller.find()`) to set up the fields you want to search by (like using "Set Field" in FileMaker); then you perform the find by using `controller.search()` (or `currentcontroller.search()`).

Our button will not actually perform the find – but it will just put the solution into "Find Mode" – waiting for the user to enter their criteria and hit the enter key (or F3) to actually perform the search.

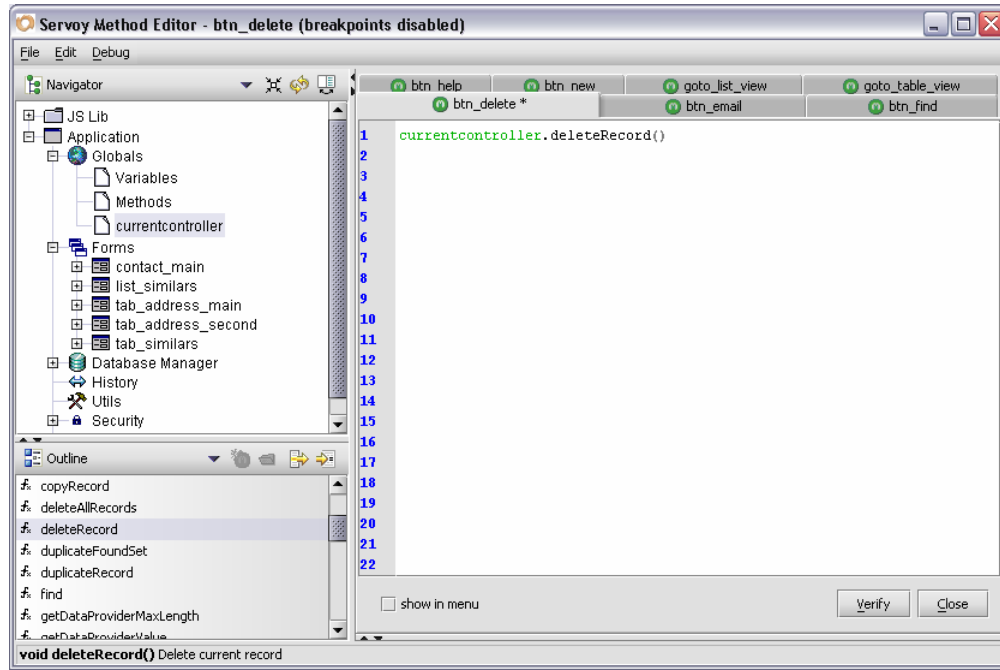
The `search()` function can also take optional *arguments* where you can specify to clear the last results (foundset) or to reduce the foundset (do a search within the current foundset).

To extend the found set (add records to the current foundset) – you would show `controller.search(false, false)`. How do I know this? Simply click on the search function and you'll see the syntax in the lower left side of the Method Editor window.

Here's a really cool feature of Servoy method functions – if you right-click (or click the "move sample" button at the top of the screen) – you can move a fully commented and working sample of the code into your method. This is really a handy feature – and one I use all the time.

Now let's enter the code for the "Delete" button.

Click on the "btn_delete" tab – and double click `deleteRecord` from the `currentcontroller` item of the Globals tree:

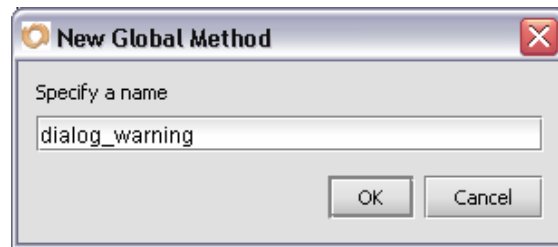


This function will delete the current (master) record of whatever the base table is of the form that's currently displayed. For example – if you're looking at the `contact_main` screen – then it will delete the contact record you're looking at. If you were looking at a form based on the address table, then the "deleteRecord" function would delete an address record.

IMPORTANT NOTE: The `deleteRecord()` and `deleteAllRecords()` functions in Servoy DO NOT prompt the user with a dialog asking "Are you sure?" before the delete – each function simply deletes the record. So, to prevent the user from accidentally deleting a record – we're going to modify our method to display a dialog confirming that they want to delete the record.

I'm going to use this opportunity to show you how to pass *parameters* into a method and receive a result back. We're going to create a global method called "dialog_warning" that will accept the message to display as the argument, and then we're going to use that new method in our "btn_delete" method to show the dialog and check which button they clicked – "OK" or "Cancel".

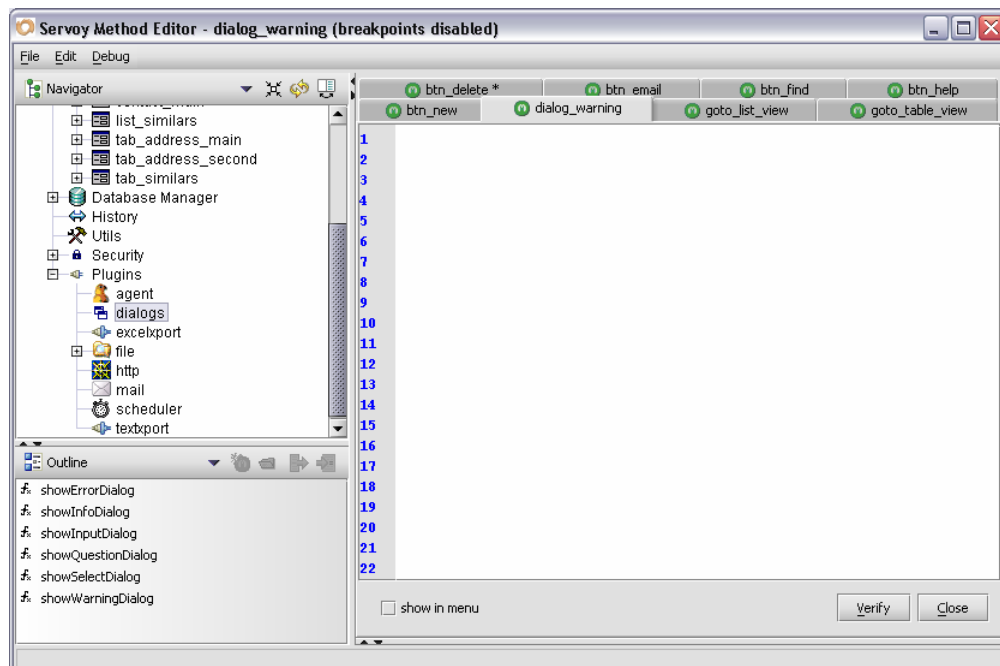
Let's start by creating our new global method "dialog_warning". Click on the "Methods" node of the Globals tree, and click the green circle button with the "m" in it (or right-click on the "Methods" node and choose "Create Global Method"). Name the method "dialog_warning" (no quotes):



The purpose of this method is to have a global method that will show our warning dialogs – so we can have a *single place to update our code* if we need to – rather than having to re-create the entire dialog sequence every time we need it. To make the global method useful – we need a way to pass in a different message (depending on our situation) – so we are going to make use of parameters when calling the method – and then we're going to use those passed parameters in our code.

Let's begin with the basic method of showing a dialog. We'll use the "dialogs" plug-in to accomplish this. Yes, you can create your own form and show it as a dialog – but I want to show you how to do it the "easy" way. Plus, there are several types of dialogs you can display – each with the OS-specific icon (Warning, Info, Question and Error).

Click on the "Plugins" tree and then click on the "dialogs" plug-in. You'll see a list of the available functions for that plug-in:



We're going to use the "showWarningDialog" function. If you were to just double-click it – you would see the following in your code:

```
plugins.dialogs.showWarningDialog( dialog_title, msg, button1,
[button2], [buttonN])
```

Although this is pretty straight forward – and you can figure out what you're supposed to put in each of the parameters – I want you to click on "showWarningDialog" and click the "Move Sample" button (or right-click on it). When you do that – you'll see this code:

```
//show dialog
var thePressedButton = plugins.dialogs.showWarningDialog('Title',
'Value not allowed', 'OK');
```

There are a couple of things going on here that I want to point out. First of all, the "//show dialog" is a **comment**.

You can use the C and C++ style of commenting in all your scripts. Simply put two slashes (//) at the starting of a line and type your comments after it. If your comments will span more than one line, begin with /* and end with */.

For example:

```
/*
This is a comment that
spans multiple lines.
*/
```

Now there's *no excuse* for not commenting all your scripts!

The next interesting thing in the example is the fact that you can create **local variables** within you method that only "live" as long as the method runs. This is really a handy feature! There is no limit to the number of variables you can create per method, and you don't even have to "type" them (i.e. tell the compiler what data type the variable is – integer, string, etc.). What's more, you can change the data types within the variables at any time.

Consider the following code snippet (piece of code):

```
var num = 1;
var num2 = 2;

//num3 will equal 3;
var num3 = var 1 + var 2;

num = 'This is ';
num2 = 'cool! ';

//num3 will equal 'This is cool!';
num3 = var1 + var2;
```

This **is** cool!

Getting back to our “dialog_warning” method – move the sample code from “showDialogWarning” into your method. Now, let’s change the code so it reads:

```
//show dialog  
var thePressedButton = plugins.dialogs.showWarningDialog('Warning!',  
arguments[0], 'OK', 'Cancel');
```

The “msg” parameter in the “showDialogWarning” function will be replaced by the FIRST *array* argument we pass into the method (in Java and JavaScript, arrays are zero-based, so the first array element is 0 and not 1). We will also provide two buttons – “OK” and “Cancel” that will be stored in the local variable called `thePressedButton`. Now, we need to return the value stored in `thePressedButton` to the calling method – so we can decide what to do, based on what button the user clicks.

NOTE: You can pass any number of parameters in a single call to another script – and you would access them sequentially.

For example we could say:

```
var x = myScript('Param 01', 2, 'Param 03'
```

Then you could access the passed parameters with `arguments[0]` (equals ‘Param 01’), `arguments[1]` (equals the number 2), `arguments[2]` (equals ‘Param 03’).

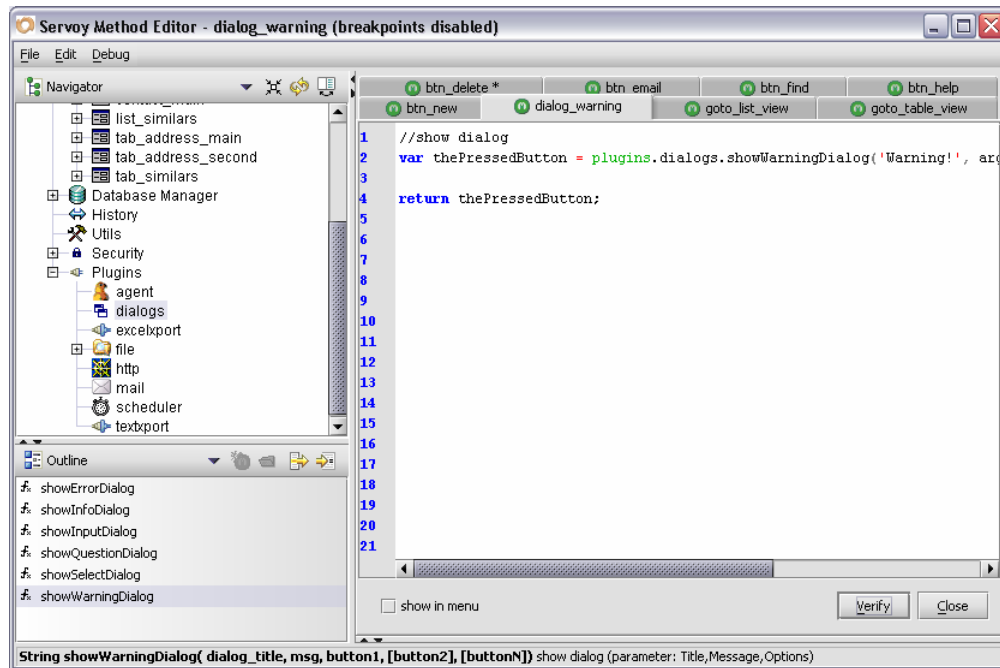
To return a value – we simply use the keyword “return” (no quotes) followed by the value that we want to return.

In this case, we want to return the contents of the local variable `thePressedButton`. So we’ll modify the script to include `return thePressedButton;` at the end.

The entire script is:

```
//show dialog  
var thePressedButton = plugins.dialogs.showWarningDialog('Warning!',  
arguments[0], 'OK', 'Cancel');  
  
return thePressedButton;
```

OK, add the `return` command to your script and click the “Verify” button. Your script should now look like this:



Next, click back on your “btn_delete” Script editor tab – and we’ll enter the rest of the code for the method.

The entire script should read:

```
var btn = globals.dialog_warning('Are you sure you want to delete this
record?');

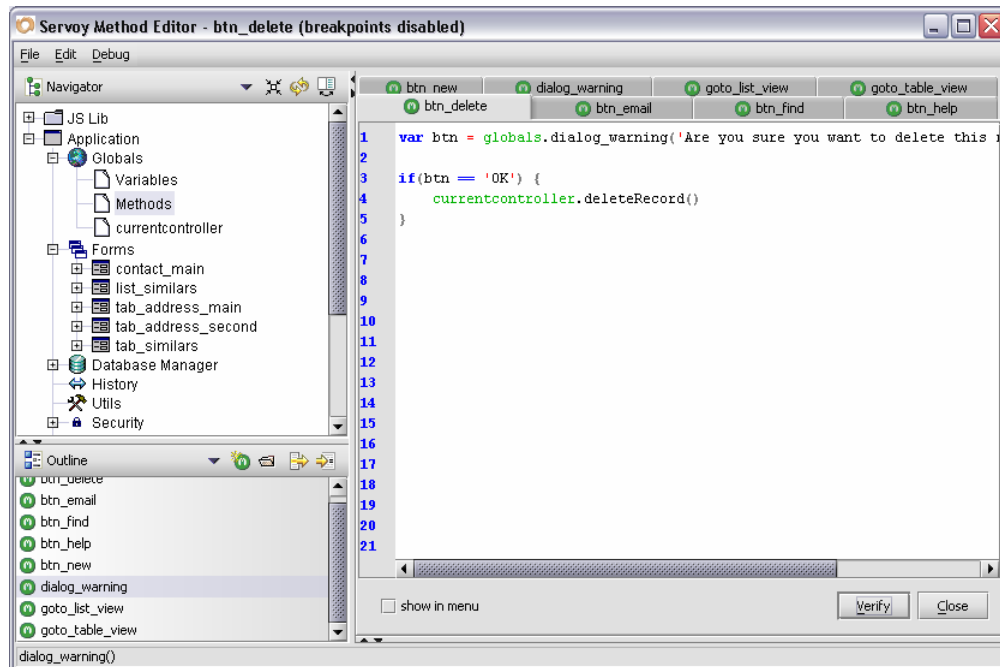
if(btn == 'OK')
{
    currentcontroller.deleteRecord()
}
```

In the script, we are creating a new local variable called `btn` to hold the contents of the result that was passed back from our `dialog_warning` method (the contents of the local variable called `theButtonPressed`). Then we use the JavaScript `if` statement to see if the user clicked the OK button – if so, we delete the record. If the text passed back was not 'OK' (case sensitive) – then we do nothing.

There’s a few very important things to note about the JavaScript `if` statement:

- The `if` is **case sensitive**.
- We use **two** equals signs `==` in an `if` statement (comparison operator).
- The code for what we want to happen if the statement is true, is surrounded by curly brackets `{ }`.

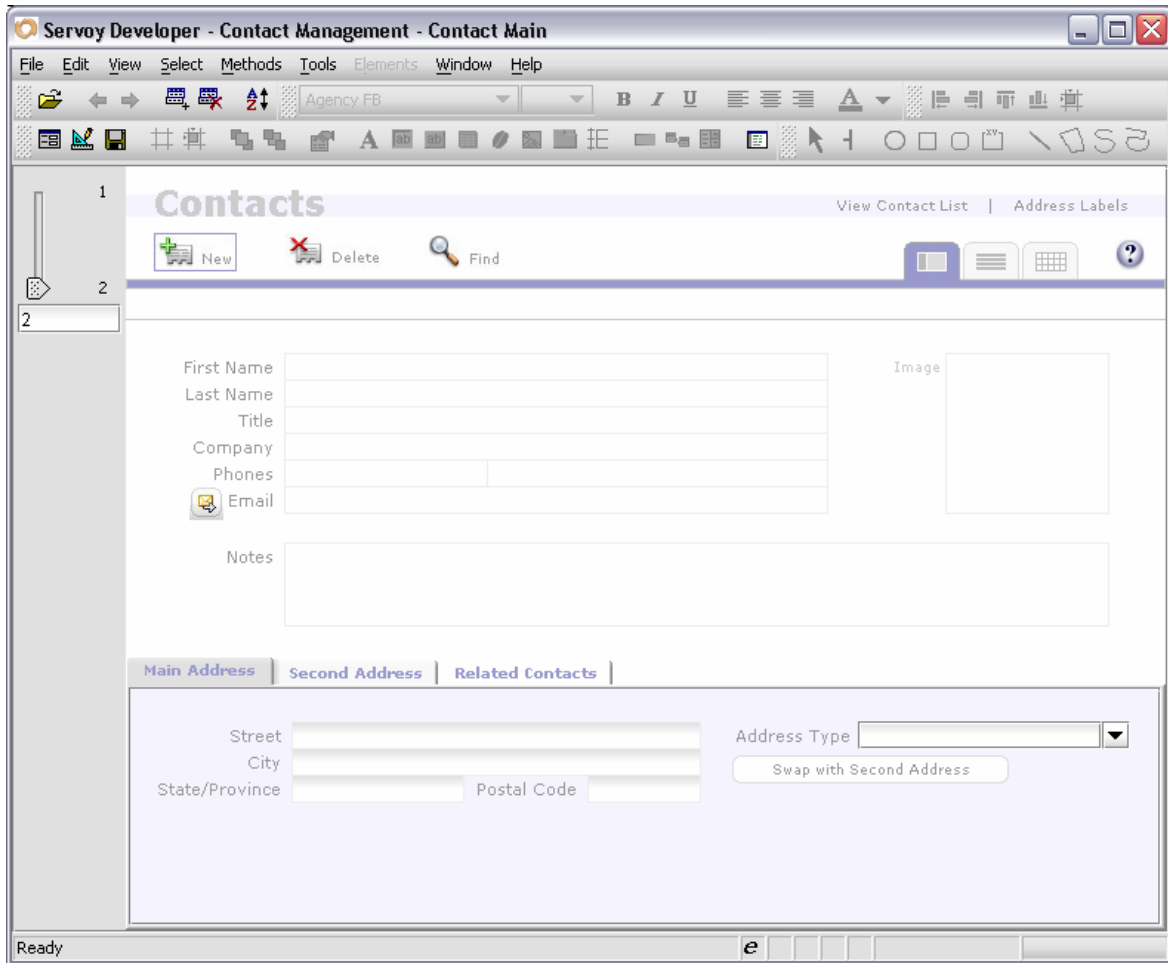
You can either type in the code above – or better yet – just copy/paste the code into your `btn_delete` script:



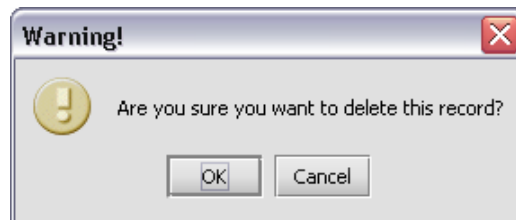
I know you're probably dying to see if the script really works or not – so you can switch back to the Servoy Designer window and try it.

That's another really cool thing about Servoy: the Method Editor window is non-modal so you can work on scripts whether you're in the Designer mode (Layout Mode) or the Data mode (Browse Mode).

Click the “New” button in Data mode to create a new record:

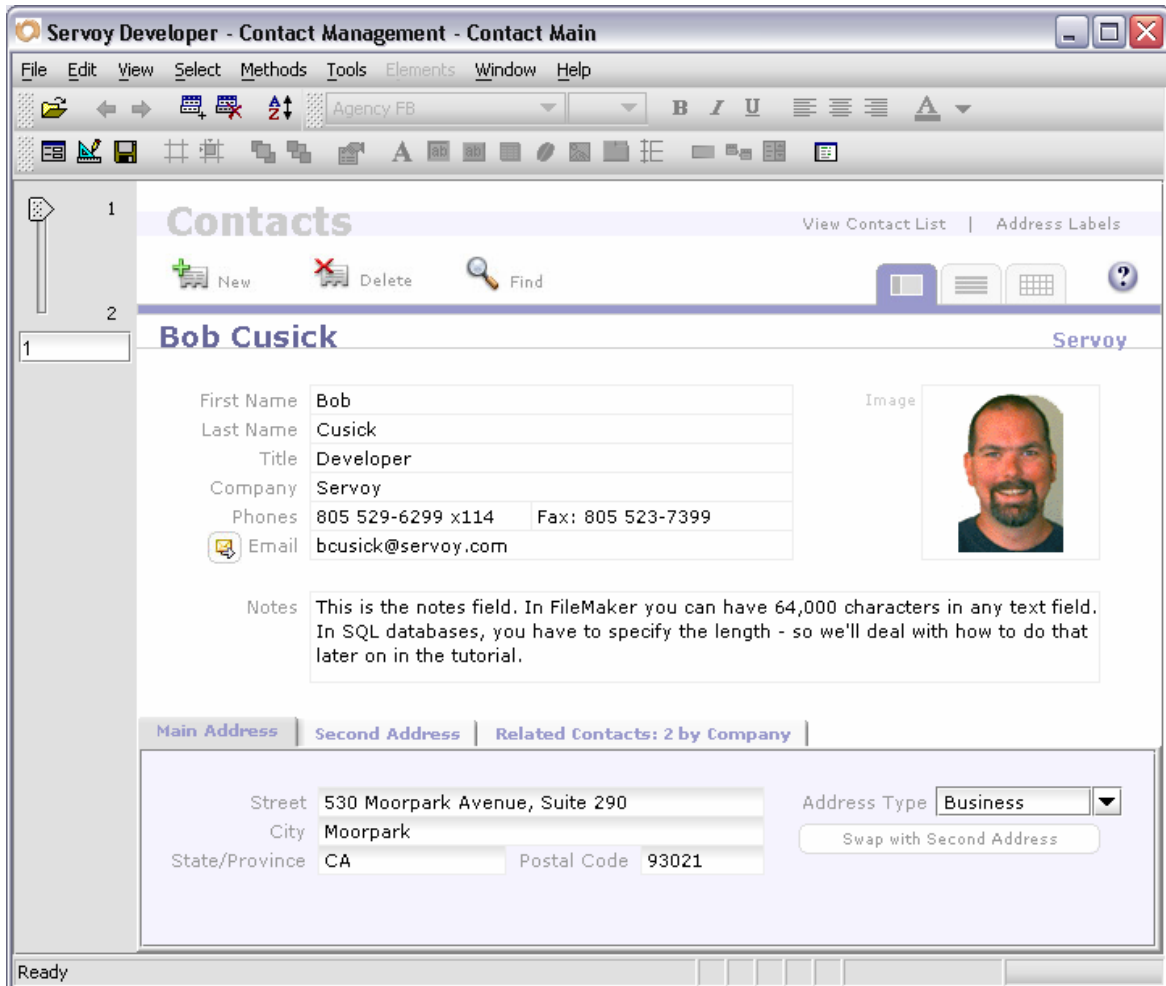


Then click the “Delete” button to delete the record you just created. You should see the following dialog box (shown on Windows XP):



Notice that the dialog has the “Warning!” title – and contains two buttons “OK” and “Cancel” – with the “OK” button appearing first.

Click the "OK" button and notice that you now only have one record:



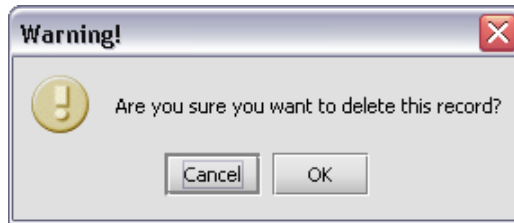
Pretty cool, huh?

