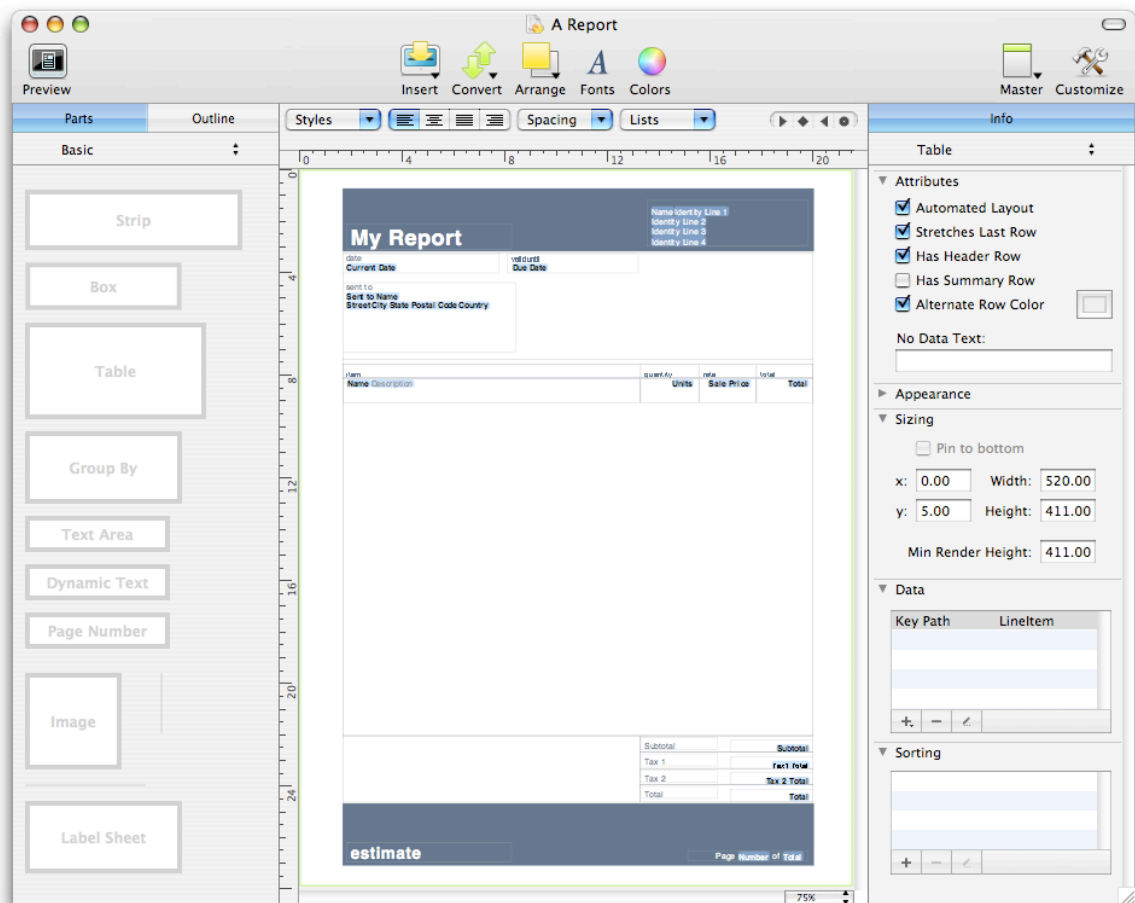


# Report Engine

## Common concepts for Daylite and Billings

Marketcircle Inc.

Last Updated December 14, 2011 • Version 1.0



# Table of Contents

<b>Document Scope</b>	<b>5</b>
<b>Overview</b>	<b>6</b>
Motivation	6
Audience	6
Design	6
Output Scope	6
Print Layout vs. Report	7
Print Layouts and Locations	7
Reports and Locations	7
Behaviors	7
Print Settings	9
Usage Process for Print Layouts	9
Usage Process for Reports	9
File Format	9
<b>Data Extraction - Basics</b>	<b>10</b>
What is an Object?	10
Starting Data Set	12
Getting Objects	13
Getting a Property	13
Browsing Objects	13
<b>Structure</b>	<b>15</b>
Element Hierarchy	15
Growing Height, Static Width	16
Coordinates, Origin and Flow	16
Element Overruns	16
Data Rules and Data	18
Data Tokens	19
Token Result and Formatting	20
<b>Elements</b>	<b>21</b>
Report	21
Page	21
Body	21
Strip	21
Table	22
<b>Marketcircle Report Engine (Common Concepts)</b>	<b>2</b>

Text Area	23
Box	23
Line	23
Image	23
Logo	24
Label Sheet	24
Group By	25
Checkmark	25
Color Chip	25
<b>Mathematical Expressions</b>	<b>26</b>
Keypath Mathematical Evaluation	26
F-Script Mathematical Evaluation	26
<b>Design Window</b>	<b>28</b>
Parts Area	29
Elements Outline	29
Inspector Area	30
Toolbar	30
Canvas Selection Behavior	32
Copy, Cut and Paste	33
Zoom In, Zoom Out	33
Layout Rectangles	33
Appearance Behavior	34
Preview	34
<b>User Input</b>	<b>36</b>
Description	36
Text Field	36
Date	36
Date Range	36
Popup	36
Element Choice	37
Accessing User Input variables	37
Accessing User Input variables from F-Script	37
Accessing User Input variables from a sub-merge	37
Accessing User Input variables from a Standard Filter Criterion	38
<b>Scripting</b>	<b>39</b>
Pre and Post Extraction Scripts	39
Script Filter	39

Script Token	39
Dynamic Text as Mathematical Expression	39
Popup User Input Script	39
Reserved variable names	40
<b>Variables</b>	<b>41</b>
Reserved Variable Names	41
Accessing Element Variables	41
<b>Help and other resources</b>	<b>43</b>

# Document Scope

This document is meant to provide an understanding of how the report engine works. It is not meant to show you how to create specific types of reports.

# Overview

## Motivation

The report engine was built out of frustration from a lack of appropriate tools on Mac OS X. We also ran into a challenge because both Daylite and Billings are applications that have an Object Orientated top end and a Relational Database Management System (RDBMS) bottom end. Most report engines are built for either pure DBMS style of applications or pure Object Orientated applications. A combination of the two is unheard of (at least to our knowledge).

Luckily, Mac OS X has great typographical and image tools and Marketcircle has a great object-relational abstraction layer. Together, with months of hard work, we created a Report Engine that mostly meets our needs. As of this writing, we still have features we want to implement.

## Audience

This manual is intended for a technical audience. Preferably you have had some report design experience in another app or environment, some scripting experience, or some software development experience. To reach some objectives, you need absolutely no coding experience and a novice can do it. For others, you may need to know f-script, the Cocoa API's, the Marketcircle Persistence API and the application's specific data model. For simple objectives such as customizing a label, please review one of the many training movies.

## Design

The Report Engine is a Marketcircle framework that is embedded in our applications. Having a separate application was deemed to be too cumbersome, especially when considering live previews as you build templates. The engine is extensible and each application that hosts the engine can add it's own unique elements. For instance, Daylite adds its own charts.

## Output Scope

The Report Engine can produce basic to complex output. For example, it can produce Dymo style labels on the basic end and f-script driven, multi-master, multi-page reports on the complex end. To simplify the configuration of a report, we have introduced 'Behaviors.' Five of the six behaviors are encapsulated by the term 'print layout' and sixth and final behavior is a pure report.

## **Print Layout vs. Report**

The difference between a Print Layout and a Report is simple. In Print Layout, a user selects which 'Objects' (i.e, Projects) they want to print. The Print Layout then works on that selection and lays it out according to the rules in a template. On the other hand, a report must fetch data from the database, using one of the query methods, then lay it out. So if what you want to print depends on what the user has selected, then you want a Print Layout. If you want to print something regardless of what the user has selected, then you will want a Report.