Let's say you wanted the "Cancel" button to come first, and not the "OK" button. All you have to do is reverse the order of the parameters in your dialog_warning method (don't forget to press the "Verify" button in the lower right after you make the change):

```
//show dialog
var thePressedButton = plugins.dialogs.showWarningDialog('Warning!',
arguments[0], 'Cancel', 'OK');

return thePressedButton;
```

Now if you create a new record and try to delete it – you should see:



This warning dialog will use the OS-specific icon for the exclamation point. So, if you're using Mac OS X (or your customer is!) – they will see the big yellow triangle with Aqua-style buttons. No graphics to create, nothing else to code. Servoy handles it for you automatically! Nice.

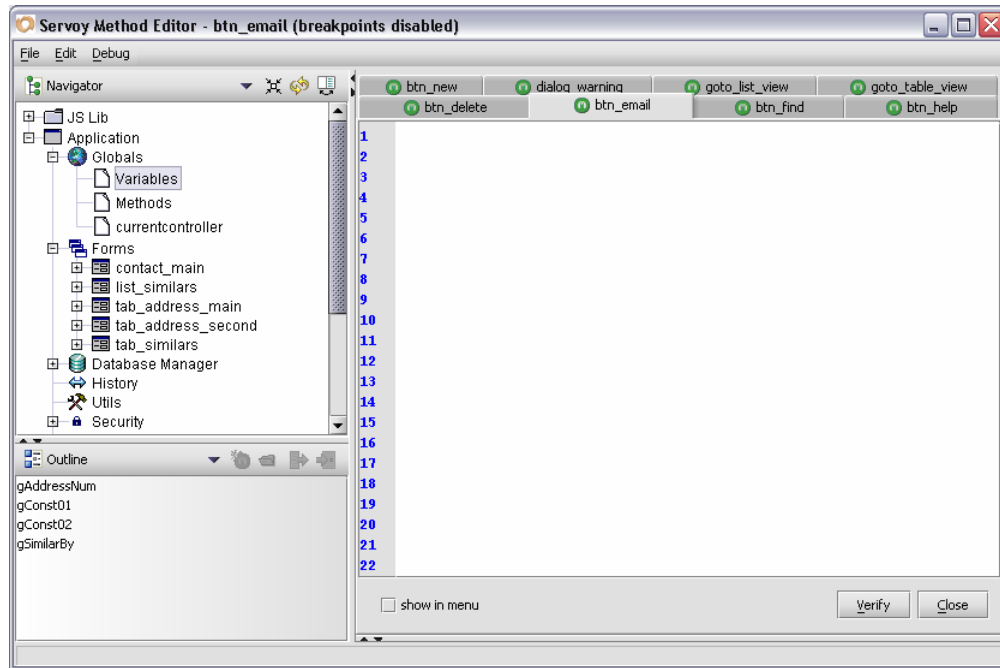
Since we haven't created forms for the list view, table view and help screens yet – we'll leave those methods blank for the time being and come back to fill them out later.

The last method we created earlier was the `btn_email` method. Servoy comes with a mail plug-in that allows you to directly send an email (without using your email client) or you can use `application.showURL` and the mail will be sent from the local email client. In this case, because we want our Servoy solution to function like the FileMaker solution, we'll use the `showURL` method - and I'll also include the code if you wanted to use the SMTP plug-in.

We'll start out by creating a new form that will let us specify the message and the body of the email. Before we get to creating the form - we have a choice to make here – we could make a database to store all the emails so we could view them later. In order to ensure that we get this tutorial done, we'll just use global fields.

EXTRA CREDIT: Create a new table called "email" that uses the `contact_id` to create linked emails for each contact.

We'll start by creating some globals: gEmail_Subject, gEmail_Body, gEmail_To, gEmail_Cc, gEmail_Bcc, gEmail_From (all TEXT globals). Click on the "Variables" section of the "Globals" tree in the Method Editor – and then either right click – or choose "Create Global Variable" from the "File" menu.



Once you're done defining the globals – go back to the Servoy Designer window and create a new form by choosing "New Form" from the "File" menu.

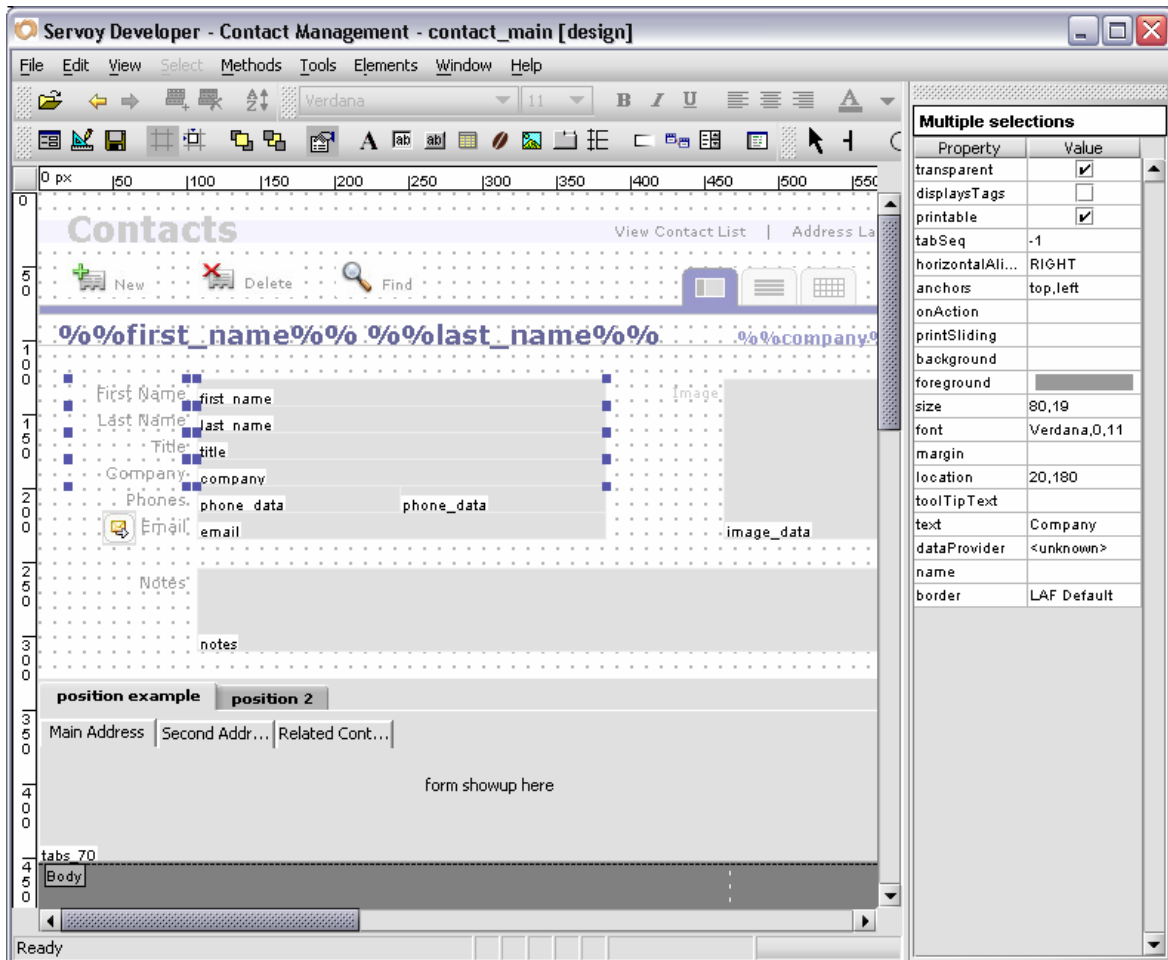
Choose the "contact_mgmt" connection, no style, and click on the "contact" table. Name the form "dialog_email" and click OK.

Don't place any fields on the form (we're going to start with a blank form) – click the "OK" button.

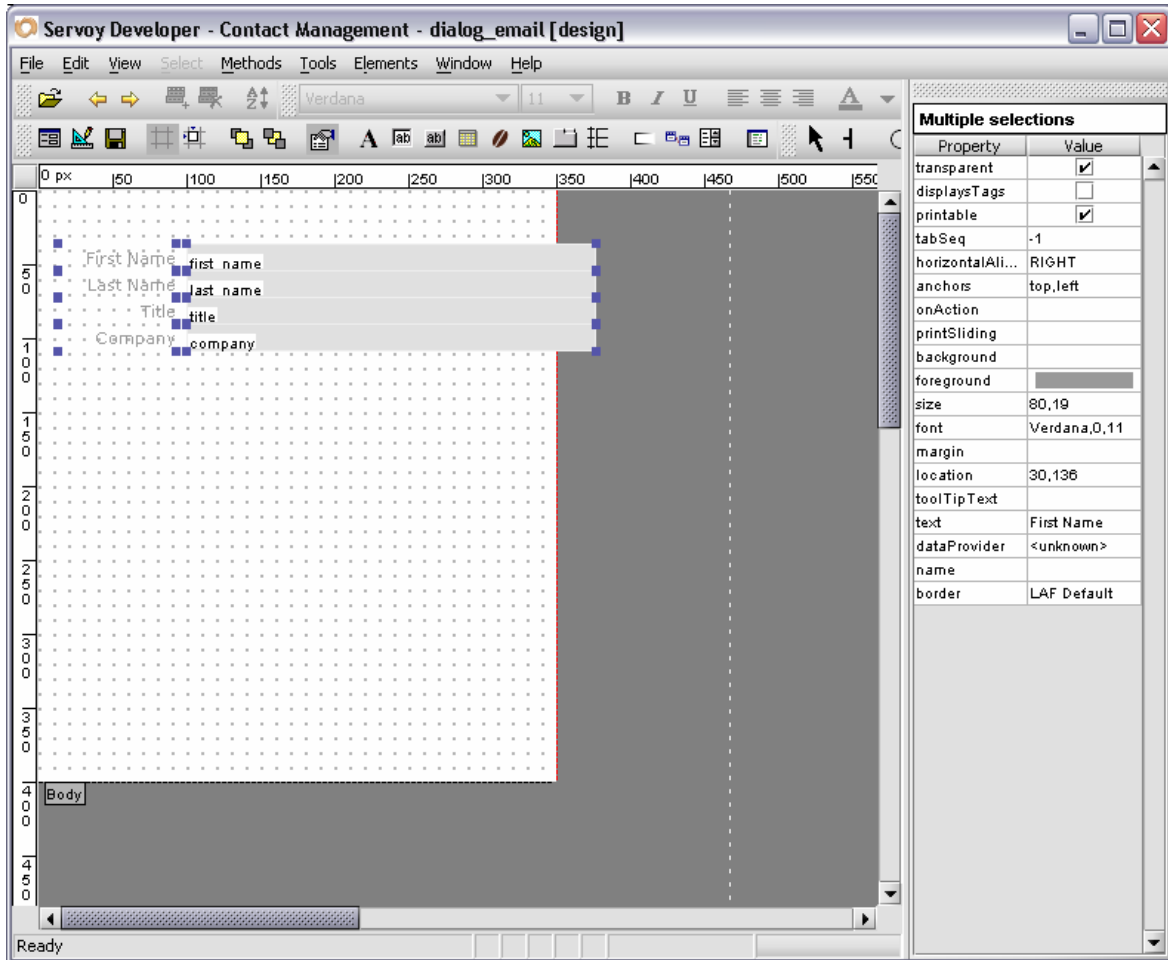
Once you have the form – we're going to change some of the default properties. Click anywhere on the form itself and modify the form properties in the Properties panel:

Object Type	Property	Value
Dialog Email Form	ShowInMenu	FALSE (UNCHECK checkbox)
	Width	350
	Scrollbars	Vertical: never, Horizontal: never
	OnShow	New Method (NOT GLOBAL METHOD): on_show
	Controller	NONE
	TitleText	Specify Email
Body Part	Height	400
	Background	Color: Red = 255, Blue = 255, Green = 255

Rather than having to place all the fields on the form and then go back and reset all the colors for the type and the field borders, etc. – it's a lot simpler to copy what we need from our existing `contact_main` form. From the "Window" menu – choose `contact_main`. Then, holding down the CTRL key (PC) or SHIFT key (Mac) – select the "First Name" label and field all the way down to the "Company" label and field.

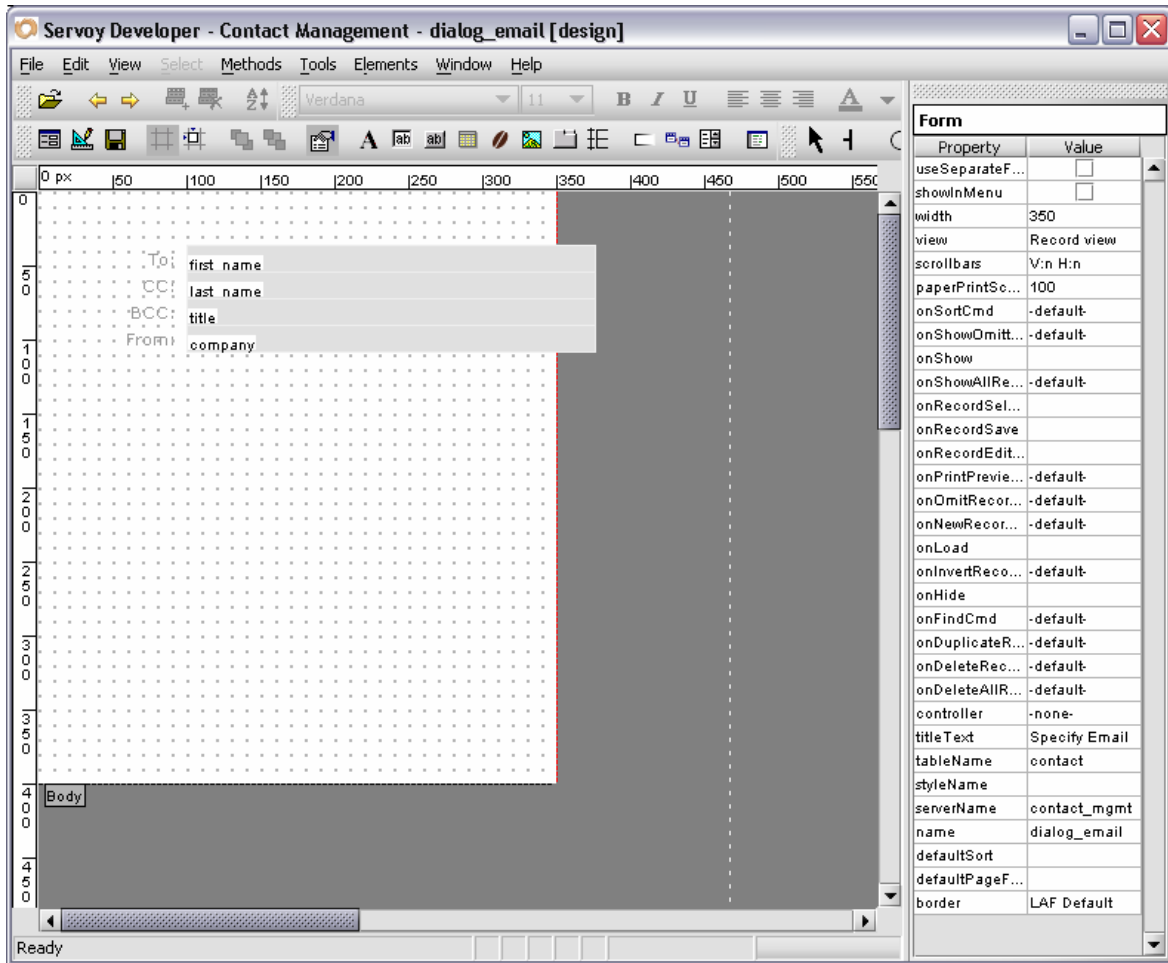


Choose "Copy" from the "Edit" menu, and then click the YELLOW back arrow at the top of the screen (or choose "dialog_email" from the "Window" menu) to return to the dialog_email form – and paste the elements.

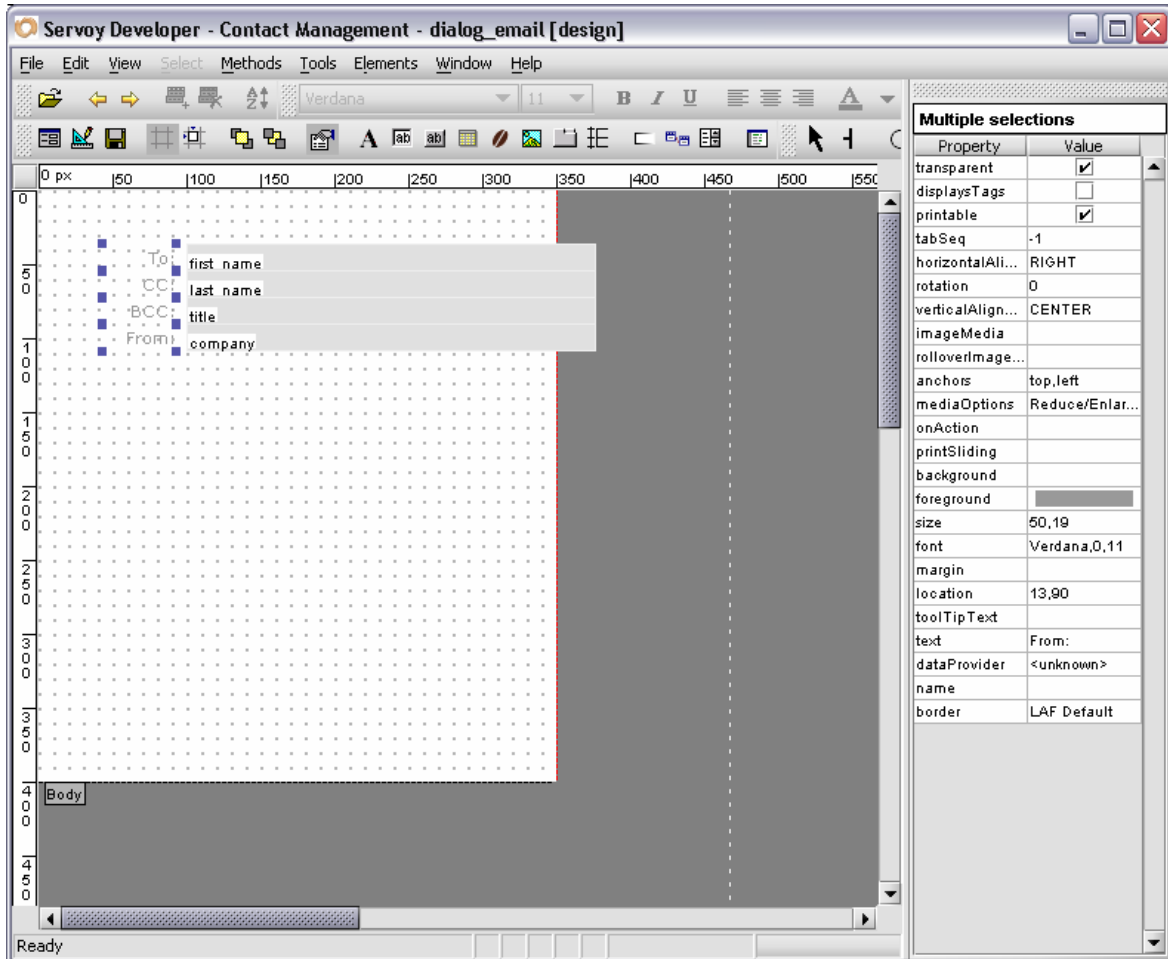


Now click somewhere on the white form to deselect the objects. Double click the label "First Name" and then press CTRL-A (PC) or COMMAND-A (Mac) to select all the fields, and type "To:" (no quotes).

Repeat the above so the labels read: To:, CC:, BCC:, From:

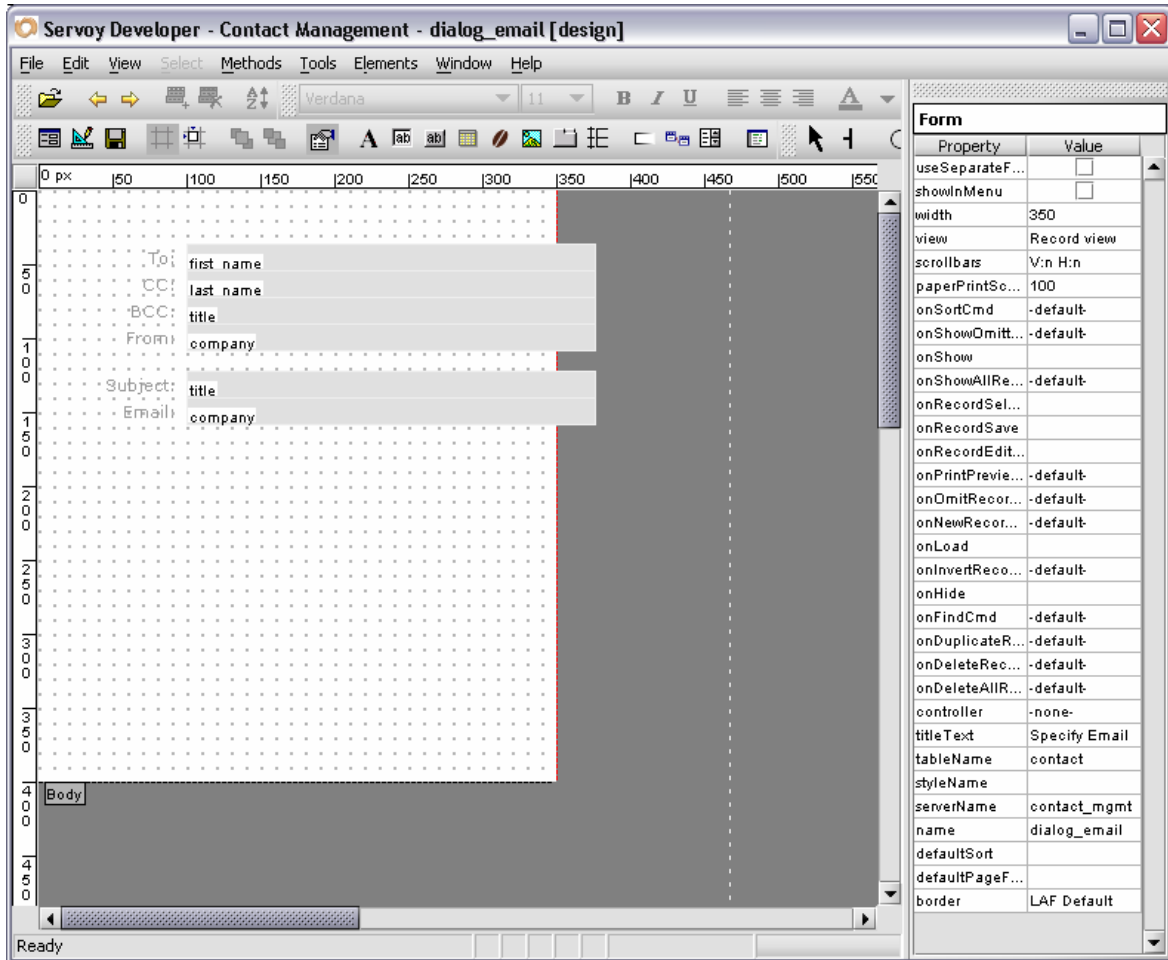


Now we're going to re-size the labels to make them shorter (so we can scoot all the fields over to the left). CTRL-click to select all the field labels and then hold down the SHIFT and CTRL keys and press the LEFT ARROW KEY on your keyboard 3 times. LET GO of the SHIFT key and press your RIGHT ARROW KEY 3 times.