## **Print Layouts and Locations**

In Daylite, you will notice that when you specify a Print Layout Behavior, a series of Locations appear (i.e Contacts). The location serves two purposes. The first purpose is to narrow the number of fields that are available for you to work with. The second is what area of the application the Print Layout should appear in. Applications the size of Daylite have thousands of fields, it would be daunting to present you with hundreds of irrelevant fields during the design phase.

## **Reports and Locations**

In the case of Reports, location serves only one purpose. The purpose is simple—to group similar reports together.

## **Behaviors**

Behaviors determine how the paging of a template works. For example, when you select 5 contacts and you want 'Dymo' style labels, the paging works such that the system generates one page per contact. In another case, you may want a phone list; i.e., you want a single page for all contacts that you have passed in (paging happens when you run out of room on the page).

Behavior	Description
Label	<p>Used when you want to print to a Dymo or Seiko style of Label Printer. During the setup of the template, you need to have Printer Drivers installed to see all the label page sizes.</p> <p>Expects user selection. One page is generated for each user selected object.</p>
Label Sheet	<p>Used when you want to print Avery style label sheets. You can choose from one of 800 predefined presets. You can customize the settings to your liking.</p> <p>Expects user selection. User selection fills as much of a page as possible before generating another page. The Print Preview panel allows you to change the start location and number of copies.</p>
Envelope	<p>Used for envelopes. Choose one of the predefined Mac OS X envelopes. Be careful about printer orientation when creating your own envelope style.</p> <p>Expects user selection. One page (envelope) is generated for each user selected object.</p>
Page	<p>Similar to a Label, except you can have a body on this type of template. Used for cases like estimates or invoices or where you want one or more pages per selected object.</p> <p>One page is generated for each user selected object. Situations with estimates and invoices are special cases where the application determines the object to print and not necessarily the user.</p>
List	<p>Opposite to the Page behavior where all of the user's selection is rendered on a single page unless there is no room left. The canonical example is the Phone list.</p>
Report	<p>The standard report where objects are fetched from the database as opposed to passed by the user. You can also specify a set of inputs (i.e date range) to ask the user before running the report.</p>

## Print Settings

Variables such as page size and orientation are stored with each template. If you change the paper size or orientation, you will have to manually adjust all the elements on the page.

## Usage Process for Print Layouts

You define a Print Layout in the Report Editor and save the template in the proper file locations (usually Home -> Library -> Application Support -> Daylite (or Billings) -> Templates). The user selects some objects (typically in a list) and clicks the 'Print' menu. A sheet opens with all the applicable Print Layouts. The user selects a Print Layout of their liking to preview the output. They can then choose to Print, Save to PDF, or choose another Print Layout.

## Usage Process for Reports

You define a Report in the Report Editor and save the template in the proper file locations (usually Home -> Library -> Application Support -> Daylite (or Billings) -> Templates). The user clicks the 'Reports' menu. A sheet opens displaying all the Reports. The user selects a Report. If the Report defines 'User Inputs,' then the user is asked for the Inputs (i.e. date range). After they have entered the input, they click the 'Run' button to generate a preview of the Report. The user can then Print, Save to PDF, or choose another Report.

## File Format

A template is stored in a 'file wrapper' or 'file package' with the extension .dlreport for Daylite and .bireport for Billings. It contains a minimum of three files:

File	Purpose
Info.plist	An XML plist that stores the behavior, location, display name, and user inputs for Reports.
Template.archive	An XML archive of layout information and data extraction rules.
Printinfo.archive	An XML archive of the NSPrintInfo object. It contains the page size and orientation.

**Developer Note:** No harm is done to CVS or SubVersion files within the file wrapper when using the Editor.

# Data Extraction - Basics

A Print Layout or Report is the consolidation of data from a database and layout. In this section, we will discuss the various ways of extracting data.

## **What is an Object?**

In an object oriented environment, the word 'object' is often used and can mean different things at different times. For the sake of this document, we will define general types of objects to simplify things.

Type	Description
Property	This is the most basic type of object. The most common examples are: String, Number (integer, float, boolean), Date, Color, Image, Data.
Dictionary	A dictionary is a container of named properties. Each property can be accessed using a name (i.e. firstname). We sometimes refer to them as Raw Row(s) because in the Marketcircle Persistence layer, they represent a single row in the database.
Business Object	<p>An object helps to define the data characteristic of an application. In Daylite, a “Contact” is considered a Business Object. Business Objects are comprised of properties (sometimes called attributes), other Business Objects (called to-one relationships) and other Arrays of Business Objects (called to-many or many-to-many relationships).</p> <p>Another important aspect is called ‘Business Logic,’ sometimes called ‘Derived Attributes’ or ‘Derived Properties.’ For example ‘derivedFullname’ on Contact in Daylite is Business Logic because the data that is returned does not really exist, but it is constructed on the fly. Many of Daylite and Billings’ Business Objects have Business Logic, some do not.</p> <p>An Entity is the description of a business object, its properties and relationships.</p>
Array	Array, List, Bag are synonyms in this document. They represent a collection of Business Objects or Dictionaries (raw rows). When you trigger a to-many relationship on a Business Object, it will return an array of other Business Objects. For example, if you call ‘tasks’ on a contact, you will get an array of Business Objects (of type Task as described by the Task entity).
Null Object	A Null Object is a placeholder for ‘no value’ and is most often used in a Dictionary and sometimes in Business Objects.

## Starting Data Set

When a Report or Print Layout is asked to run and render, the system gives it a Starting Data Set. This set contains a minimum of three data elements:

Key	Description
currentUser	The Business Object that represents the current user using the application.
userDefaults	The object representing the application's user defaults (NSUserDefaults).
databaseUserDe- faults	The object representing the current user's defaults that are stored in the database (MCUserDefaults).
objectContext	The gateway to the database. (See Developer API for more info.)
objects	Only available with List and Label Sheet behaviors. Represents the user's selection (an array of business objects).
object	Only available with Label, Envelope, and Page behaviors and represents the object passed in. In some cases, it is the user's selection broken down and in other cases, the application is passing a single object (i.e Estimate or Invoice).

## Getting Objects

There are five ways you are going to get data to work on.