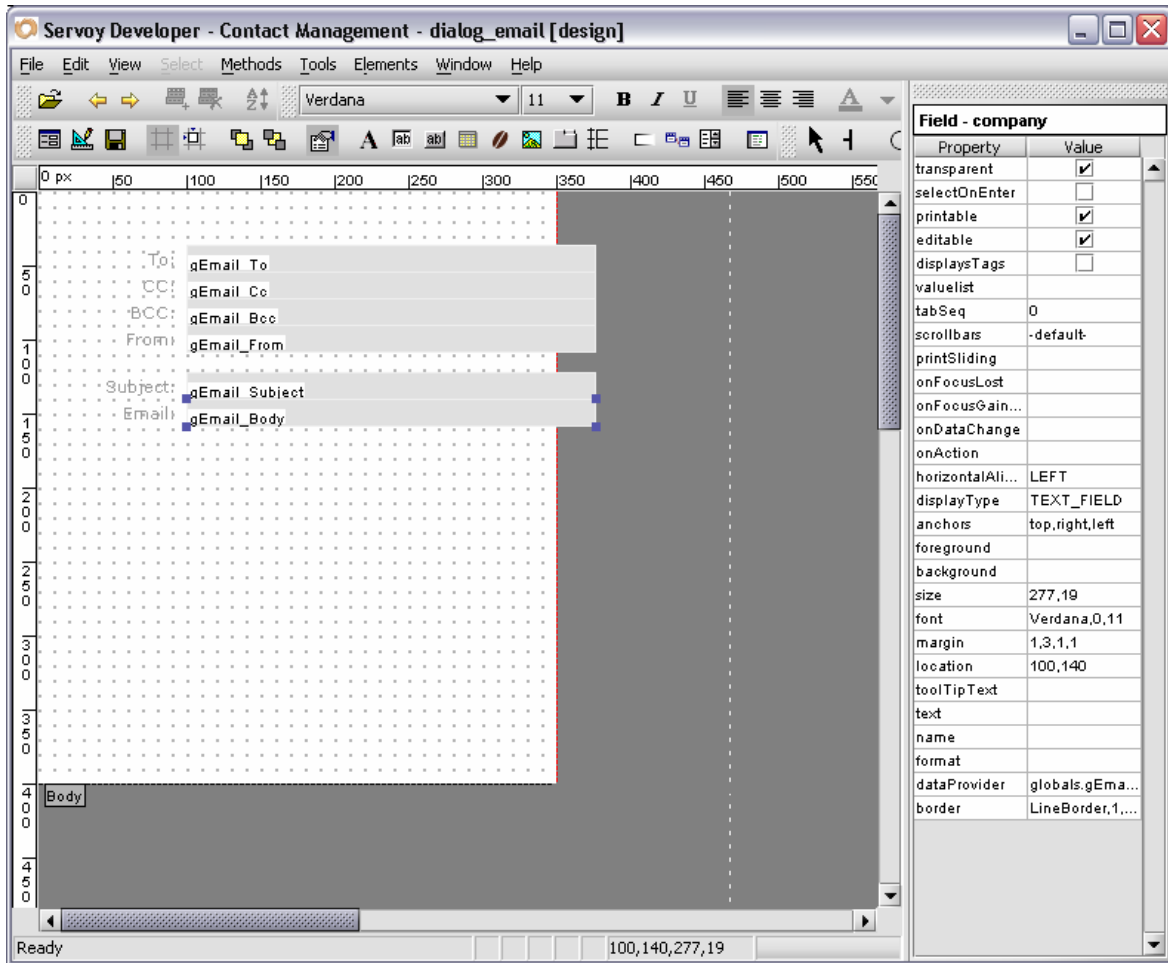
Now all of your labels are the same size and are all aligned – without clicking and dragging stuff all over the place.

Next we're going to make place holders for the Subject and Email Body – by copying/pasting the "BCC:" and "From:" labels and fields. Change the label of the "BCC:" you just pasted to "Subject:" and change the text from "From:" to "Email:"



OK, let's change the fields to reference the globals we created earlier. Double-click on the field (now called "first_name") next to the "To:" label and choose the field gEmail_To. Repeat the process for the remaining fields – matching them up to the globals we created earlier.

When you're done – you should see something like this:



Next, we'll arrange the items by moving them to the left, and by making the fields slightly shorter. Once you've done that – click on the gEmail_Body field and make it taller. In the Properties panel, change the `scrollbars` property to: Vertical = when needed, Horizontal = never, and change the `displayType` property of the gEmail_Body field to `TEXT_AREA`.

Finally, we'll add two buttons to the bottom of the layout – one titled "Cancel" and one titled "Send Email". On both buttons – set the `font` property to Verdana, 11pt, Plain.

On the "Cancel" button – make a new method (NOT a global method) for the `onAction` property called "btn_cancel" – and hook up the "Send Email" button's `onAction` method to another new method (again, NOT a global method) "btn_send_email".



Now we're going to add some code to the methods we created. Open the Method Editor (if the window is not already open) and click on the Script editor tab `btn_cancel`. Click on the "Application" tree and double-click the function `closeFormDialog`. Replace `[closeAll]` with the word `true`.

Your code should look like this:

```
application.closeFormDialog(true);
```

Click the "Verify" button in the lower right of the Method Editor dialog – and then click on the tab `btn_send_mail`. Before we enter the code for the sending of the email – we need to decide how we want our new application to function. Do we want to duplicate the functionality in FileMaker – and have an email client open and a new message appear; or do we want Servoy to simply send the email directly without using an email client? I'll show you the code required for BOTH approaches – and you can decide which is better in your situation.

We'll take the approach of opening the email in the local mail client first. To accomplish this, we're going to create a mailto URL and then use `application.showURL()` to actually create the email.

Here's the code:

```
//send email - check to make sure the TO, FROM and SUBJECT are filled out

if(globals.gEmail_To && globals.gEmail_Subject)
{
    // set a local variable to hold a URL string

    var url = 'mailto:' + globals.gEmail_To;

    //replace any '&' characters in the subject and body

    url += '?subject=' + utils.stringReplace(globals.gEmail_Subject, '&', '%26');
    url += '&body=' + utils.stringReplace(globals.gEmail_Body, '&', '%26');
    url += '&cc=' + globals.gEmail_Cc;
    url += '&bcc=' + globals.gEmail_Bcc;

    //replace spaces and return characters

    url = utils.stringReplace(url, ' ', '%20');
    url = utils.stringReplace(url, '\n', '%0d');

    application.showURL(url);
}
else
{
    //not everything was filled out

    plugins.dialogs.showErrorDialog( 'Error!', 'You must specify a TO and SUBJECT in order to send the email.', 'OK');
}
```

Let's have a look at the code:

First we check to see if the TO and SUBJECT lines are filled out and if they are, then we create a local variable called `url` to hold our string. After that, we concatenate (combine) more data to our `url` variable:

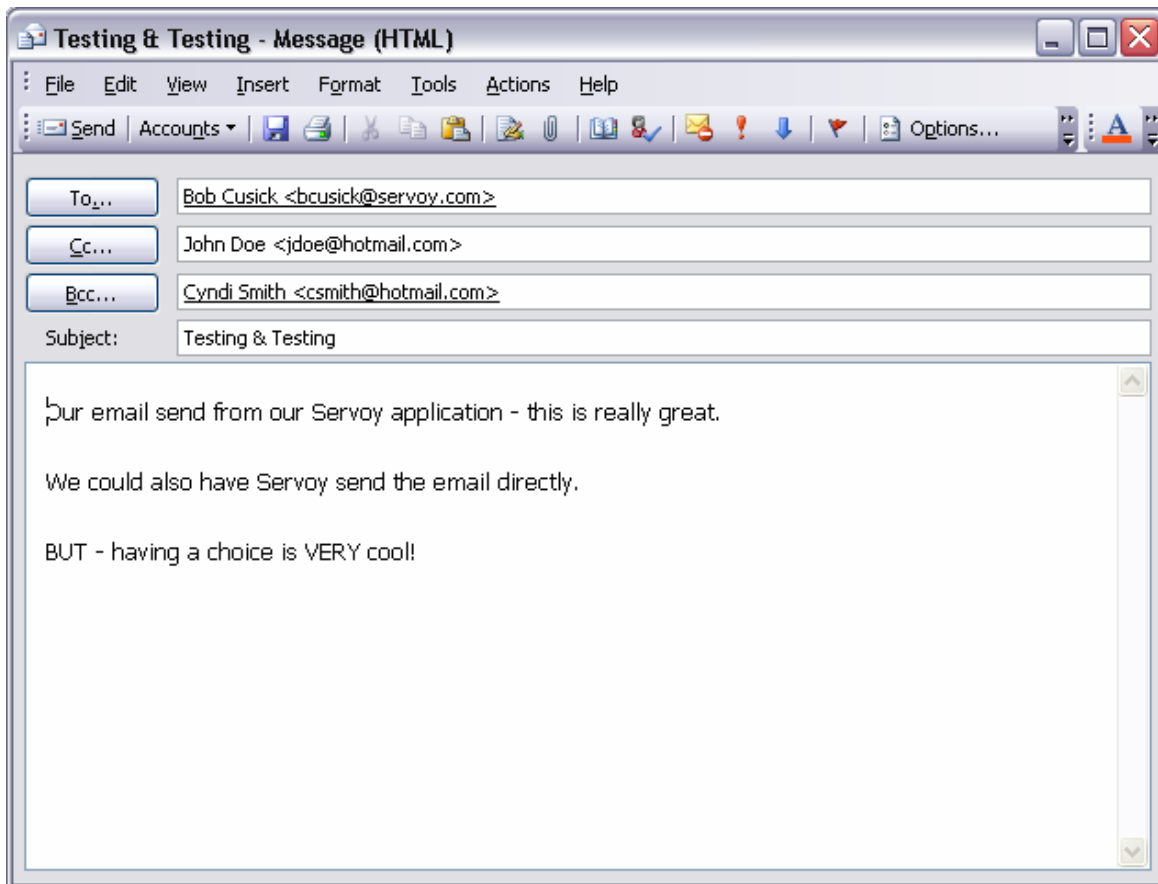
```
url += '?subject=' + utils.stringReplace(globals.gEmail_Subject, '&', '%26')
```

Notice we are concatenating data using the `+=` operator. This is a JavaScript shorthand way to say: `url = url + 'string'`. Handy! We are going to replace any ampersand characters (&) in the subject and body with the HTTP equivalent (%26) – so that our URL string will work correctly.

After we've set all the fields to build one big string – we also need to replace any spaces with %20 (the HTTP equivalent for space) and any return characters (\n in JavaScript) with %0d – to make sure all of our data in the email body makes it into our final email.

Finally – after the last `else` – we will show a dialog to the user – if they forgot to fill out either the TO or the SUBJECT fields.

When you click the "Email" button – the local email client will launch, and the data will be filled out automatically:



Alternate Method

If you want Servoy to send the email directly (without using the local email client) then we can use the mail plug-in. One important note here – you have to specify an outgoing SMTP mail server in the Preferences in order for this to work. Also, your outgoing SMTP email server must NOT require user authentication (if your outgoing SMTP server does require user authentication – you can use the ezSMTP JavaBean as discussed below).

Here's the code for using the Mail plug-in:

```
//send email - check to make sure the TO, FROM and SUBJECT are filled out

if(globals.gEmail_To && globals.gEmail_From && globals.gEmail_Subject)
{
    // set a local variable to the result of the
    //attempt to send the mail

    var success = plugins.mail.sendMail( globals.gEmail_To,
    globals.gEmail_From, globals.gEmail_Subject, globals.gEmail_Body,
    globals.gEmail_Cc, globals.gEmail_Bcc);

    if(success)
    {
        //everything went OK - so show an info dialog
        //and close the form in dialog

        plugins.dialogs.showInfoDialog( 'Send Email', 'Email successfully
        sent.', 'OK');
        //perform the script "btn_cancel"

        btn_cancel();
    }
    else
    {
        //there was an error - so tell them.

        plugins.dialogs.showErrorDialog( 'Error!', 'There was an error
        encountered when trying to send the email.', 'OK');
    }
}
else
{
    //not everything was filled out

    plugins.dialogs.showErrorDialog( 'Error!', 'You must specify a TO, FROM
    and SUBJECT in order to send the email.', 'OK');
}
```

This script looks long – but half of it is comments.

At the top of the script, we check to make sure they have filled in the TO, FROM and SUBJECT fields:

```
if(globals.gEmail_To && globals.gEmail_From && globals.gEmail_Subject)
{
```

If they have, then we set the local variable `success` to the result of the plug-in command to send the email:

```
var success = plugins.mail.sendMail( globals.gEmail_To, globals.gEmail_From,
globals.gEmail_Subject, globals.gEmail_Body, globals.gEmail_Cc,
globals.gEmail_Bcc)
```

If the variable `success` is TRUE, then we show a confirmation dialog that tells the user their email was sent successfully:

```
plugins.dialogs.showInfoDialog( 'Send Email', 'Email successfully sent.', 'OK')
```

If the variable `success` is NOT TRUE – then an error occurred – so we inform the user with a warning dialog:

```
plugins.dialogs.showErrorDialog( 'Error!', 'There was an error encountered when trying to send the email.', 'OK')
```

Finally – going back to our original `if` statement – if they didn't fill out the TO or the FROM or the SUBJECT – then we show them a warning dialog:

```
plugins.dialogs.showErrorDialog( 'Error!', 'You must specify a TO, FROM and SUBJECT in order to send the email.', 'OK')
```

I know what you're asking yourself – you're saying – "Hey, Bob! Didn't we create a global method called 'dialog_warning' that we could call from other methods? Why didn't we just use that global method?" GOOD POINT! However, our global method `dialog_warning` was designed to show an "OK" and "Cancel" button. In this case – there was nothing to cancel – it was just an informational dialog.

EXTRA CREDIT: Create two new global methods called "dialog_info" and "dialog_warning_ok" that will allow you to re-use them throughout your solution and modify the code to use the global methods.

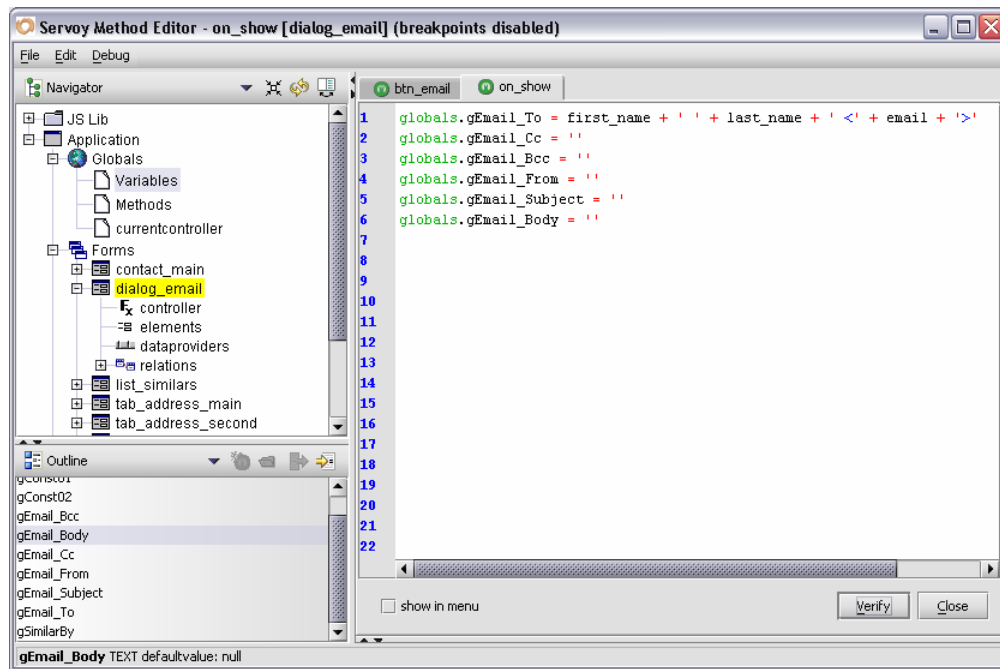
When the warning dialog opens – we want it filled out with the name of the contact we're currently on – and we want it to clear out anything that was entered in a previous email. To accomplish this – we're going to use the `onShow` property of the form to create a new method called `on_show`. Once you've double-clicked the `onShow` property of the `dialog_email` form – switch back to the Method Editor and we'll enter the code. Click on the "+" next to the form `dialog_email`.

We want to set the "To:" field (really the `gEmail_To` global) to the name of the person and their email address surrounded by brackets. Click on the "dataproviders" object and use a combination of double-clicking on the field names and typing to come up with the following code:

```
globals.gEmail_To = first_name + ' ' + last_name + ' <' + email + '>'
```

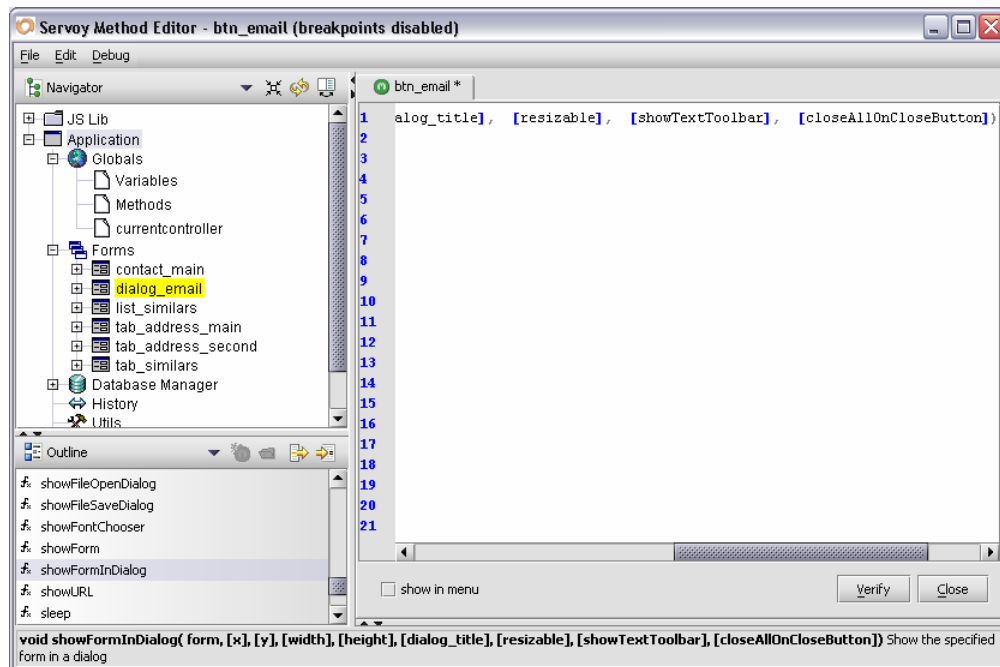

Next, we're going to "blank out" all the rest of the fields:

```
globals.gEmail_To = first_name + ' ' + last_name + ' <' + email + '>'
globals.gEmail_Cc = ''
globals.gEmail_Bcc = ''
globals.gEmail_From = ''
globals.gEmail_Subject = ''
globals.gEmail_Body = ''
```



Click the "Verify" button in the lower right to save the method and then you can click the "Close" button in the lower right to close the method in the Method Editor.

Now that we have the code to make the warning dialog function – let’s add out the code that will let us show the dialog. We already have a global method called “btn_email” – so let’s click on that tab in the Method Editor and enter the code.



Click on the “Application” tree and double-click showFormInDialog.

You’ll see this code inserted into your method:

```
application.showFormInDialog(form, [x], [y], [width], [height],  
[dialog_title], [resizable], [showTextToolBar], [closeAllOnCloseButton]);
```

The showFormInDialog function will open the form we just created in a modal dialog. You can also optionally specify the x (horizontal) and y (vertical) position of the dialog; in addition to the height, width, and title of the dialog; whether the dialog can be resized; whether or not to show the text toolbar; and whether clicking the “X” in the upper right of the dialog will close all open forms in that dialog (more on that later).

If you were to try the “mode sample” option of the same function – you’d see:

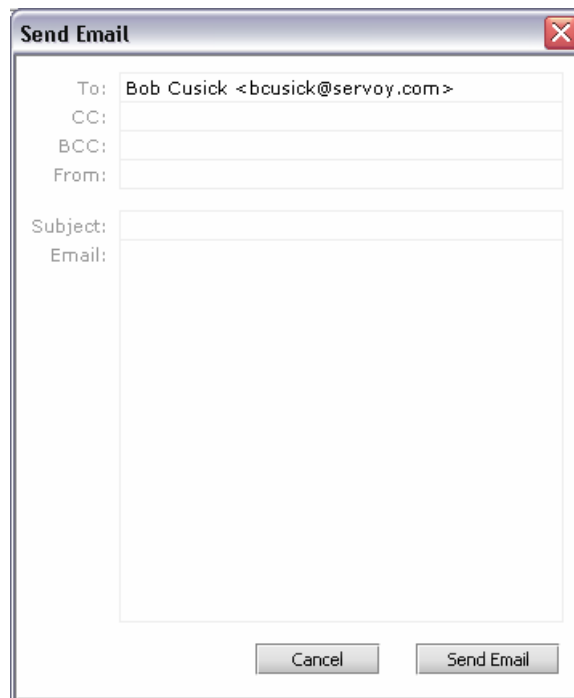
```
//Show the specified form in a dialog, on default location and size (x,y,w,h)  
//application.showFormInDialog(contacts);  
//Note: this call is blocked until the dialog is closed  
//Show the specified form in a dialog, on specified location and size with  
custom title, not resizable but with text toolbar,  
//And if there are multiple forms open in the dialog the Close button should  
close them all at once  
application.showFormInDialog(forms.contacts,100,80,500,300,'my own dialog  
title',false,true,true);
```

For our code – we want the dialog to show in the middle of the screen; with the size we’ve specified for the form itself (in the form width and the body part height); we don’t want the dialog to be resizable; nor do we want the text toolbar to show (because they’re not formatting text in our email); and we really don’t care if all the forms are closed when the “X” is clicked. SO, we can modify our code to read:

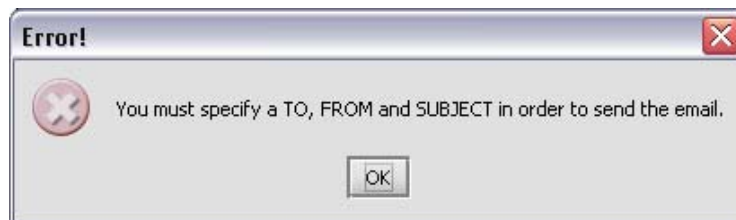
```
application.showFormInDialog(forms.dialog_email, -1,-1,-1,-1, 'Send Email',  
false, false, true)
```

The `-1, -1, -1, -1` tells the function to use the default `x` and `y` position (the center of the screen) and to make the dialog window the size of the form. Once your code reads like the above, click the “Verify” button – and let’s try it.

Switch back to the Designer window – and go to the `contact_main` form. Go into Data Mode – if you're in Designer Mode, use CTRL-L (PC) or COMMAND-L (Mac) or click the Designer tool in the toolbar – and click the email button. You should see your new dialog appear:

A screenshot of a "Send Email" dialog box. The dialog has a title bar with a close button (X). Inside, there are input fields for "To:", "CC:", "BCC:", "From:", "Subject:", and "Email:". The "To:" field is pre-filled with "Bob Cusick <bcusick@servoy.com>". At the bottom, there are "Cancel" and "Send Email" buttons.

Try clicking the “Send Email” button – and you should see the error dialog:



Now click the "OK" button and the dialog should dismiss.

Congratulations – you’ve created your first custom dialog!

NOTE: You’ll probably get an error if you try to send the email – probably because you didn’t specify your outgoing mail server in the preferences. Choose "Preferences" from the "Edit" menu, click on the "Mail" tab – and enter your outgoing SMTP mail server.

ALSO NOTE: The Mail plug-in that ships for free with Servoy does NOT support user authentication – so if you need to authenticate before connecting to your outgoing server – you can use a \$35 JavaBean called EZsmtp (<http://www.ezjavabeans.com/ezsmtp/>) that will support it. For an example on how to use the EZsmtp bean – check out Servoy Magazine – <http://www.servoymagazine.com>.

The last thing we’re going to do in this section is to replicate some functionality that’s unique to FileMaker – namely the automatic creation of related records.