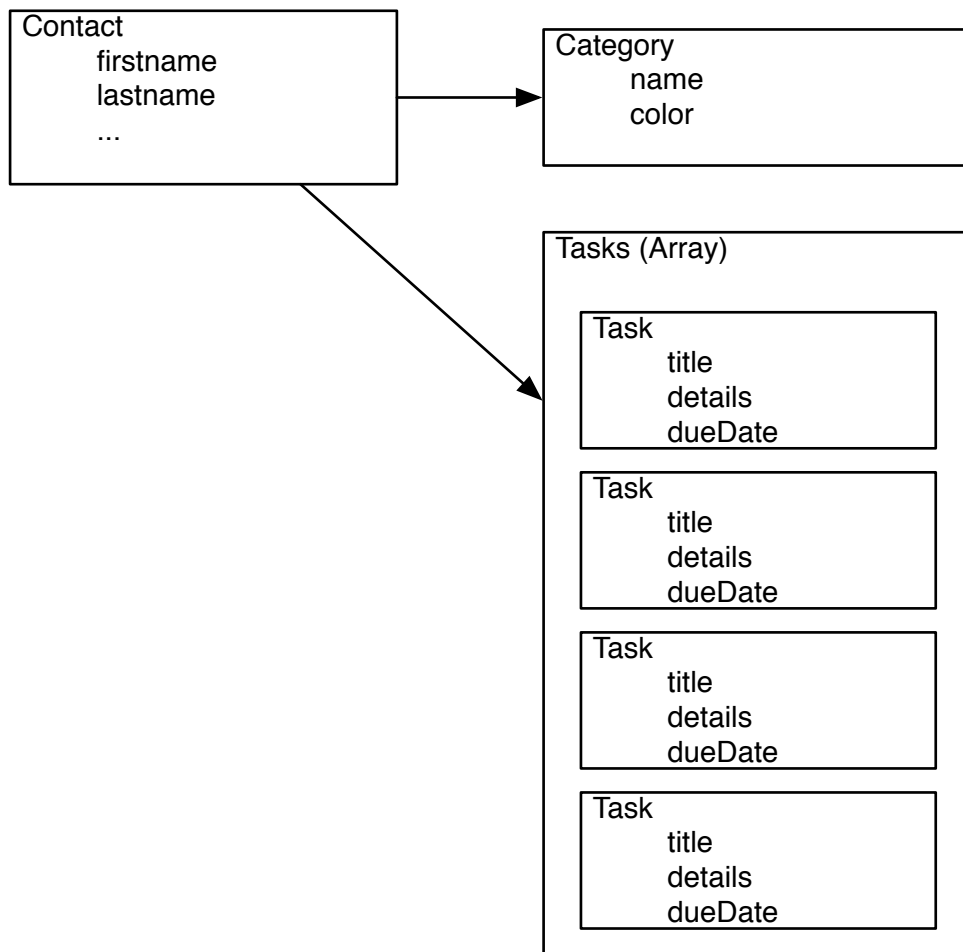
Type	Description
Standard Filter	This is the most common filter for tables and group by's. You can specify where you want the objects to come from—whether directly from the database, or the user selection (where applicable), or a relationship on a parent element. Once a 'source' is specified, you can filter it using criteria. It also includes conveniences to format the table.
SQL Filter	You construct SQL and get an array of dictionaries back. (i.e. <code>SELECT firstname, lastname FROM Contact WHERE firstname LIKE 'Jo*'.</code> ) With this technique, you can substitute user input keys. (i.e. <code>SELECT firstname, lastname FROM Contact WHERE firstname LIKE '&lt;\$userInput.nameStart\$&gt;*'.</code> )
Script Filter	Using F-Script, you interact with the Object Context to get objects. (i.e. <code>myObjects := objectContext objectsForEntityNamed:'Contact'.</code> ). You can also substitute user input keys in the script.
Relationship Filter	While this method is not technically a data fetcher, it is here because it is how you specify that you want the passed in objects (or a relationship from a parent object in a nested table) to be set on a specific table.

## Getting a Property

Now that we have arrays (lists) of objects, we need to display individual fields or property. The only way to display a property (i.e a firstname) is by using a Dynamic Text Area. A Dynamic Text Area is a mini text layout engine. You can mix regular text with 'tokens.' Learn more about tokens below.

## Browsing Objects

To navigate the data model, you need to understand Object Oriented Data Models. It is not as scary as it sounds. Above, we mentioned properties and business objects. You need to add two more concepts to the mix—To-One relationships and To-Many relationships. To-One relationships return a maximum of one Business Object. To-Many relationships return an Array of Business Objects.



In the illustration above, we are showing 3 business objects (Contact, Category and Task), 7 Foundation Objects (firstname, lastname, name, color, title, details, dueDate), 1 to-one relationship (category) and 1 to-many relationship (tasks).

So, if you have a Contact and want to display all the associated Tasks, you would use a Table (it could be nested inside another Table).

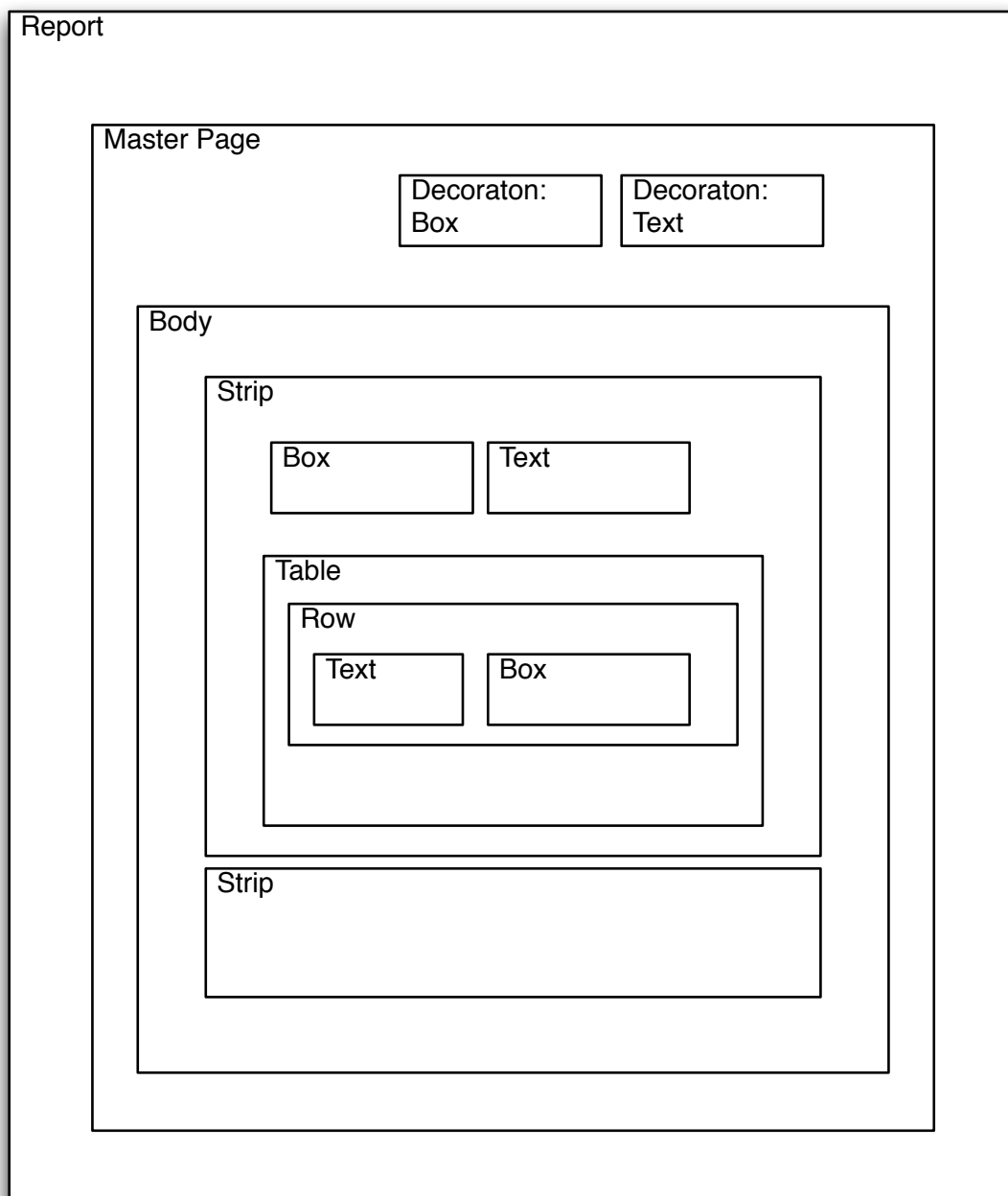
The important thing to learn here is that you should be aware where you are in the graph. You can think of it as a tree with branches and leafs.

# Structure

The Report Engine is built around one prevailing concept—the element hierarchy.

## Element Hierarchy

Every piece or chunk or segment of a report is an ‘element’ and some elements can contain other elements. For example, a report contains master pages, each master page can contain a body, a body contains one or more strips etc.