In Servoy – when you click or tab into a related field (and a related record doesn’t already exist) and then you exit the related field, a record is created for you automatically (just like in FileMaker). However, if you use another form in a tabpanel, portal or list – you have to create the related record before the user can enter data into it.

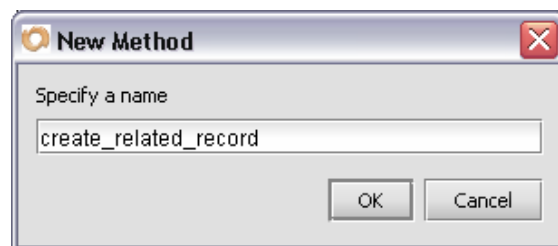
In our case – the user can be either on the "Main Address" tab or the "Second Address" tab – and they can click into any of the fields (street, city, state_province, postal_code or address_type) and begin entering data.

How do we solve this problem?

It’s really quite easy – we’ll create a new method called `create_related_record` in which we’ll check to see if the related record exists or not – if not, we’ll create it. "But Bob", you’re probably saying to yourself, "how will we trigger the script to fire in the first place?" AH! Good question.

Servoy, unlike FileMaker, is *event-driven* – so we can add the method to the `onFocusGained` event property on each of the fields on the form. That way, no matter which field they click into to begin entering data – a related record will be created automatically.

Let’s start by creating the method. Click on the "tab_address_main" form in the Method Editor, then click the green "M" button (or right-click on the form name) to create a new method. Name the method `create_related_record`:

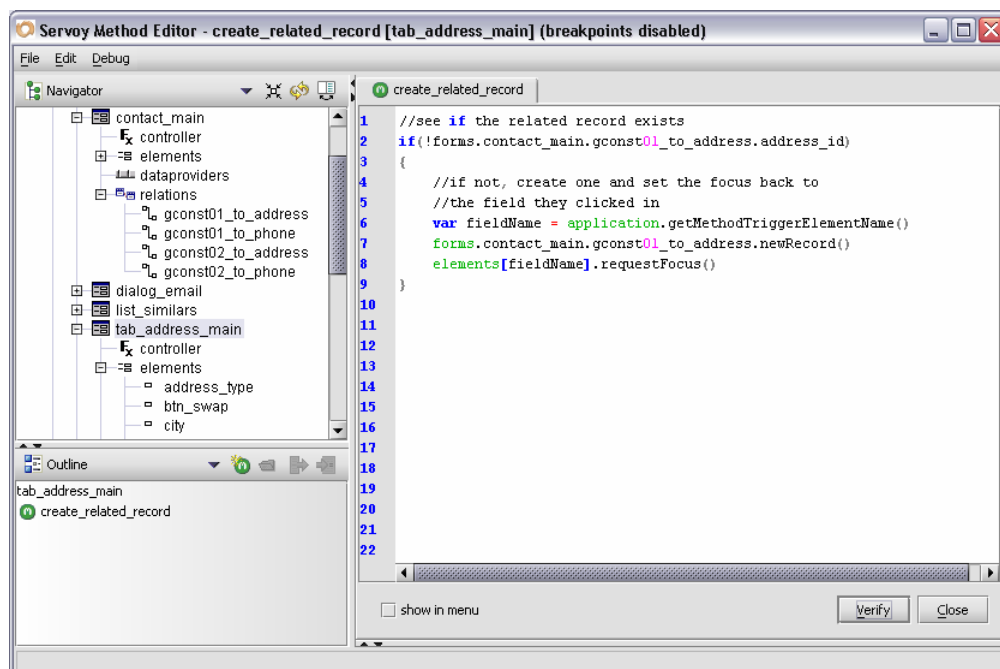


The code for the script is:

```
//see if the related record exists
if(!forms.contact_main.gconst01_to_address.address_id)
{
    //if not, create one and set the focus back to
    //the field they clicked in
    var fieldName = application.getMethodTriggerElementName();
    forms.contact_main.gconst01_to_address.newRecord();
    elements[fieldName].requestFocus();
}
```

First, the `if` statement looks through the relationship on the main contacts screen (the one we used to display the `tab_address_main` form) to see if there is already an `address_id`. The exclamation point before the field reference means “not” – so the `if` statement is saying “if there isn’t a related record with the `address_id` filled out, THEN do the following.”

We then store the NAME of the field that triggered the script using into a variable called `fieldName`. Next we create a new record using the relationship `gconst01_to_address` – this will automatically fill out the `contact_ID` and the `address_num` (the two fields used in the relation). Finally, we use `elements[fieldName].requestFocus()` to put the cursor back into the field that triggered the method – otherwise the cursor would just “disappear” even though the user clicked into a field.

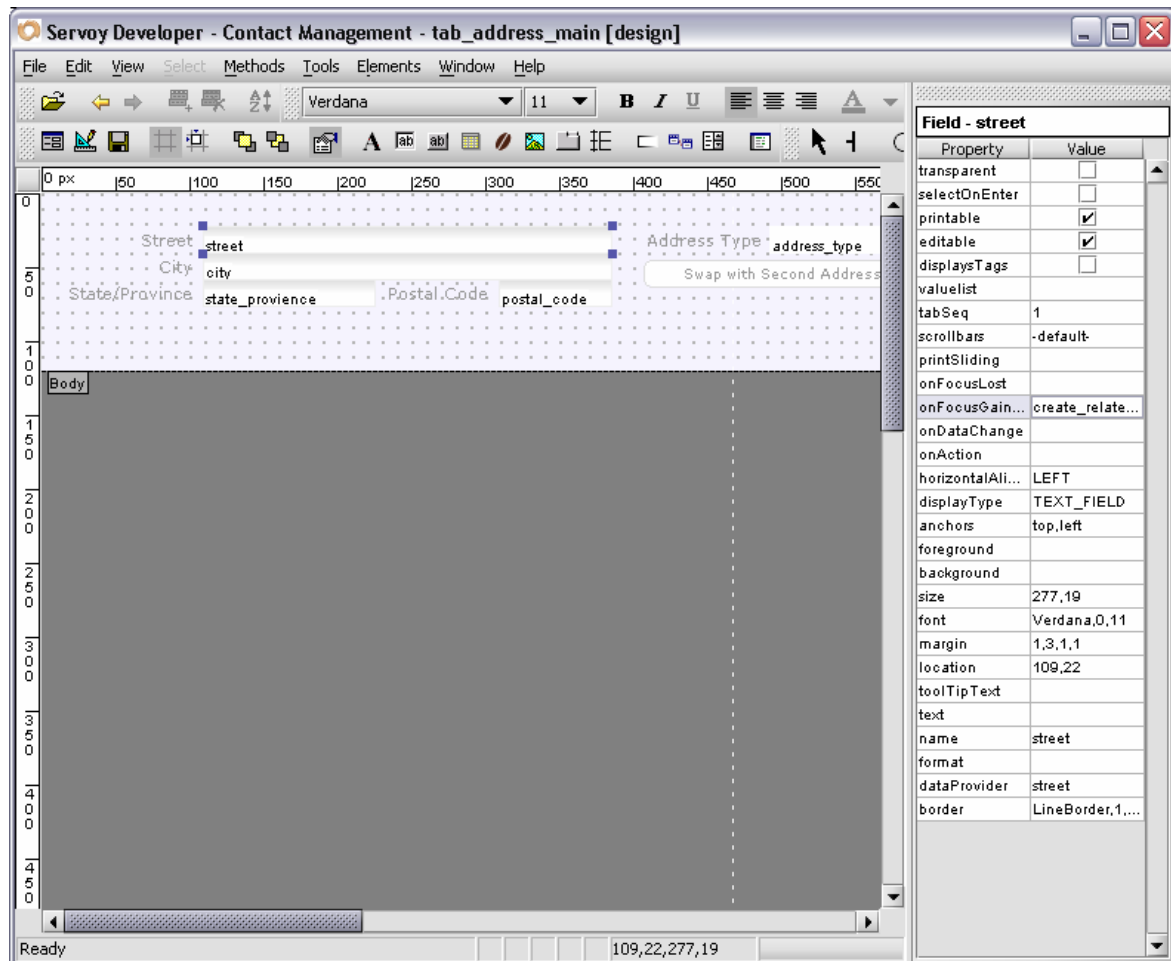


Once you have the code entered (or you copied/pasted it) – click the “Verify” button to save the method.

Now, select the entire method, copy it, and click on the “`tab_address_second`” form. Create a new method called “`create_related_record`” and paste the code in.

Then, simply change `if(!forms.contact_main.gconst01_to_address.address_id)` to `if(!forms.contact_main.gconst02_to_address.address_id)` and change `forms.contact_main.gconst01_to_address.newRecord()` to `forms.contact_main.gconst02_to_address.newRecord()` and click the "Verify" button.

Now, back in the Servoy Designer window – use the "Window" menu to navigate to the form called "tab_address_main." Click on the "street" field – double-click next to the "onFocusGained" property.



Choose the `create_related_record` method we created.



Repeat this process for all the rest of the fields on this form (street, city, state_province, postal_code, address_type). When you're done – use the "Window" menu to switch to the form "tab_address_second" and do the same thing for each of the fields (street, city, state_province, postal_code, address_type).

To try out your new functionality – switch to the "contact_main" form, go into Data mode and click in the "street" field on the "Address Main" tab. You'll see that you can enter data and it will be automatically saved.

Try the same thing on the “Address Second” tab – but this time click in the “Address Type” field and choose a value. That wasn’t so tough, was it?

Servoy Developer - Contact Management - Contact Main

File Edit View Select Methods Tools Elements Window Help

Agency FB

Contacts View Contact List | Address Labels

New Delete Find

Bob Cusick Servoy

First Name: Bob
Last Name: Cusick
Title: Developer
Company: Servoy
Phones: 805 529-6299 x114 Fax: 805 523-7399
Email: bcusick@servoy.com

Image:

Notes: This is the notes field. In FileMaker you can have 64,000 characters in any text field. In SQL databases, you have to specify the length - so we'll deal with how to do that later on in the tutorial.

Main Address Second Address Related Contacts: 2 by Company

Street: 1234 Main Street
City: Anytown
State/Province: CA Postal Code: 93021

Address Type: Home
Swap with

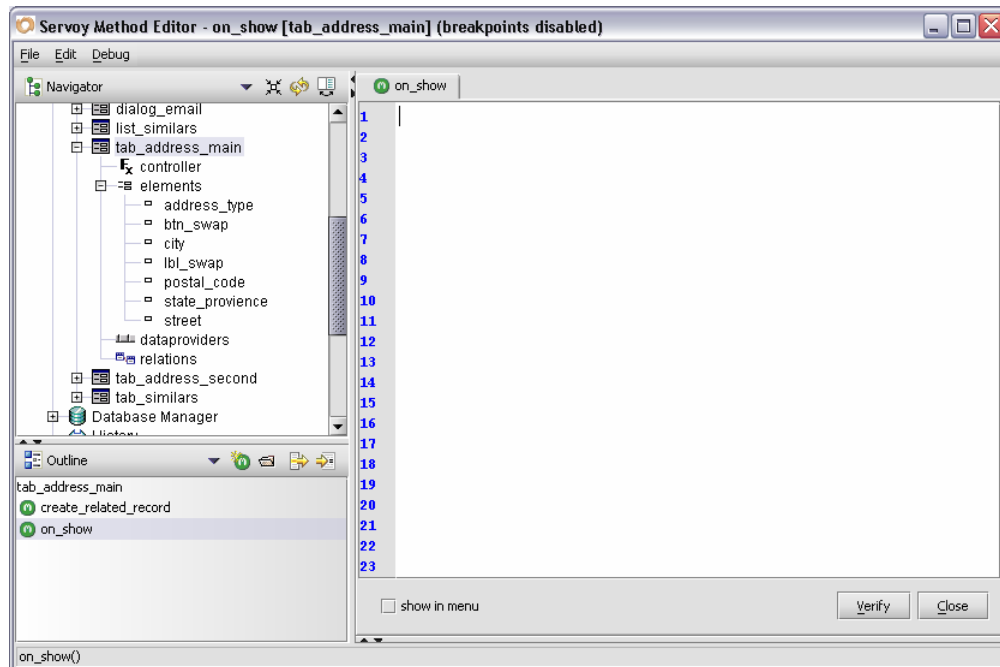
Home
Business
Home Office
Vacation

Ready

Just for the sake of completeness – the “Swap with” button on each form shouldn’t show up unless we have a record to “swap” with (i.e. swap the main address with the second address).

In Servoy – we can hide/show items automatically. To do this, we’ll create a simple method that we’ll attach to each of the address form’s “onShow” event property. The `onShow` event will trigger just before the form is drawn on the screen. Because we are using a tabpanel to display the forms, our method will be triggered every time we switch tabs.

In the Method Editor screen, click on the `tab_address_main` form and create a new method called `on_show`:



We're going to make sure that there is a main address record as well as a second address record and if so, then show the button and label – otherwise we will hide them.

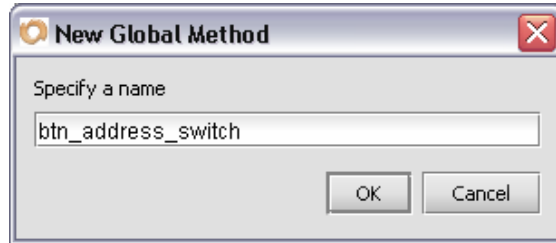
Here's the code:

```
if(forms.contact_main.gconst01_to_address.address_id &&
forms.contact_main.gconst02_to_address.address_id)
{
    elements.lbl_swap.visible = true;
    elements.btn_swap.visible = true;
}
else
{
    elements.lbl_swap.visible = false;
    elements.btn_swap.visible = false;
}
```

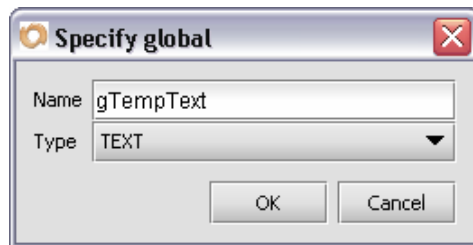
Pretty straight-forward! If you don't see all the elements displayed above under the "elements" node of the `tab_address_main` form – this means that you didn't fill out the NAME property of all the objects. Only objects with the NAME property filled in will show up in the Method Editor. This is to reduce the potential "clutter" that would happen if Servoy just automatically listed ALL items on your form. You only need to fill the "name" property – in the form's Properties panel – for those objects that you're going to want to control (change the color, size, position, show/hide, etc.) during runtime.

Now, copy your method code, switch to the `tab_address_second` form, create a new method called `on_show` and just paste the code there. Because we are checking for the existence of both records in the `if` statement, all you need to do is click the "Verify" button and you're done.

The LAST piece of business we need to take care of is what happens when the button is actually clicked (we know it will be visible only when there are both addresses). Click on the Methods node of the Globals tree and create a new global method called `btn_address_switch`:



Click on the Variables node of the Globals tree, and create a new global TEXT field called "gTempText":



Here is the code we're going to use for the "swap" button method:

```
//set global to street 1
globals.gTempText = forms.contact_main.gconst01_to_address.street;

//set street 1 to street 2
forms.contact_main.gconst01_to_address.street =
forms.contact_main.gconst02_to_address.street;

//set street 2 to the global value
forms.contact_main.gconst02_to_address.street = globals.gTempText;

//repeat for other fields
globals.gTempText = forms.contact_main.gconst01_to_address.city;
forms.contact_main.gconst01_to_address.city =
forms.contact_main.gconst02_to_address.city;
forms.contact_main.gconst02_to_address.city = globals.gTempText;

globals.gTempText = forms.contact_main.gconst01_to_address.state_province;
forms.contact_main.gconst01_to_address.state_province =
forms.contact_main.gconst02_to_address.state_province;
forms.contact_main.gconst02_to_address.state_province = globals.gTempText;

globals.gTempText = forms.contact_main.gconst01_to_address.postal_code;
forms.contact_main.gconst01_to_address.postal_code =
forms.contact_main.gconst02_to_address.postal_code;
forms.contact_main.gconst02_to_address.postal_code = globals.gTempText;
```

```
globals.gTempText = forms.contact_main.gconst01_to_address.address_type;
forms.contact_main.gconst01_to_address.address_type =
forms.contact_main.gconst02_to_address.address_type;
forms.contact_main.gconst02_to_address.address_type = globals.gTempText;

currentcontroller.saveData();
```

In this method – we’re simply storing the data in the related main address into our global `gTempText`; then we’re setting the value of the related main street to the value of the related second address; and finally setting the related second address to the data stored in the global. We need to do the same thing for each of the fields on the form.

The final step `currentcontroller.saveData()` will then save the data, and have the effect of exiting the fields.

You don’t have to type all of the code above – in fact, I only typed the comments and the equal signs in the code. Simply click on the `contact_main` form and click the “+” next to the `relations` item. Click on `gconst01_to_address` and you’ll see the list of fields. Double-click the field to add it to your form. You can use this technique of alternating between `gconst01_to_address` and `gconst02_to_address` to create the method.

The last thing we’re going to do in this chapter is to hook up the global method to our buttons and to set the `onShow` property of the address forms to show/hide the switch address (“Swap with”) button.

Switch to the `tab_address_main` form from the “Window” menu – and double-click the empty space next to the `onShow` property in the Properties panel. Choose the form method `on_show`. Next, click on the label object (not the rectangle – but the object on top of the rounded rectangle) and set the `onAction` property to the global method `btn_switch_address`.

Now switch to the form `tab_address_second` and do the same as above – set the `onShow` method for the form and the `onAction` method for the button.

When you’re done – switch to the `contact_main` form and try it out. You can create a new record, fill in a Main Address and a Second Address, then click the “Swap with” button and they should reverse. Click it again (or click the button on the other tab) and the addresses will switch back.

In the next section, we’ll create the list view, table view, and the help forms - and then hook them up to the interface with some new (simple) methods.

Chapter 9 – Building the User Interface – Part 5

Estimated Time To Complete: 1 hour

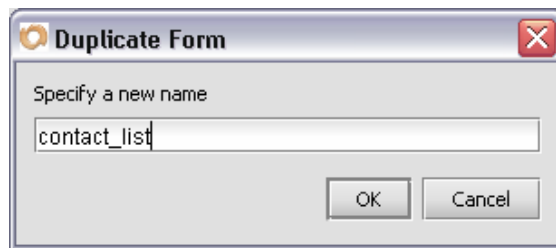
Now we’ll create the list view, the table view and hook up the navigation. Open the Servoy solution (if it’s not already open), navigate to the `contact_main` form and go into the Designer Mode – press CTRL-L (PC) or COMMAND-L (Mac); or choose “Designer” from the “Tools” menu.

Our list view and table view will use the top navigation – so rather than going through the hassle of re-creating them – we’ll simply duplicate the form and modify it to suit our needs.

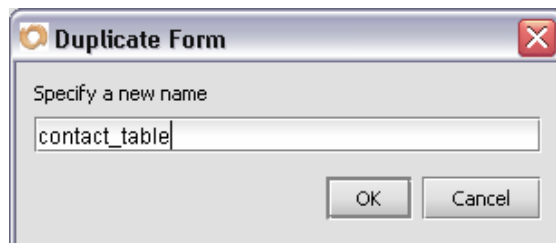
BEFORE WE DUPLICATE the contact_main form – let’s make sure that we’ve set things up properly. Click on the first_name field and MAKE SURE that the name property reads: first_name and the TEXT property reads: If it doesn’t, fix it now! Same for each of the other fields:

Field	Name Property	Text Property
first_name	first_name	First
last_Name	last_name	Last
title	title	Title
company	company	Company
gconst01_to_phone.phone_data	phone_1	Phone 1
gconst02_to_phone.phone_data	phone_2	Phone 2
email	email	Email
notes	notes	Notes
image_data	image_data	Image

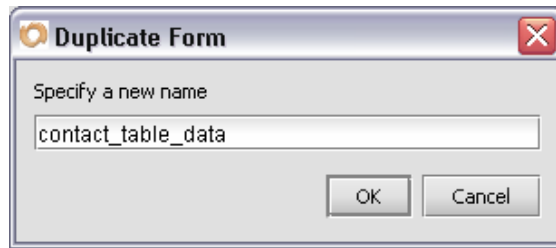
Once you’ve set all the name and text properties - choose “Duplicate Form” from the “File” menu – and name this form `contact_list`:



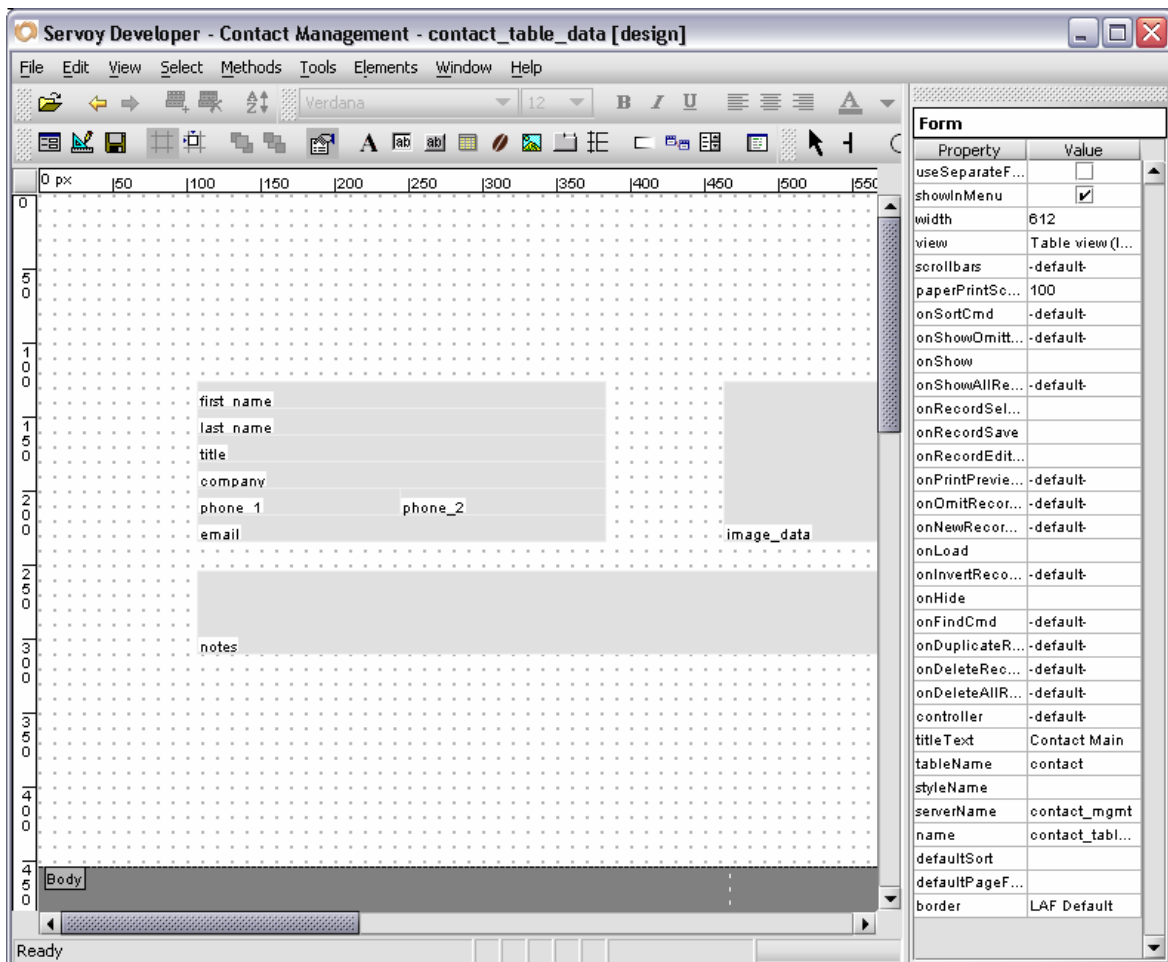
Once you click the “OK” button – you’ll have a duplicate form – all ready to go. Before we change anything – we’ll duplicate this form again for our table view. Choose “Duplicate Form” again from the “File” menu – and call this form `contact_table`:



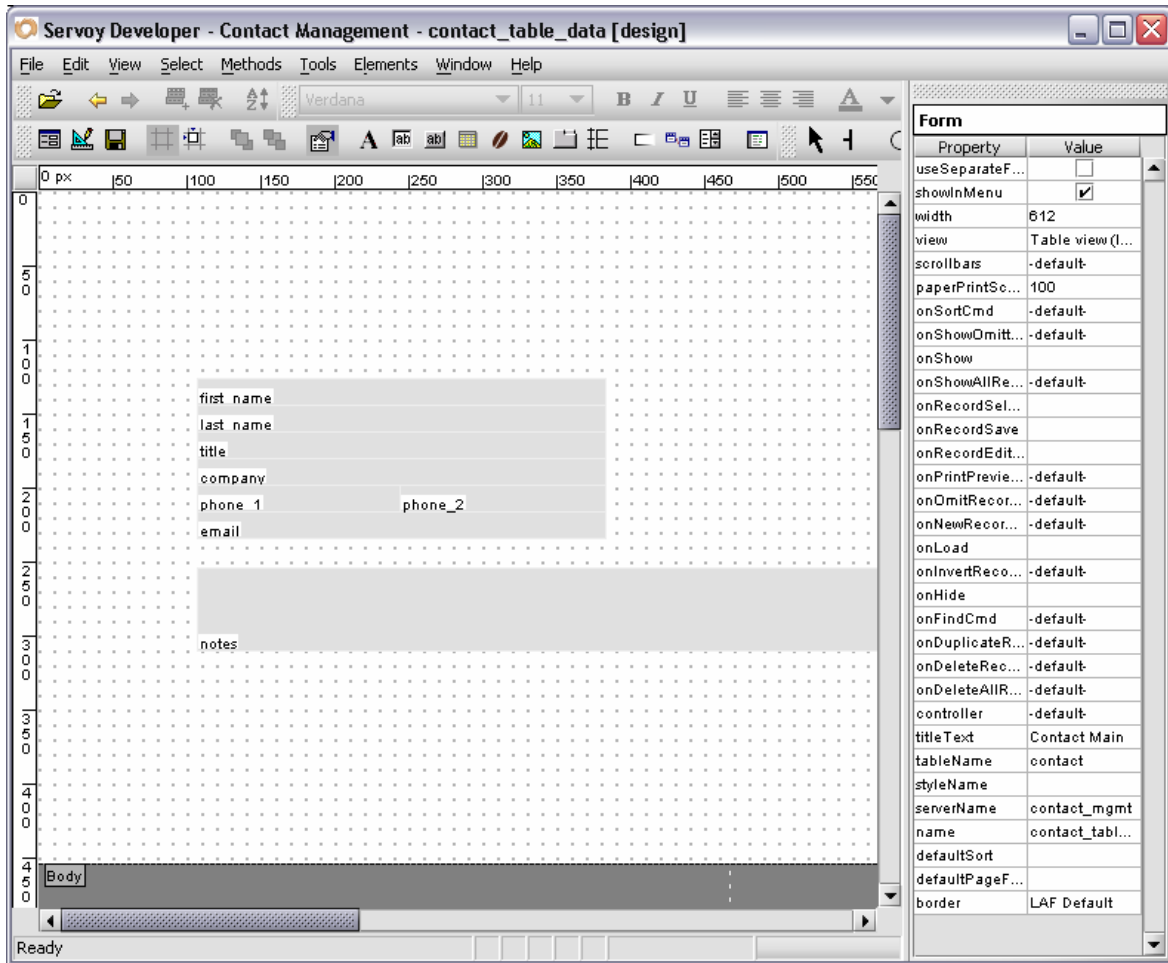
Choose "Duplicate Form" again from the "File" menu – and call this form `contact_table_data`:



We'll do the `table_view_data` first – since it's the easiest. Delete EVERYTHING on the form – except for the actual fields themselves:



You can then delete the image_data field as well:

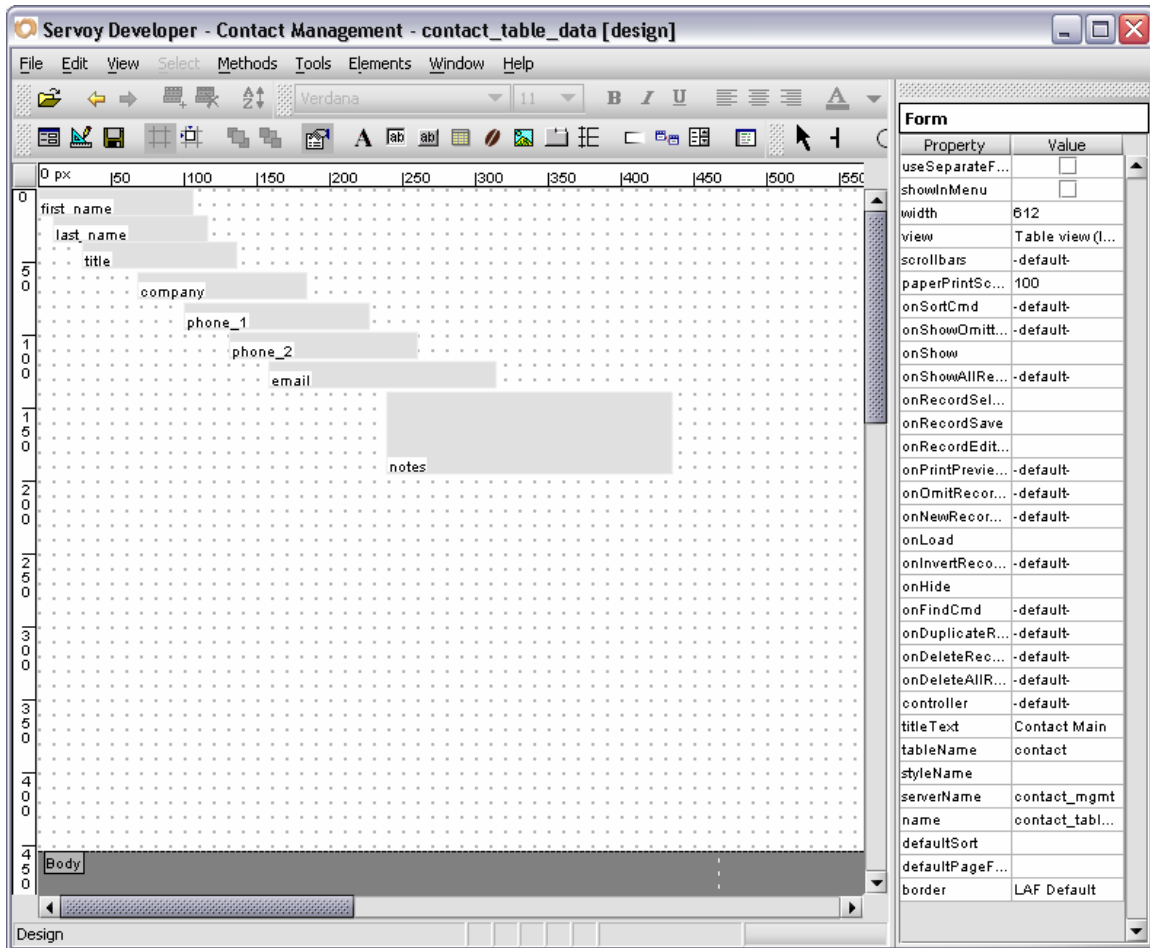


Click on the white part (the Body) of the form, and change the following properties:

Object Type	Property	Value
Contact Table	ShowInMenu	FALSE (UNCHECK checkbox)
	Scrollbars	Vertical: never, Horizontal: never
	View	Table view (locked)
	TitleText	Table View Data

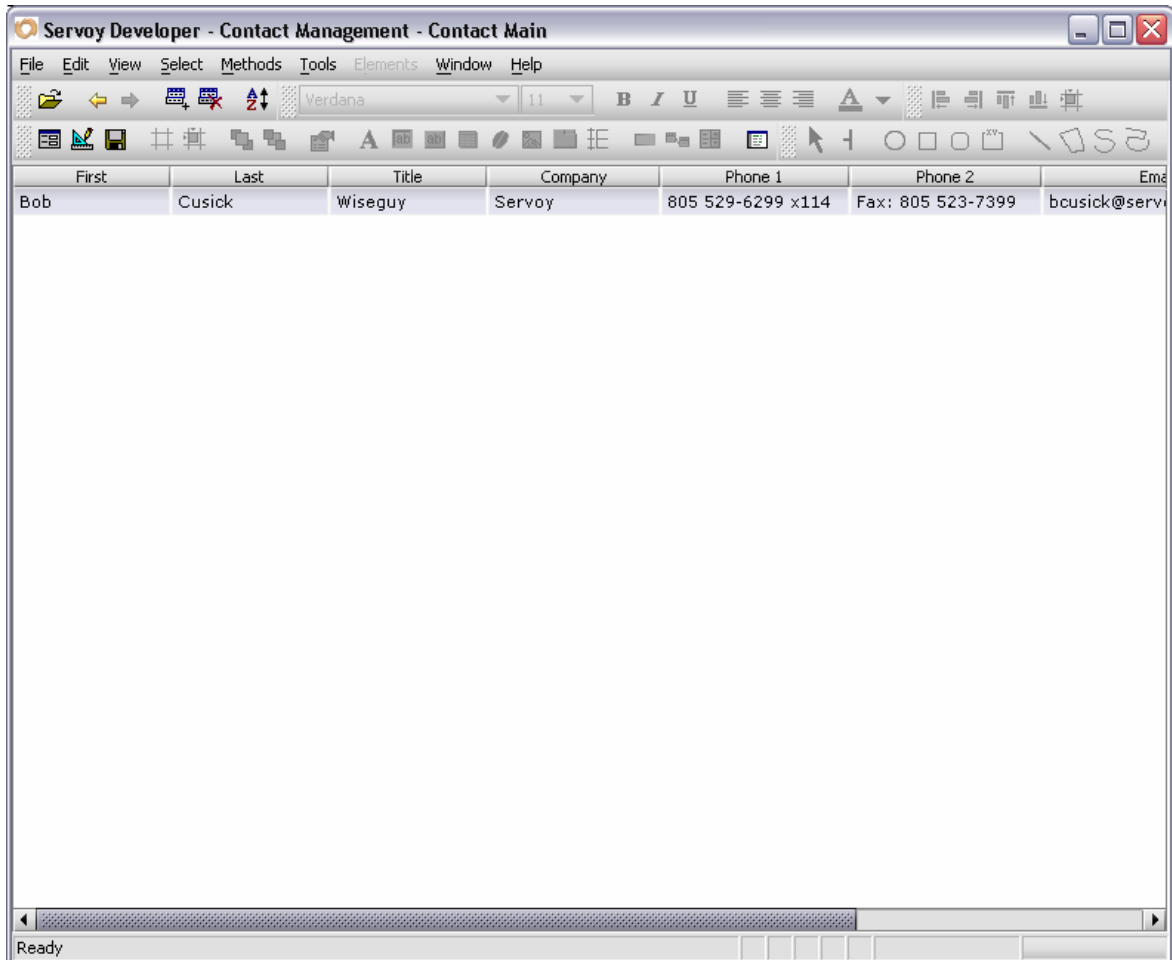
You should arrange the fields in the order you want them appear. Servoy will try to “draw” the table view based on the physical layout of the columns (from left to right, top to bottom) and will make the default width of the column the width of the field itself.

Let's re-arrange the fields and make them shorter, so we can see more of them in Data mode:



When you come out of Designer mode (into Data mode) – you can see that we created a table that has labels of each column based on the TEXT property of each field:

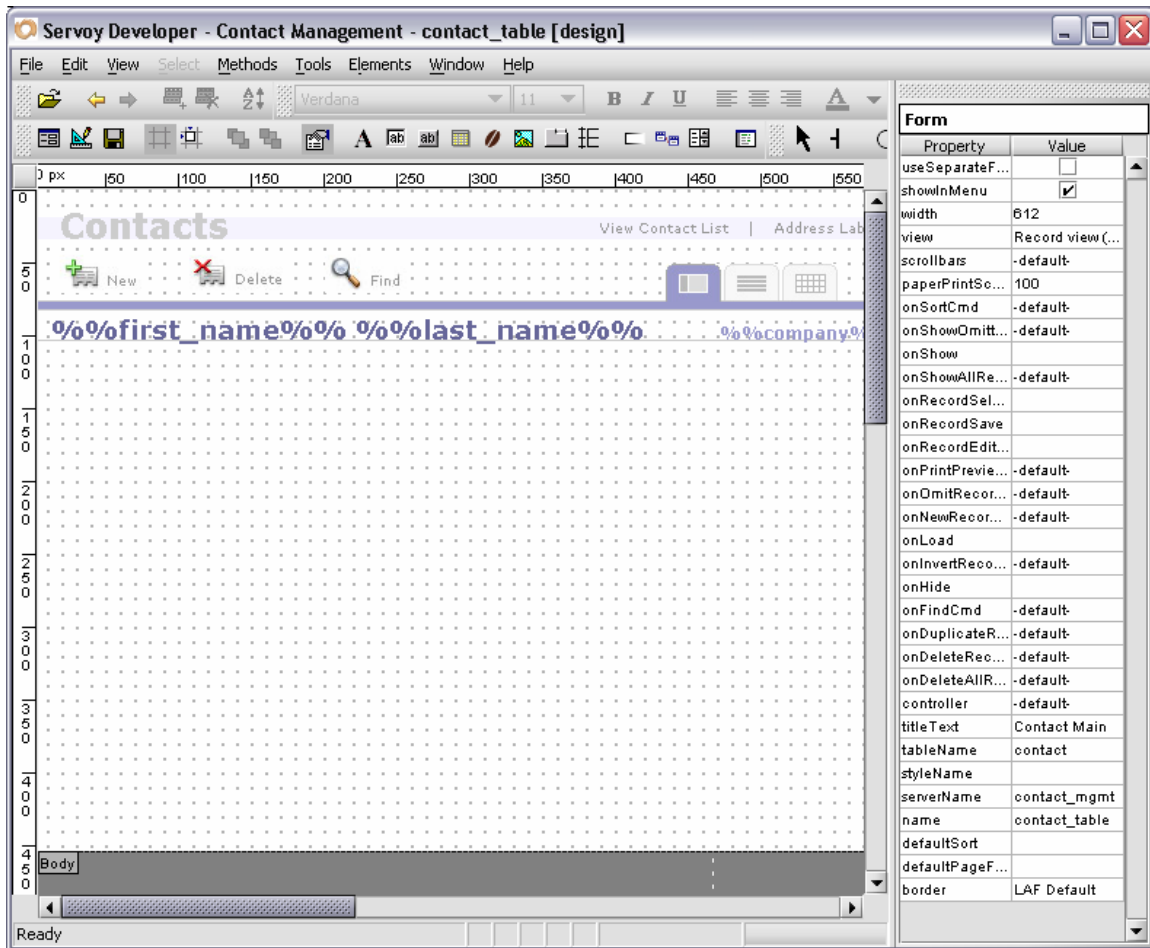
Notice that the order of the fields in the table view is the same as the fields you just arranged in Designer mode for this form.



You can resize and re-order the columns, and even edit the data. If you're not happy with the order of the fields – simply re-order them (left to right) and the default order will change. Each user can move the fields into a different order (that lasts as long as they have the Client open, and then defaults back to the order you designed next time the Client is opened); and they can click the header to sort the data – all without any programming on your part. Cool!

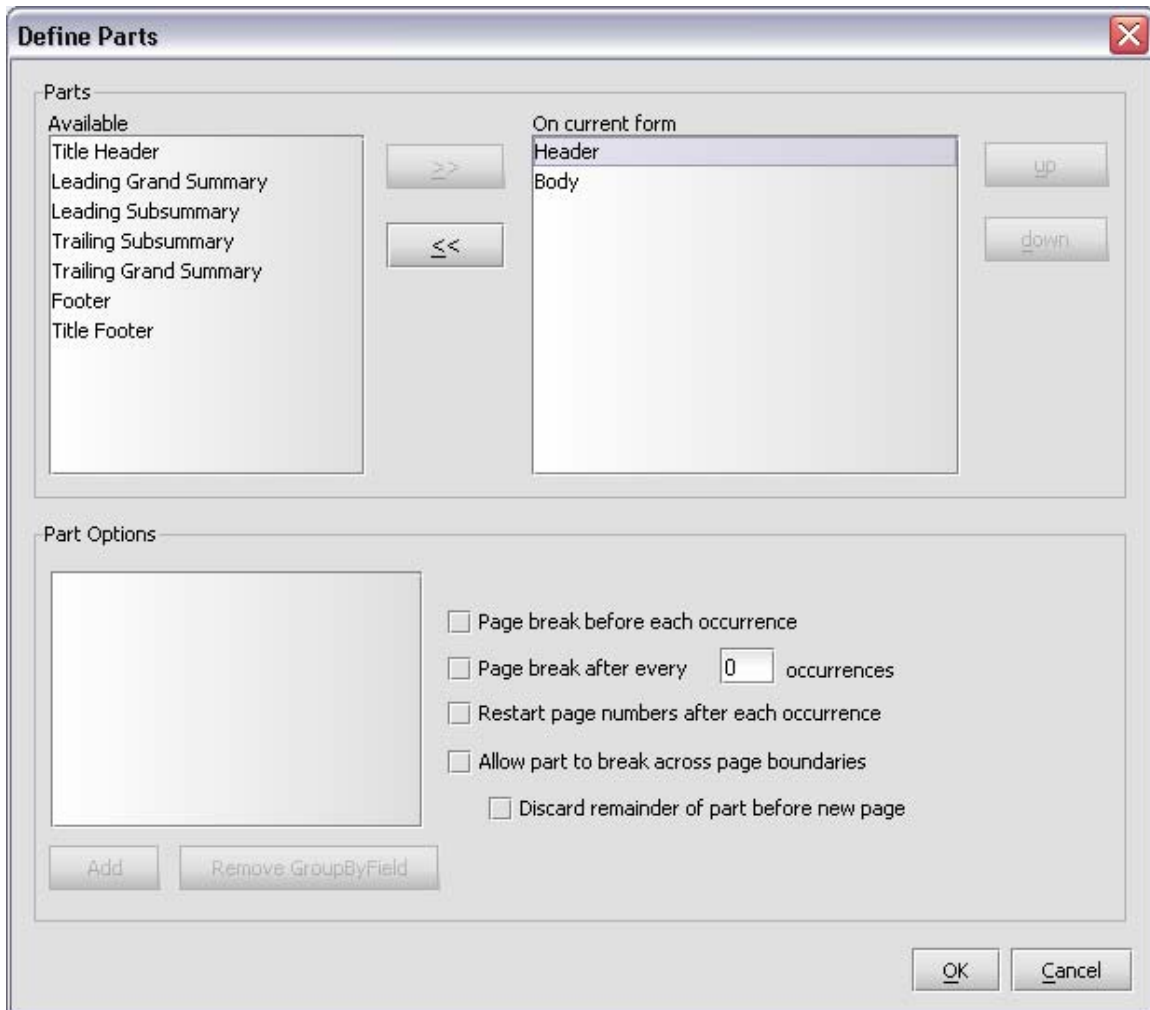
Now we'll show this table view on our `contact_table` form using a tabpanel – but first, we have to prepare the `contact_table` form.

Navigate to the `contact_table` form by choosing it from the "Window" menu and then delete EVERYTHING below the `%%first_name%% %%last_name%%` label at the top:



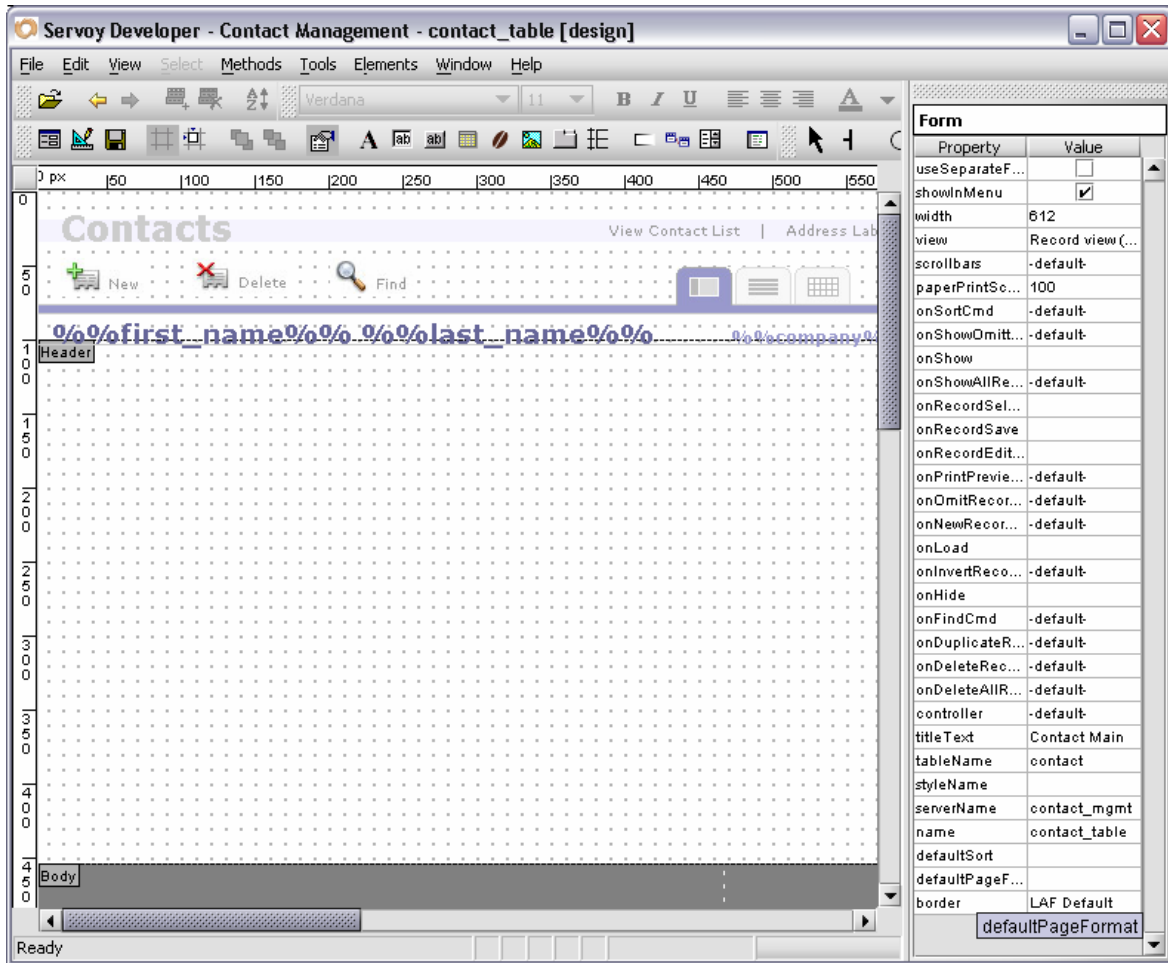
Now we need to add a header part so our buttons on the top will show up. Choose "Define Parts" from the "Elements" menu or select the Parts tool in the toolbar.

Click on the "Header" part in the "Available" list on the left and then click on the ">>" move button:



Click the "OK" button.

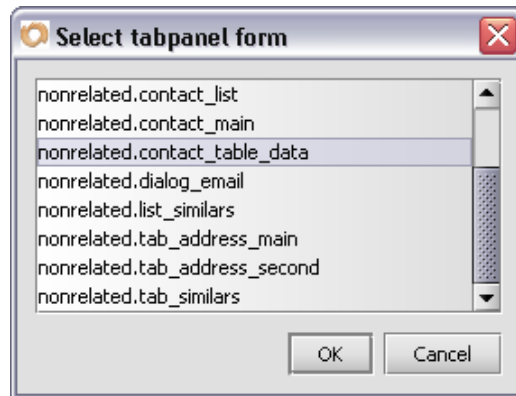
You'll see that a header part has been added to your form:



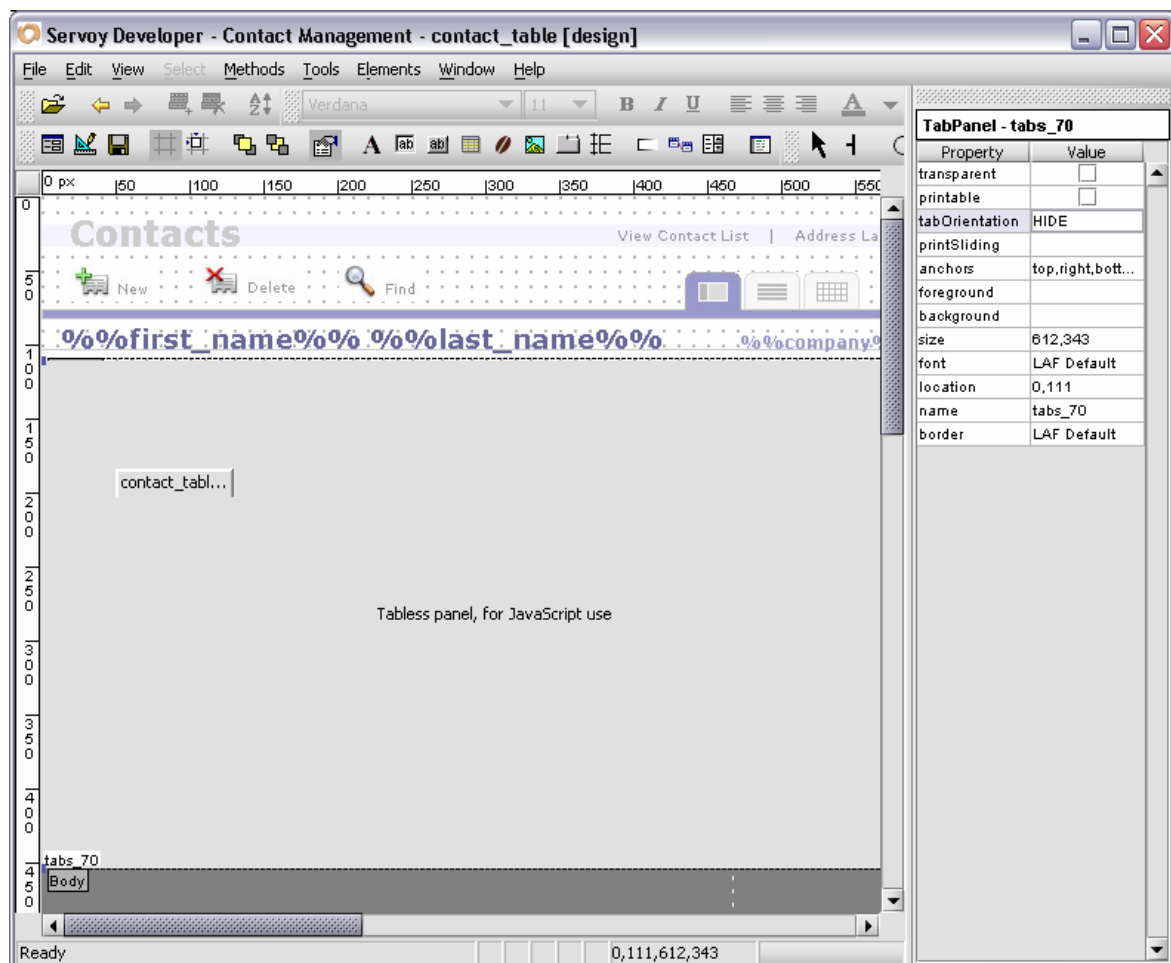
Click on the gray label of the "Header" part and set the following properties:

Object Type	Property	Value
Header Part	Height	109
	Background	Color: Red = 255, Blue = 255, Green = 255

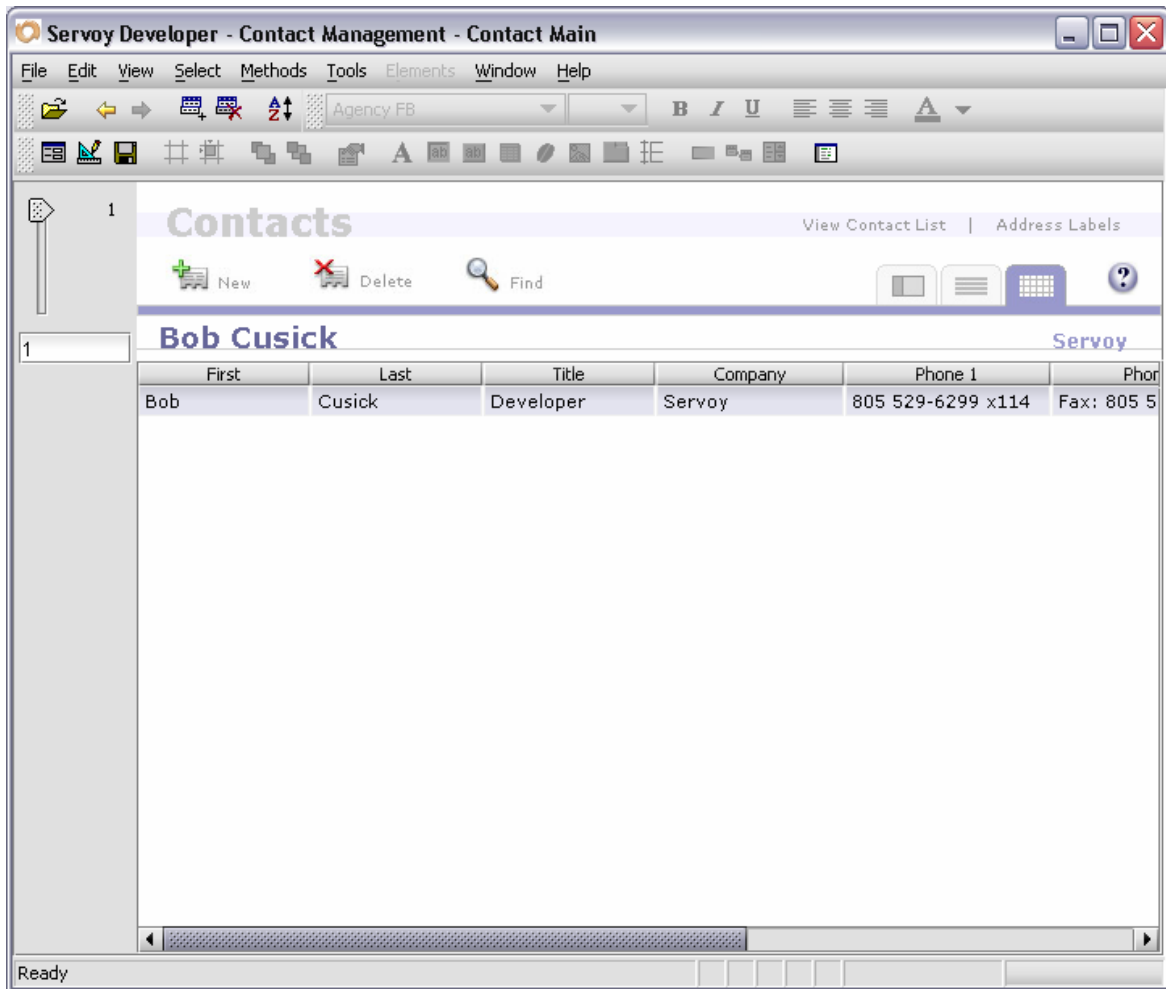
Now we can add the tabpanel to display our table view. Click on the Tabpanel tool in the toolbar or choose "Place Tabpanel" in the "Elements" menu. From the list of available forms, choose `nonrelated.contact_table_data`:



Click "OK", and then place the tabpanel just below the Header part – make the tabpanel the same height and width as the rest of the form:

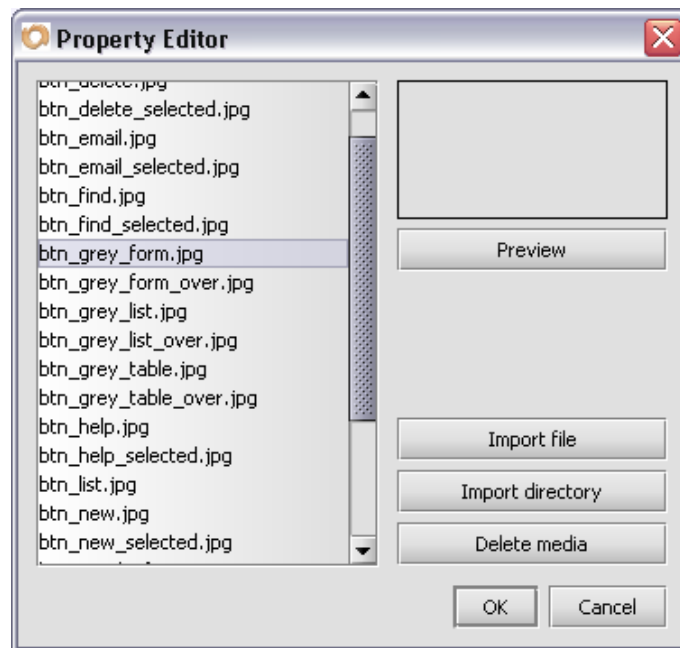


With the tabpanel still selected, set the `tabOrientation` property to `HIDE`; set the `location` to 0,111; set the `size` to 612,343 and set the `anchors` property to `top, right, bottom, left`. Exit the Designer mode and you'll see your table view appear:

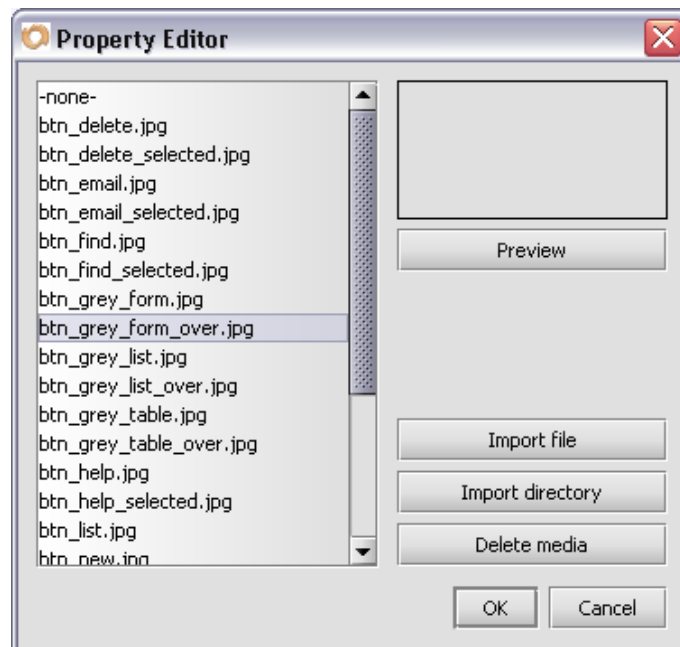


We need to clean up a couple of the graphics and then we'll be done with the table view. Go back into Designer mode and click on the purple form view button.

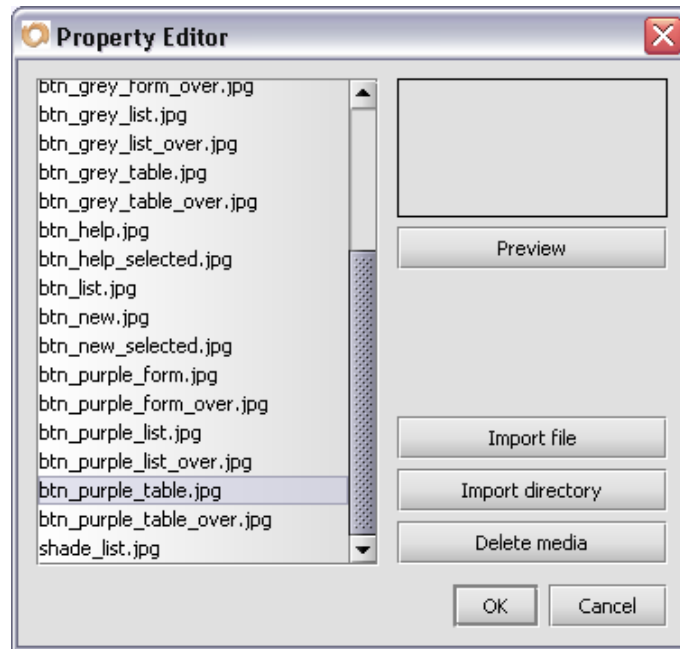
Change the `imageMedia` property to `btn_grey_form.jpg`:



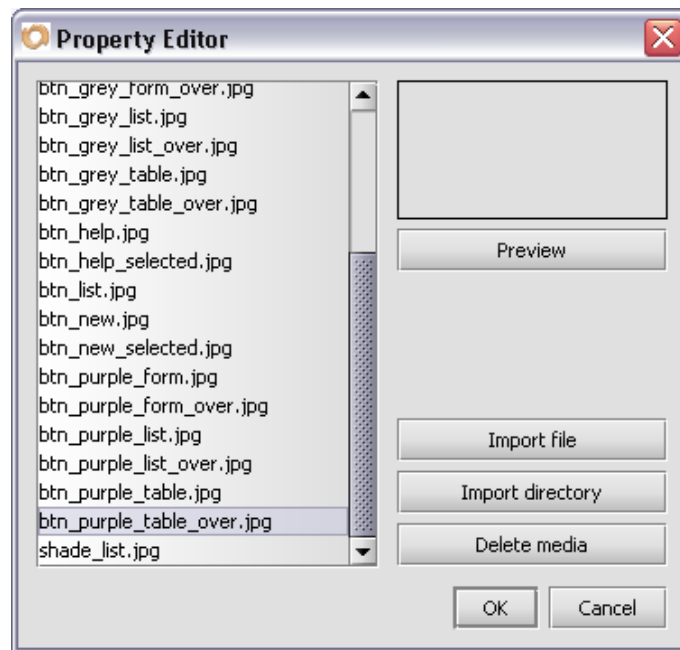
Change the `rolloverImageMedia` to `btn_grey_form_over.jpg`:



Next, click on the grey table view button (the furthest on the right – to the left of the Help button) and change the `imageMedia` property to `btn_purple_table.jpg`:



Change the `rolloverImageMedia` to `btn_purple_table_over.jpg`:



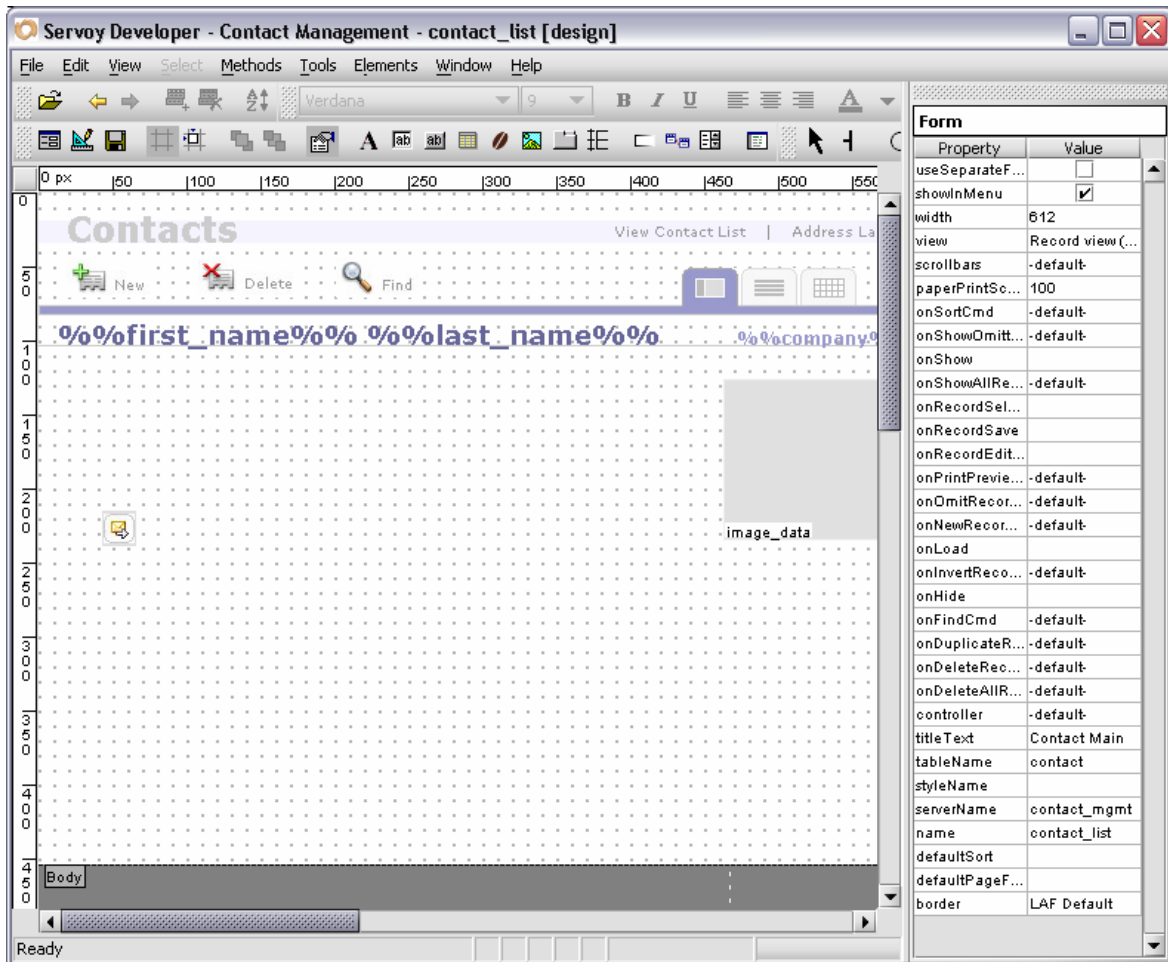
Your form should look like this (in Data Mode):

The screenshot shows the Servoy Developer interface for a 'Contact Management' application. The window title is 'Servoy Developer - Contact Management - Contact Main'. The menu bar includes File, Edit, View, Select, Methods, Tools, Elements, Window, and Help. The toolbar contains various icons for file operations, editing, and viewing. The main area displays a form titled 'Contacts' with a subtitle 'View Contact List | Address Labels'. Below the title, there are buttons for 'New', 'Delete', and 'Find'. The form is currently in 'Data Mode', showing a table with contact information. The table has columns for First, Last, Title, Company, Phone 1, and Phone 2. The data row shows 'Bob' as the first name, 'Cusick' as the last name, 'Developer' as the title, 'Servoy' as the company, and '805 529-6299 x114' as the phone number. The status bar at the bottom indicates 'Ready'.

First	Last	Title	Company	Phone 1	Phone 2
Bob	Cusick	Developer	Servoy	805 529-6299 x114	Fax: 805 5

Now that we have the table view done – let's work on the list view. Switch to the contact_list form via the "Windows" menu and enter the Designer mode.

This time, we're going to delete everything below the `%%first_name%% %%last_name%%` label – EXCEPT the email button and the `image_data` field:

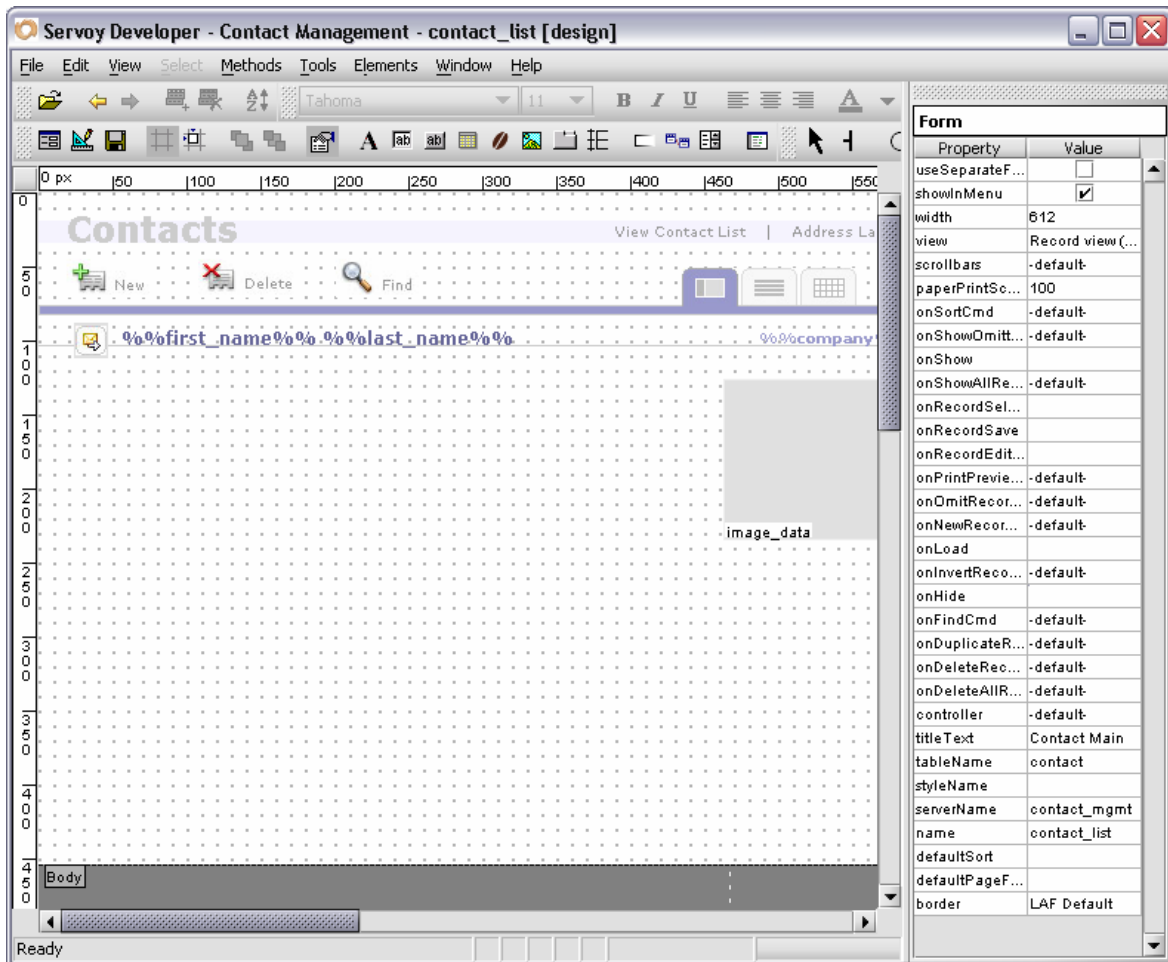


Next, we'll change the font size of the merge text labels and move the email icon up to the top.

Click on the `%%first_name%% %%last_name%%` label and change the font to Verdana, 12pt, bold; change the size to 320,20; and the location to 56,88.

Click on the `%%company%%` label and change the font to Verdana, 10pt, bold; change the location to 371,88.

Click on the email button icon and change the location to 24,88.



We're now going to add a couple of other merge labels to the form.