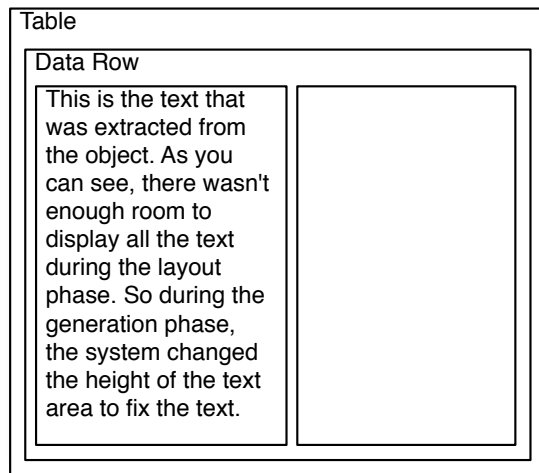
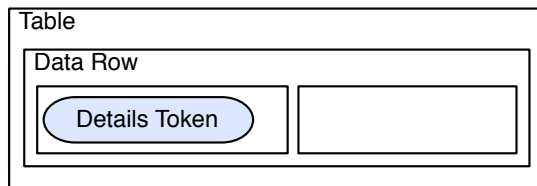
## Growing Height, Static Width

When you perform your layout, you are guaranteed that the width will remain the same once the report generation process is completed. You cannot count on the height however. The height is dynamically determined by the amount of space needed for the data during the generation phase.



This is the text that was extracted from the object. As you can see, there wasn't enough room to display all the text during the layout phase. So during the generation phase, the system changed the height of the text area to fit the text.

In most cases, if a child element grows, it's parent element will grow as well. So, if a dynamic text area in table row grows, the table row will grow as will the table etc.

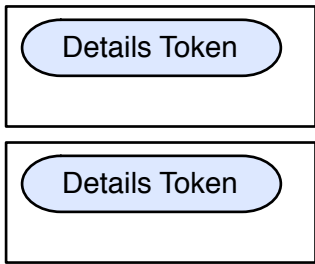


## Coordinates, Origin and Flow

The engine uses a flipped Cartesian coordinate system. The origin (0,0) is at the top left. The engine always flows from top to bottom. So, if an element grows, it grows downwards, not sideways or upwards.

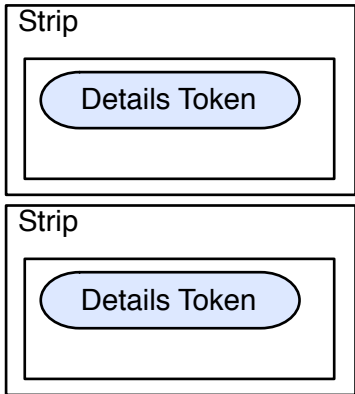
## Element Overruns

Since elements can dynamically grow during the generation phase, we need a way to push other elements further down.



This is the text that was extracted from the object. As you can see, there wasn't enough room to display all the text during the layout phase. So during the generation phase, the system changed the height of the text area to fix the text.

Strips are used to distinguish moveable areas. (as of 3.0, strips can only be added to bodies. However, this may change in future).



Strip  
This is the text that was extracted from the object. As you can see, there wasn't enough room to display all the text during the layout phase. So during the generation phase, the system changed the height of the text area to fix the text.

Strip  
This is the text that was extracted from the object. As you can see, there wasn't enough room to display all the text during the layout phase. So during the generation phase, the system changed the height of the text area to fix the text.