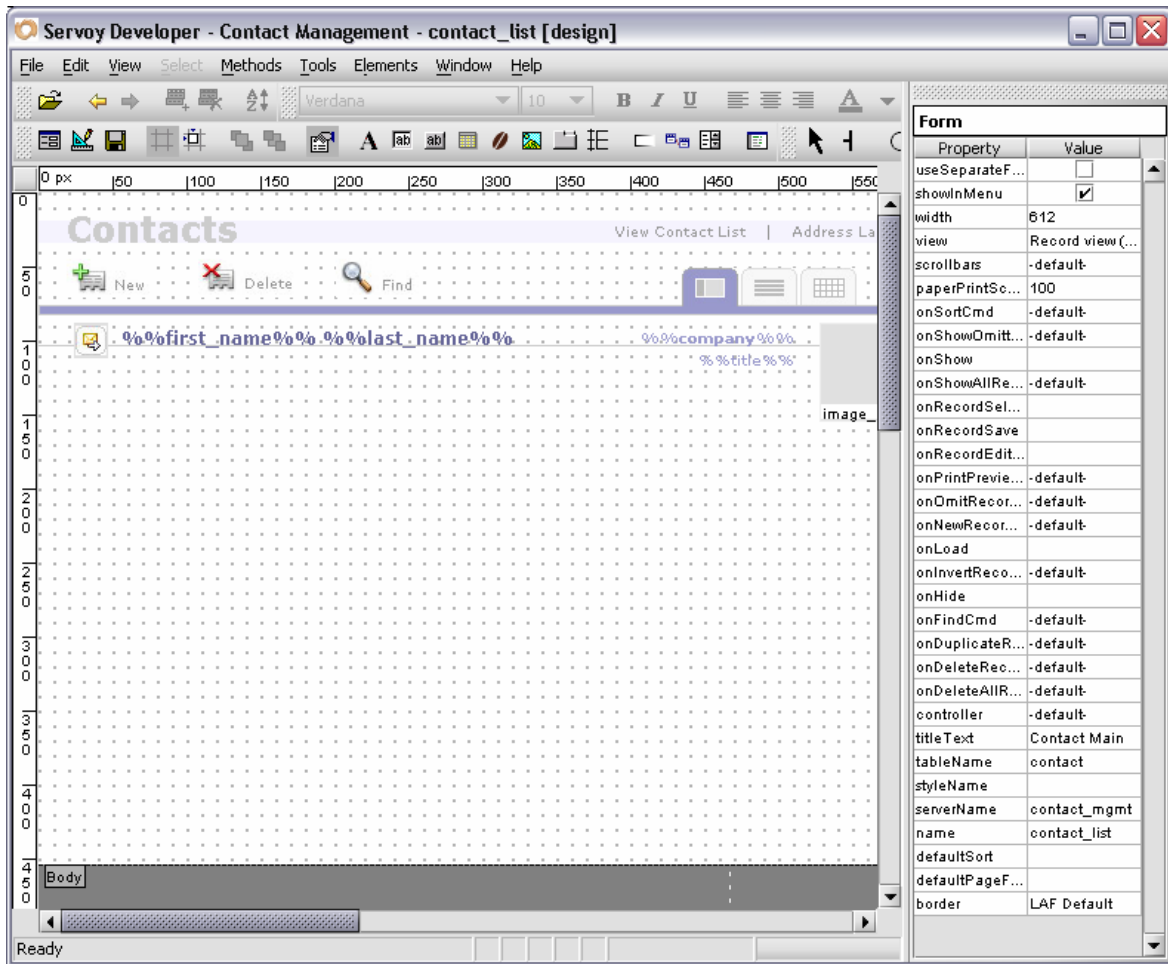
Click on the `%%company%%` label and copy/paste. WITHOUT DESELECTING THE OBJECT, hold down the CTRL key and press the DOWN arrow key once, and the LEFT arrow key once. This will move the object in 10 pixel increments down and to the left.

Now let go of the CTRL key and press the UP arrow key FIVE times (you can also just set the location to 371,103).

Double-click the text property and change it to `%%title%%`; and change the font to Verdana, 10pt, regular and the horizontalAlignment to RIGHT.

Hold down the CTRL key (PC) or SHIFT key (Mac) and select both the `%%company%%` label and the `%%title%%` label. With the CTRL key down – click the LEFT arrow key 8 times (or set the location of the `%%company%%` label to 291,88 and the location of the `%%title%%` label to 291,103).

Click on the `image_data` field and set the size to 78,76 and the location to 524,88:

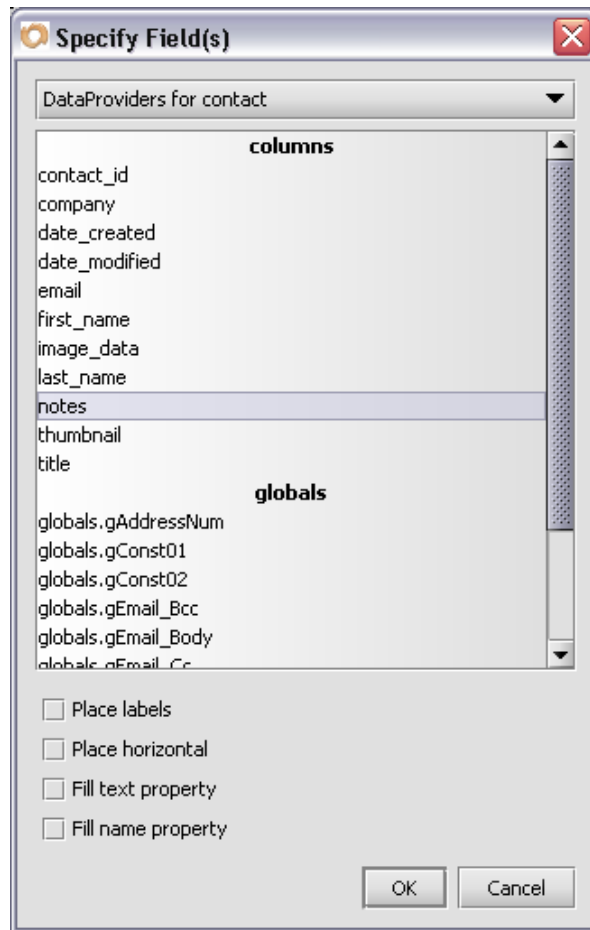


Now click on the %%first_name%% %%last_name%% label and copy/paste. Double-click the text property and change it to %gconst01_to_phone.phone_data%. Set the location to 56,103; and set the foreground color to: Red=102, Green=102, Blue=153.

We also want to show the notes field below the phone number – and since the field can hold multiple lines of data, we have to make a decision on what kind of object we’re going to use to display the field. If we use a label object – we would have to create a calculation that would return HTML with
 in place of the line returns – since labels can only show a single line of data without being formatted as HTML.

However, we can place a field on the form – and set it to non-editable – and not have to create the calculation. In this case – we’ll choose the field option.

Click the "Place Field" tool in the toolbar or choose "Place Field" from the "Elements" menu; and choose Notes from the list (UNCHECK "Place labels"):

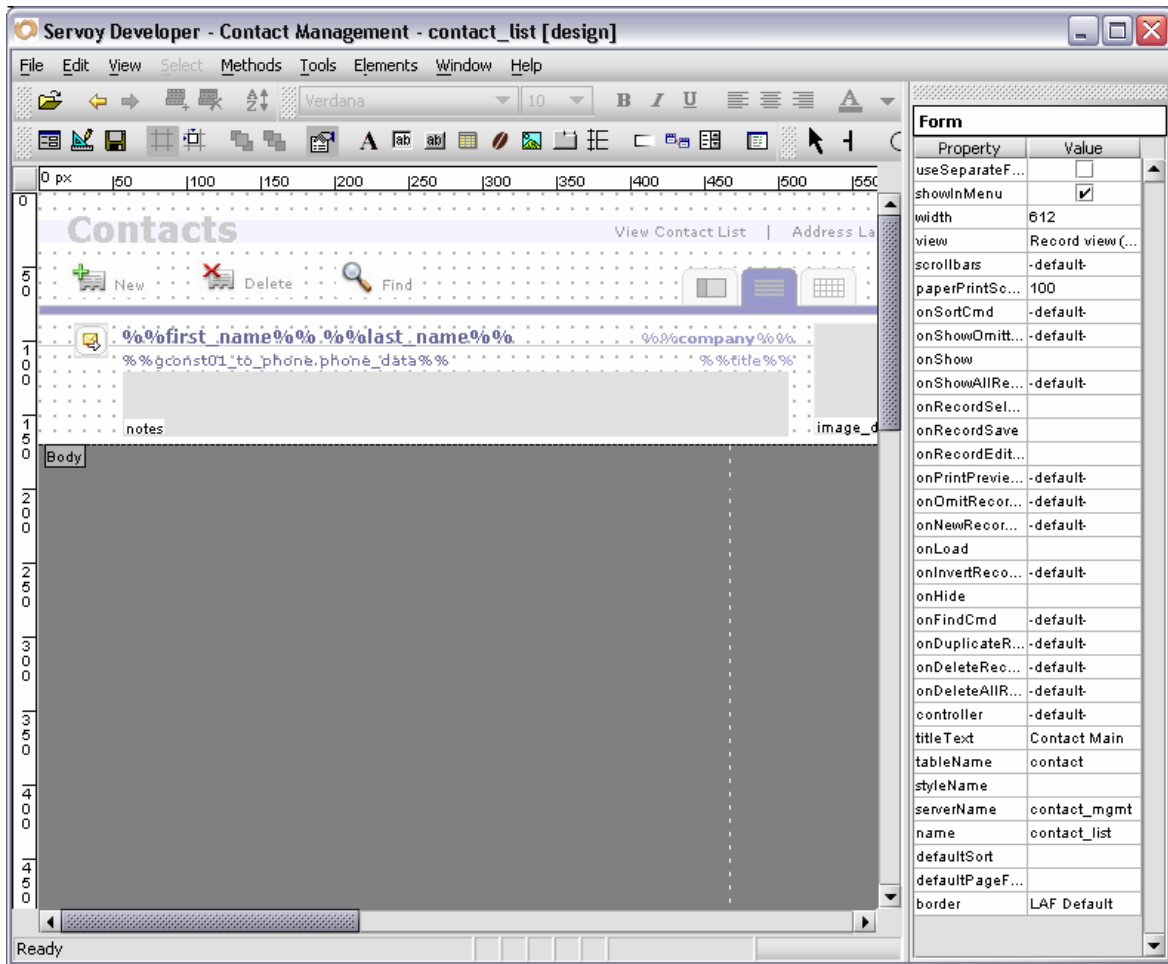


Click on the newly placed field and CHECK the `transparent` property; UNCHECK the `editable` property; set the font to Verdana, 10pt, Regular; set the location to 56,121; set the size to 450,44; set the foreground color to: Red=153, Green=153, Blue=153; and set the border to EMPTY.

Click on the gray label of the "Body" part and set the `height` property to 170.

Next we'll change the navigational graphics on this form. Click on the purple form button and do what you did on the `contact_table` form:

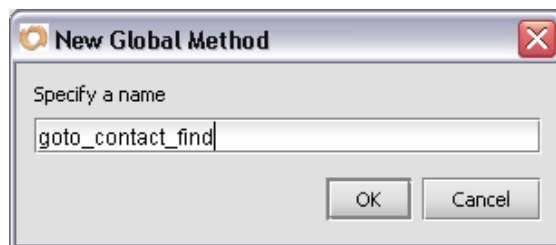
Change the `imageMedia` property to `btn_grey_form.jpg`; and the `rolloverIageMedia` property to `btn_grey_form_over.jpg`. Click on the list navigation button (the second one) and change the `imageMedia` property to `btn_purple_list.jpg`; and the `rolloverIageMedia` property to `btn_purple_list_over.jpg`.



We have to do one more thing before we're finished with this list view form.

Because we're using merge fields to display the data (and not "field" objects) – if we click on the "Find" button – the only field we can search by is the "notes" field. So, we're going to make a new method that will take us to the `contact_main` form and then perform the `btn_find` global method. Click on the "Find" button and double-click the `onAction` property. The button is now hooked up to the global method `btn_find`.

Click the "New global method" button and name the new method `btn_goto_contact_find`.



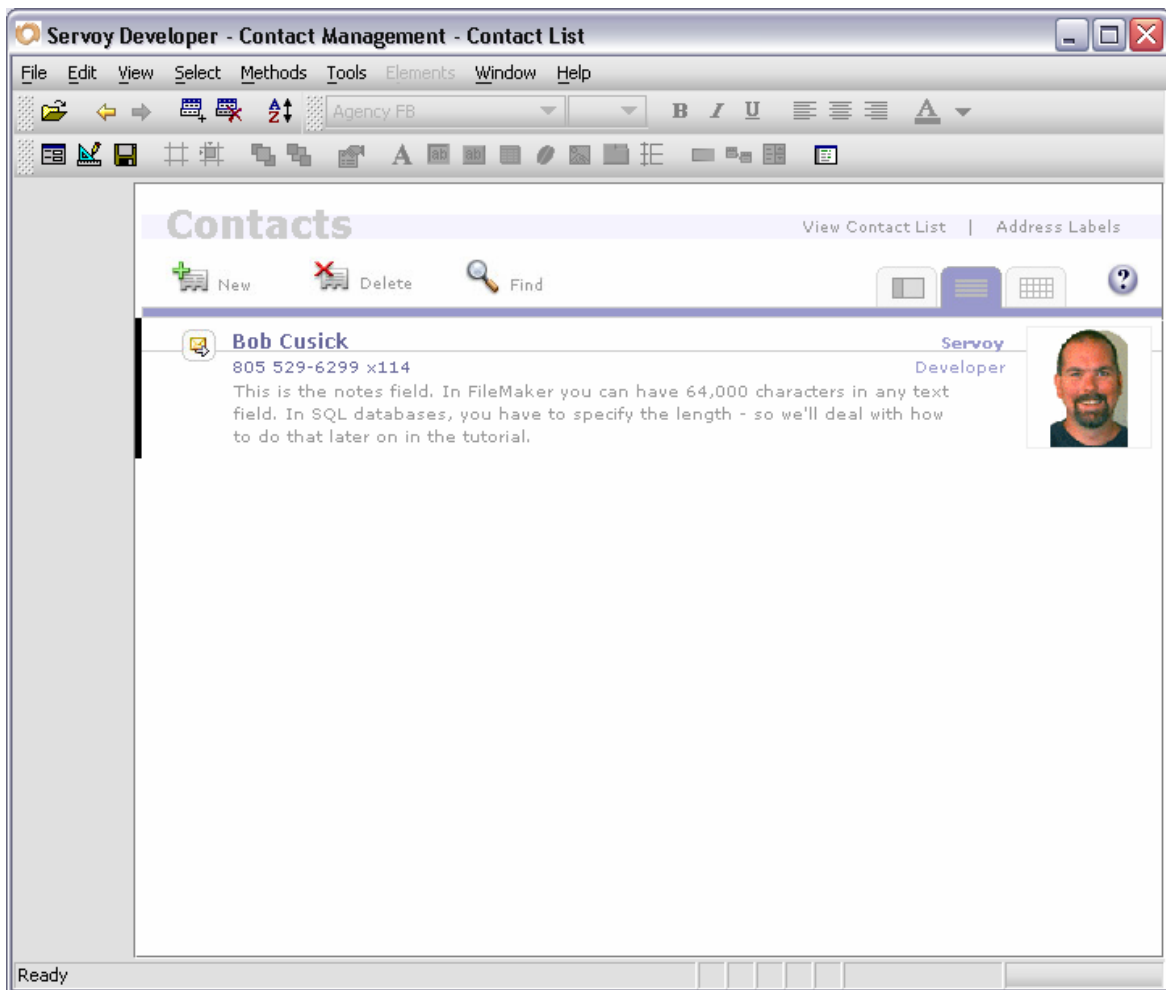
Open the Method Editor by choosing “Methods...” from the “Tools” menu.

Here’s the code for the `btn_goto_contact_find` method:

```
//show the contact_main form
forms.contact_main.controller.show();

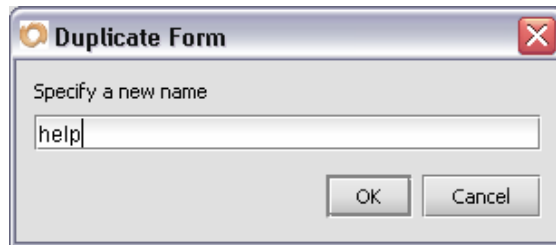
//perform the global btn_find method
globals.btn_find();
```

Don’t forget to click the “Verify” button in the lower right of the Method Editor dialog. When you exit Designer mode (into Data mode) you should see something like this:

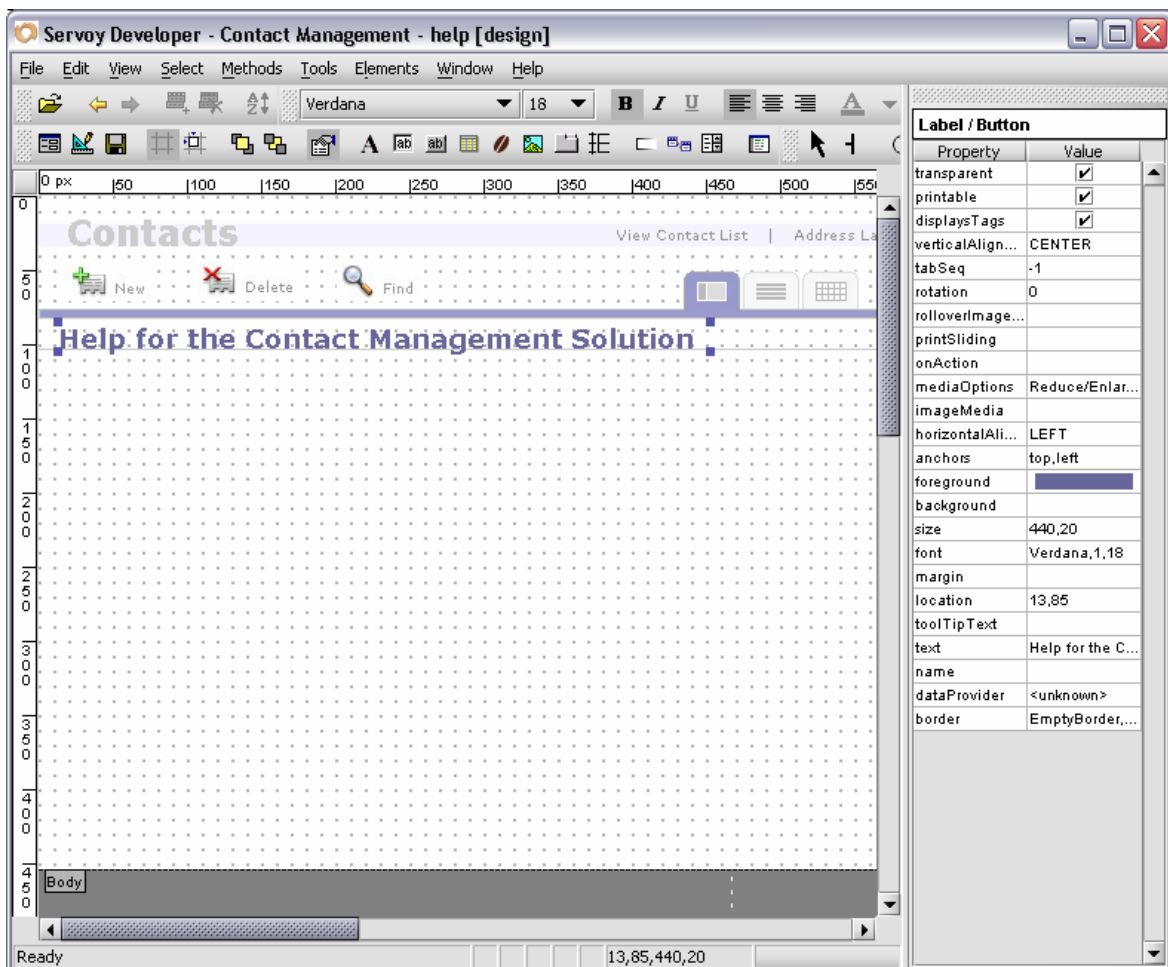


Now we’ll create the “Help” form. We’ll start by navigating to the `contact_main` form via the “Window” menu.

Choose "Duplicate Form" from the "File" menu and call the new form "help".

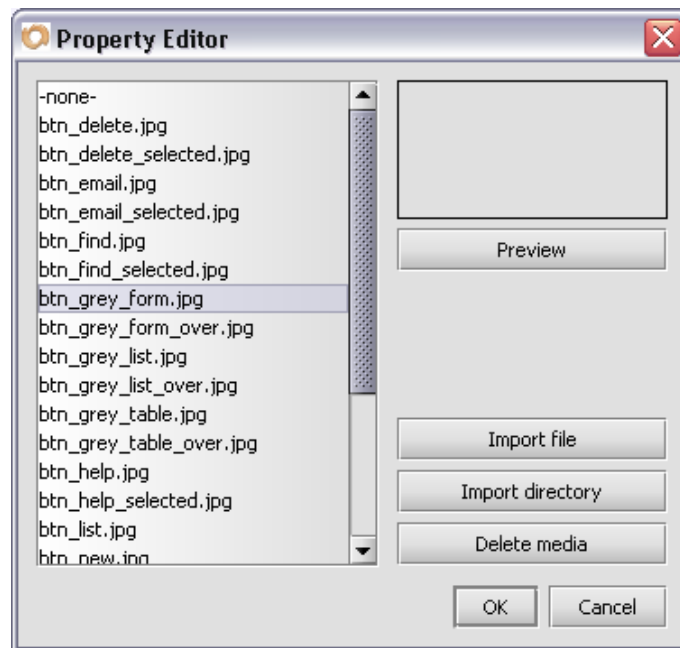


DELETE everything below the %%first_name%% %%last_name%% label. Delete the %%company%% label. Double-click the %%first_name%% %%last_name%% label and change the text to "Help for the Contact Management Solution" (you'll need to change the size property of the label to 440,20 for it all to show up).

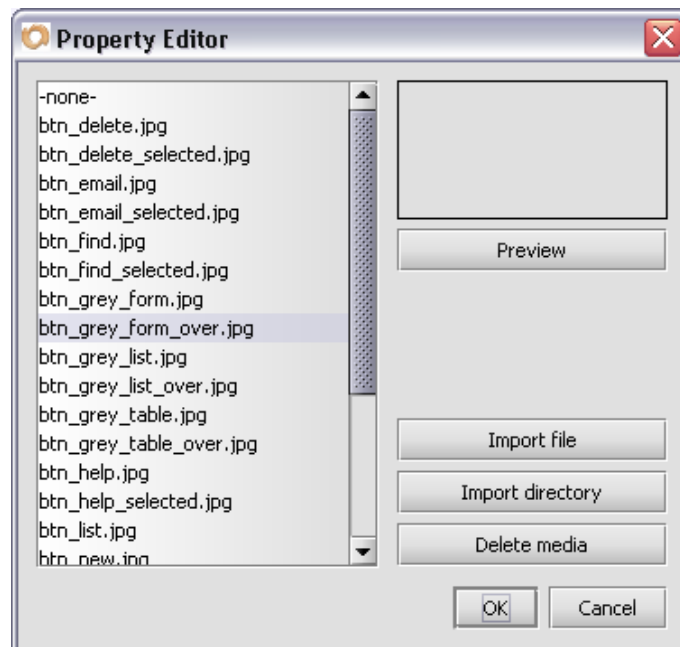


Delete the New, Delete and Find buttons at the top and change the purple forms button:

Set the imageMedia to btn_grey_form.jpg



Set the rolloverImageMedia to btn_grey_form_over.jpg



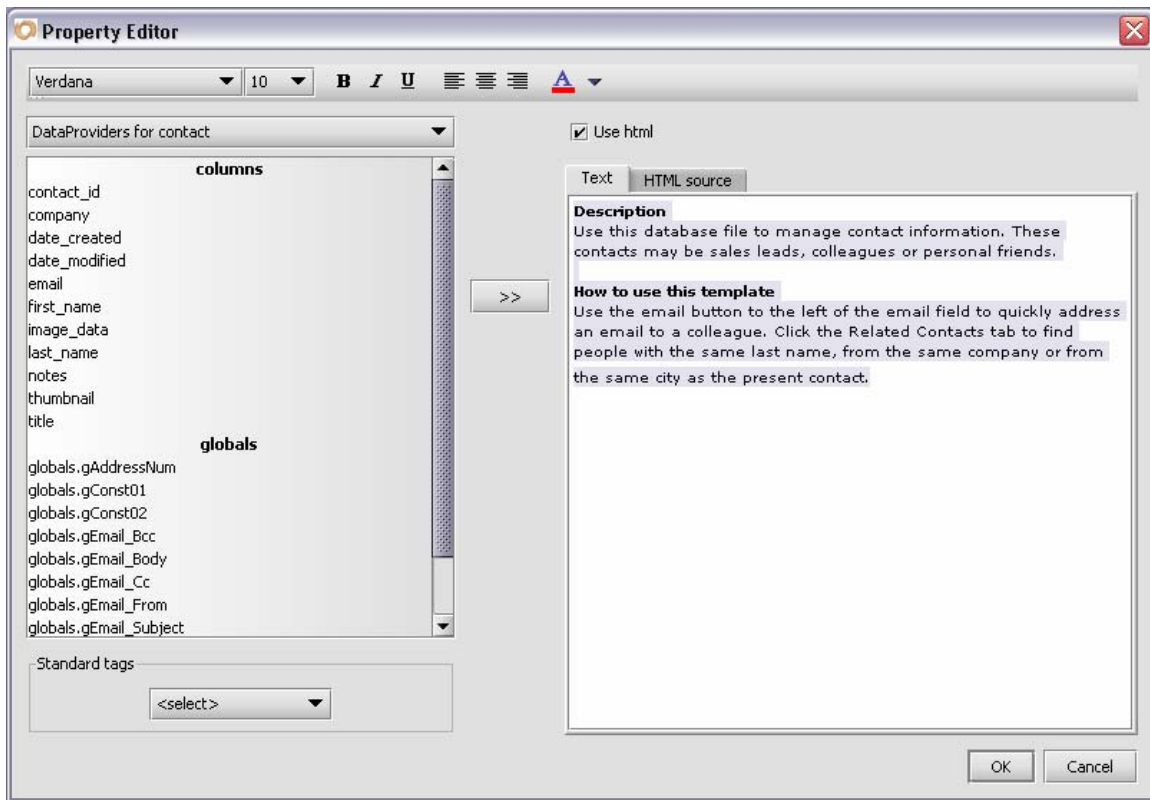
Set the method attached to the `onAction` property to the global method `goto_form_view`.



You can now do a couple of different things with regard to the help text. You can create a label with the instructions; or create a global variable – when the solution loads, you can set the contents of the global to the instructions, etc.

In this case, we're going to put the text into a label. Place a new label on the form by clicking on the "Place Label" tool in the toolbar or choosing "Place Label" from the "Elements" menu. Set the `verticalAlignment` to `TOP`, the `horizontalAlignment` to `LEFT`, the `size` to `586, 329` and the `location` to `14, 113`. Then double-click the `text` property.

Enter or paste the text you want to use for the help – and then click the “Use html” checkbox. Select all the text – CTRL-A (PC) or COMMAND-A (Mac) – choose Verdana 10pt, and format the text as you want.

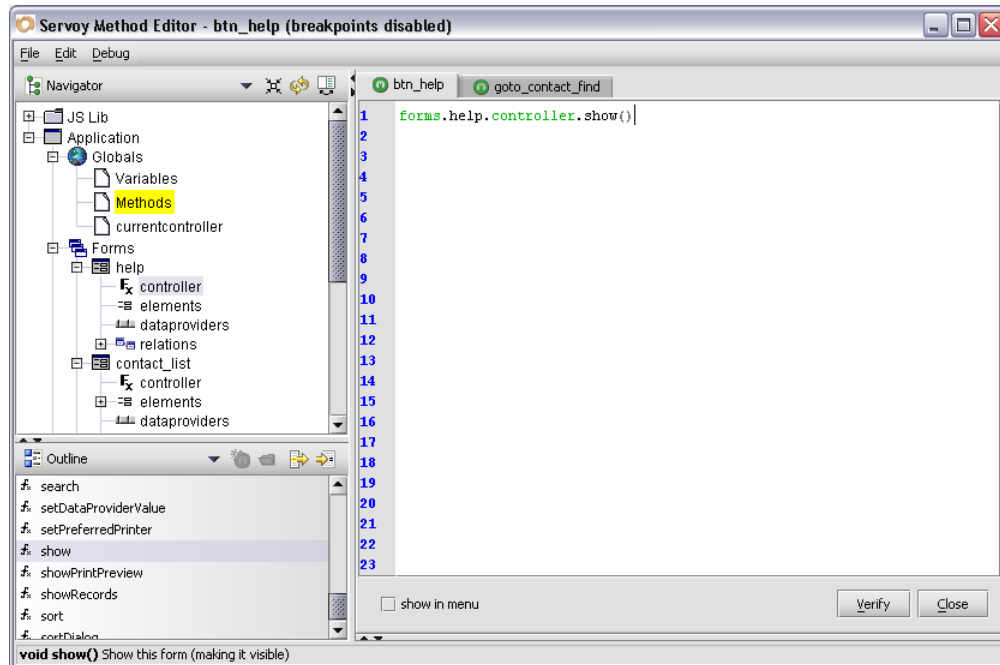


Then click the “OK” button. Now all that’s left to do is to add some code to the global method `btn_help` that we created earlier.

Open the Method Editor and double-click on the global method `btn_help`. Click the “+” next to the `help` form – and double-click the `show` function from the controller object.

Your code should look like this:

```
forms.help.controller.show()
```



Click the “Verify” button in the lower right.

WHEW! We’re done with the layout of all the forms. In the final steps – we’ll hook up the navigation buttons and then we’re FINISHED!

Navigate back to the contact_main screen via the “Windows” menu and click on the grey form button (to the left of the purple list button) – double-click on the `onAction` property and choose the global method `goto_form_view`.

Click on the grey table button (to the right of the purple list button) – double-click on the `onAction` property and choose the global method `goto_table_view`.

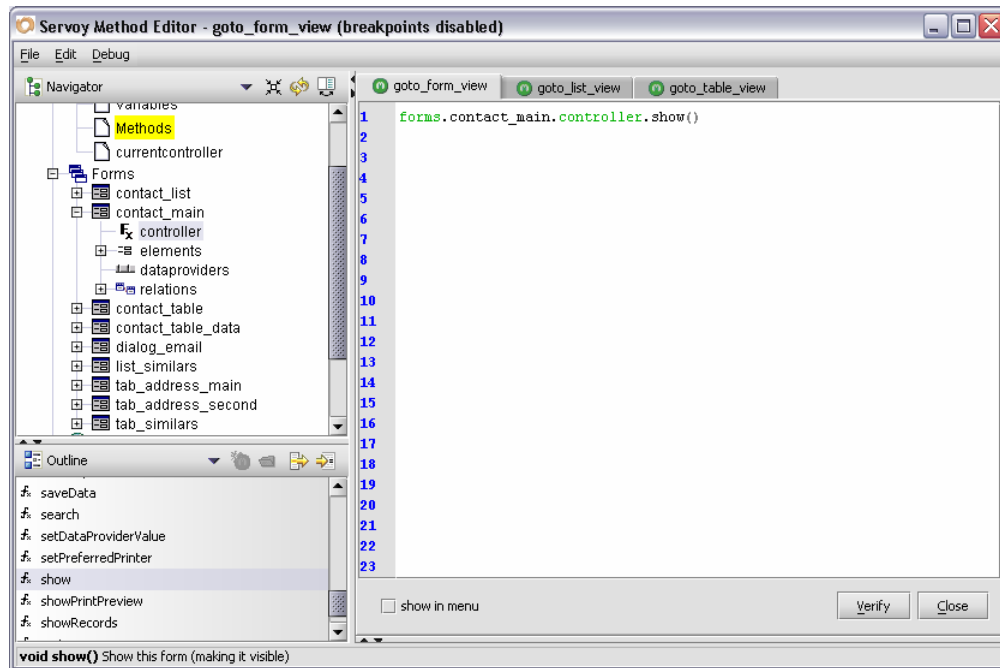
Now switch to the contact_table form and click on the grey form button, double-click on the `onAction` property and choose the global method `goto_form_view`.

Click on the grey list button (to the left of the purple table button) – double-click on the `onAction` property and choose the global method `goto_list_view`.

Open the Method Editor (select the “Methods...” tool from the toolbar or choose “Methods...” from the “Tools” menu) and click on the Methods node of the Globals tree. Open up the methods `goto_form_view`, `goto_list_view`, and `goto_table_view`. Click on the Script editor tab `goto_form_view`: click on the controller object on form `contact_main` (click the “+” next to the `contact_main` form if you can’t see the controller object) – and then double-click the `show` function.

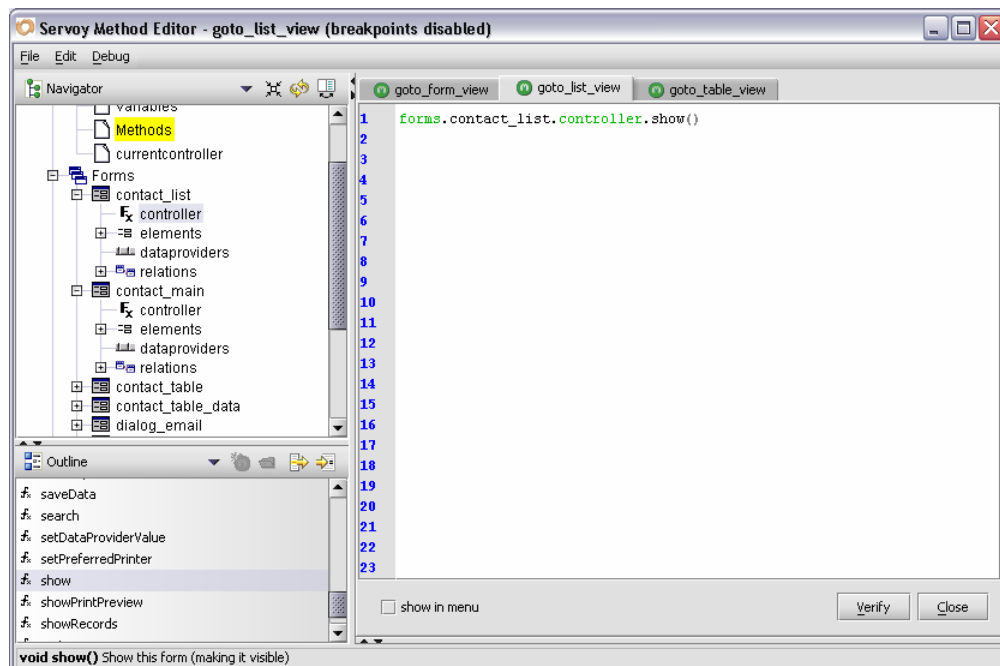
Your code should read:

```
forms.contact_main.controller.show()
```



Click on the tab `goto_list_view`; click on the controller object on form `contact_list` (click the "+" next to the `contact_list` form if you can't see the `controller` object) – and then double-click the `show` function. Your code should read:

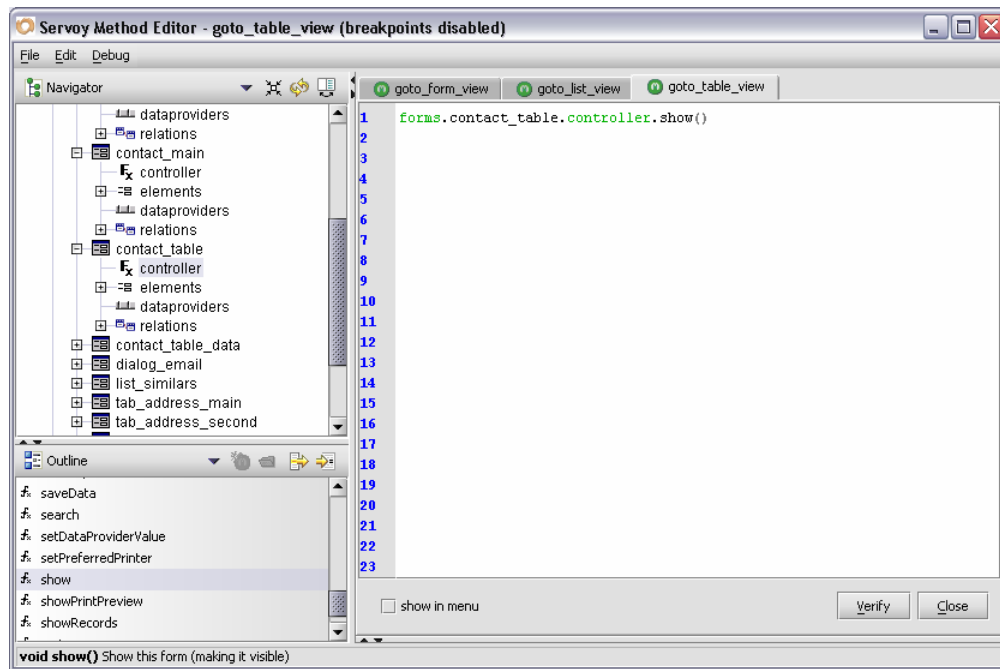
```
forms.contact_list.controller.show()
```



Click on the tab `goto_table_view`; click on the controller object on form `contact_list` (click the “+” next to the `contact_table` form if you can’t see the controller object) – and then double-click the `show` command.

Your code should read:

```
forms.contact_table.controller.show()
```



Now all of the navigation buttons are hooked up!

We have just a couple of more things to finish up. For the Related Contacts tab – we need to add a method to trigger that will update the displayed data when:

- The last name field is entered or changed
- The company field is entered or changed
- The city on the “Main Address” tab is entered or changed
- When we change contact records

To accomplish this – we’re going to create a new global method that will simply trigger the `search_similars` method (on the `tab_similars` form). We will then attach the script to the `onDataChange` property of the fields and to the `onRecordSelection` property of the `contact_main` form.

Let’s start by creating the method. In the Method Editor – click on the Methods item of the Globals tree – and create a new global method called `update_related_contacts`:

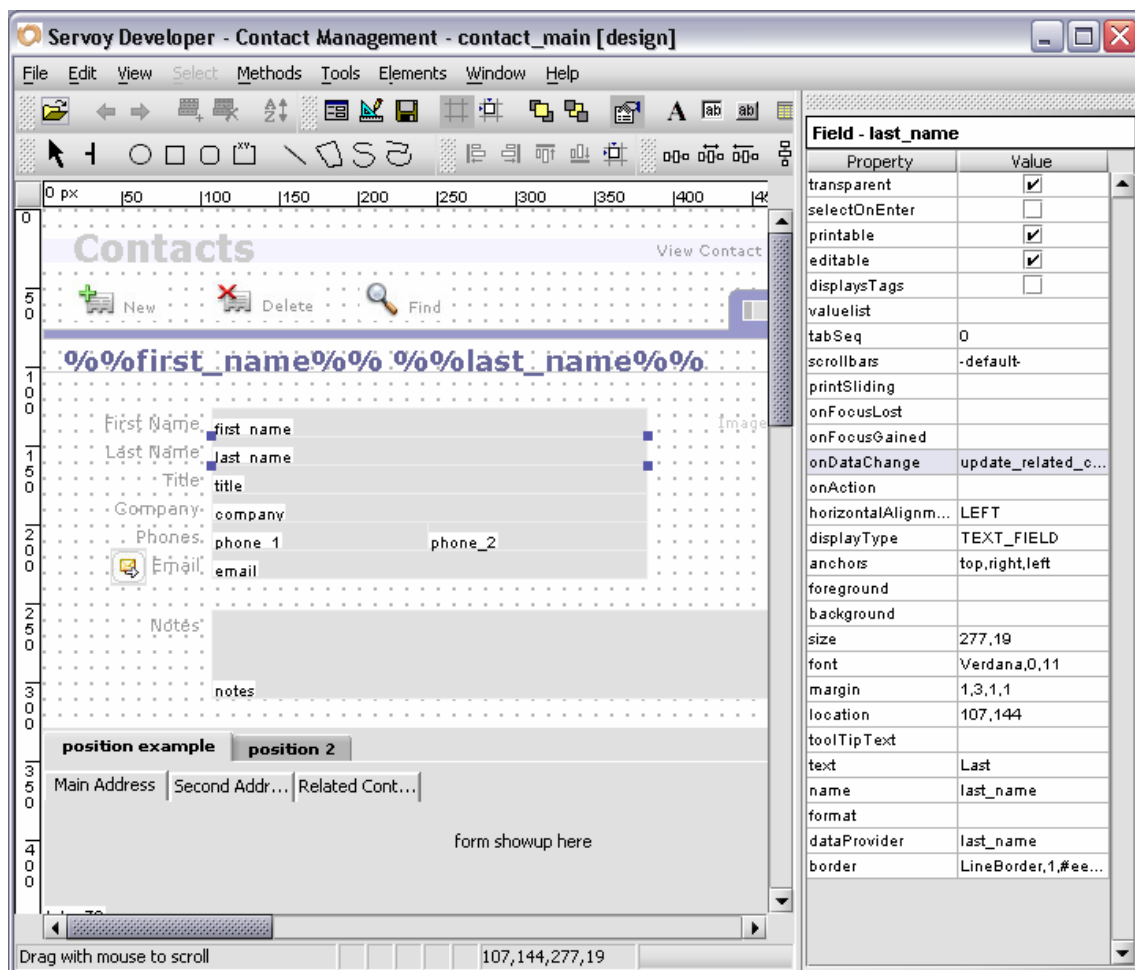


Here's the code we're going to use:

```
if(!globals.gSimilarBy)
{
    //the radio button is empty - so set it to Company
    globals.gSimilarBy = 'Company';
}

//perform search_similars method
forms.tab_similars.search_similars();
```

Now, navigate to the `contact_main` form – and click the `last_name` field. Double-click the `onDataChange` property in the Properties panel and choose the `update_related_contacts` global method:



Do the same thing for the Company field, and then click on the white part (the Body) of the contact_main form (so you can see the form properties). Set the `onRecordSelection` property to the global method `update_related_contacts`.



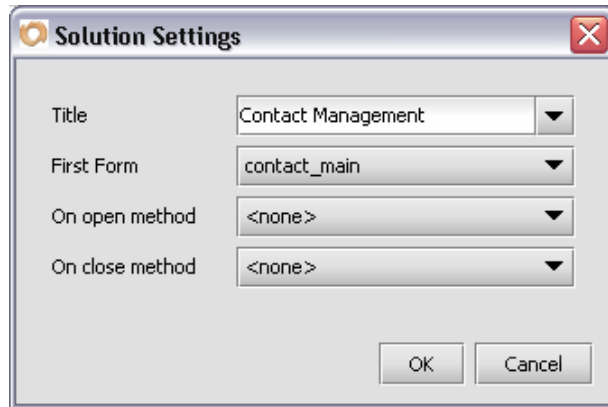
In the tabpanel – double-click the “Main Address” tab that’s inside the tabpanel (and you’ll be automatically shown the `tab_address_main` form). Set the `onDataChange` property of the “city” field to the global method `update_related_contacts`.

Navigate to the `tab_similars` form and set the `onShow` property of the FORM to the global method `update_related_contacts`.

Now the contents of our Related Records tab will change automatically when:

- The last name field is entered or changed OR
- The company field is entered or changed OR
- The city on the “Main Address” tab is entered or changed OR
- When we change contact records

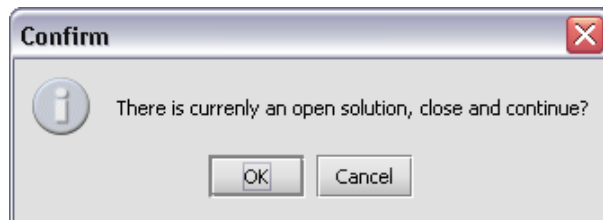
The last thing we're going to do in this tutorial is setup some solution properties. Choose "Solution Settings" from the "File" menu:



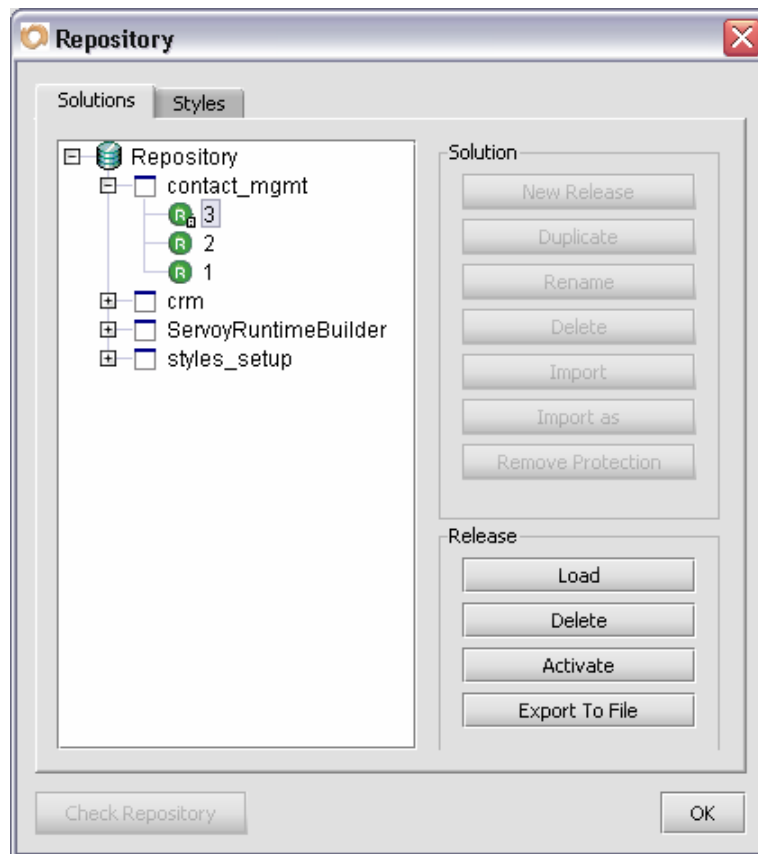
Enter "Contact Management" as the title, and choose contact_main as the first form. Click "OK" – and that's it.

Click through your solution – add and delete records, add and delete images, click between the tabs and test out your new solution. When you're satisfied that everything is working the way you want – then I always make an exported version of the solution as a backup.

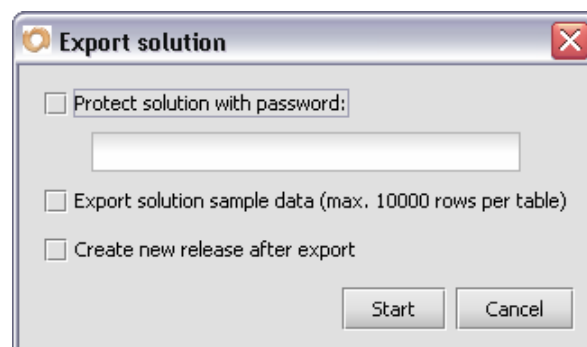
To make an exported backup – choose "Repository..." from the "File" menu. When you're asked if you want to close the current solution – click "OK."



Click on "+" next to the solution "contact_mgmt", click on the active release (the active release is the one with the little black "A" on it – usually the top one) and click the "Export to File" button.



You should see the Export solution dialog:



You can choose to export sample data with your solution; and whether or not you want to create a new release after the export is complete. If you choose to create a protected solution – you will be asked to enter a password. Once someone else imports that solution into their repository – and they try to go into Designer mode – or try to edit methods in the Method Editor – they will be prompted for the password. If they don't know the password, then they won't be able to open the Method Editor or enter Designer mode.

Click the "Start" button and choose a location to save your solution (I keep a "Servoy Solutions" folder on my desktop) – and name the solution (I use the solution name + "_x" where x is the version: i.e. contact_mgmt_v3.servoy) – and DO keep the .servoy extension.

The native file format of Servoy is .zip.

Yep, a plain ol' zip file.

Inside the .zip file are XML files that describe your solution along with the binary data of all the images you've imported. Our solution (without data) is a "whopping" 100K. Yep, 100K – and half of that are the graphics we used!

Pretty cool, huh? You can now send this solution to others – import it into other Servoy repositories running ANY SQL database (Oracle, MS SQL Server, MySQL, etc.) – and it will work – all without ANY re-coding on your part.

Conclusion

We've covered a lot of ground in this tutorial! This should give you a good overview of how Servoy works - and I hope that you've been inspired to try converting your own solution to Servoy.

I welcome all your feedback and comments: bob@clickware.com.

Acknowledgements

Many thanks to Marc Norman and Bob Cart for their tireless editing efforts! Thanks guys, I owe you a 7-up and Flat Tire Ale (respectively).

Copyright ©2004 ClickWare, Inc.. All Rights Reserved. All trademarks and registered trademarks are the properties of their respective owners.