## Data Rules and Data

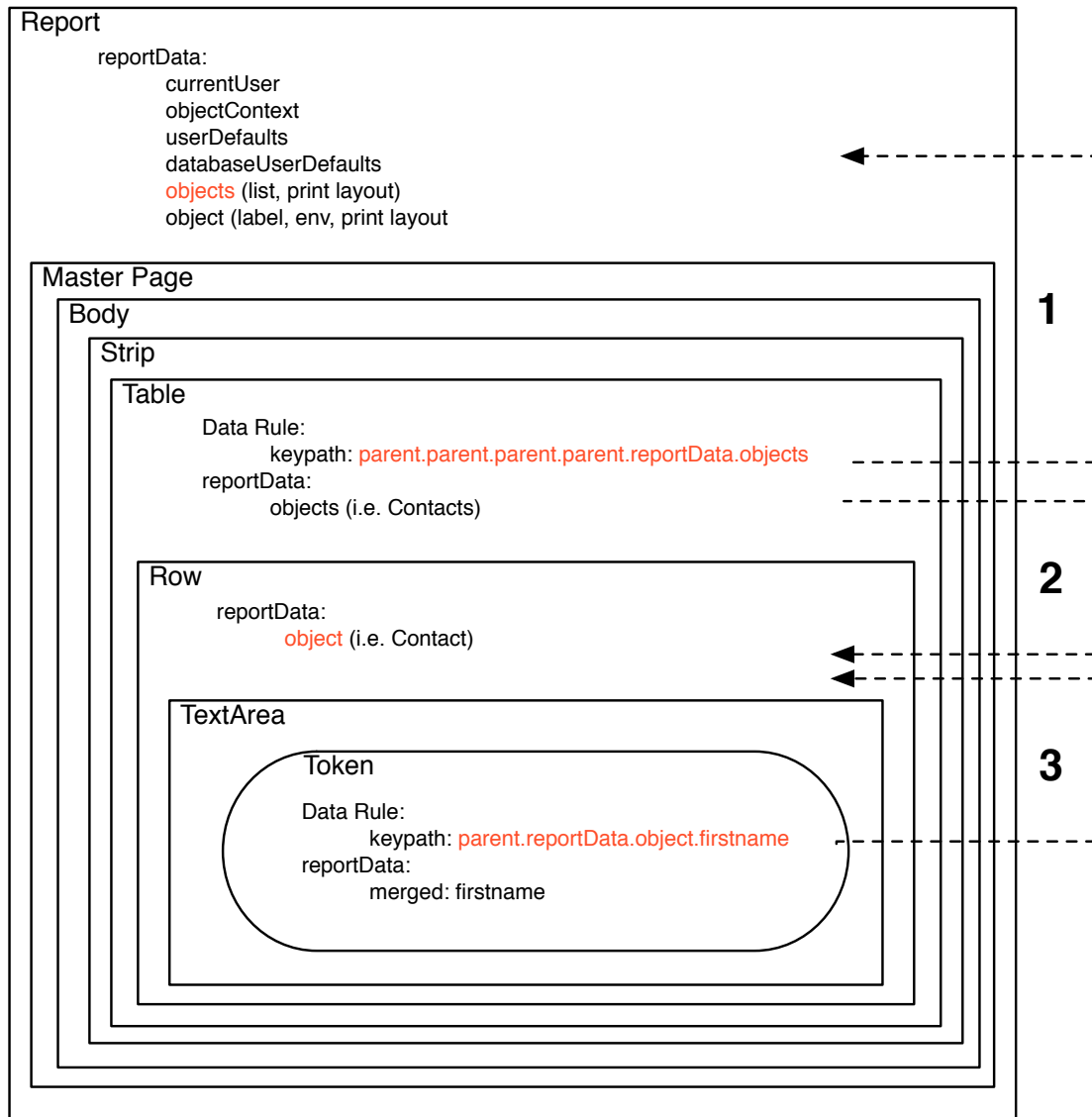
Now that we have a system of laying out elements and growing them, we need a way to specify what data goes into those elements. We use Data Rules to identify specifically what data an element works with. During the generation phase, the Data Rules are expressed and the result of the rules are put in a data dictionary on the element. The rules are expressed from top to bottom starting with the report element itself. Each element can have zero, one, or more data rules.

For example, Table A has a rule: (F-Script, fetch all contacts). During the generation phase, the rule is expressed and F-Script is executed. The result of that rule is 100 contacts. Those 100 contacts are put into the data dictionary on Table A. The most standard keys are 'objects' or 'object.' If you called `reportData.objects` on Table A, you would get the 100 contacts. You can think of it as populating a variable.

To continue the example, after the rule is expressed, Table A would create 100 data rows. For each data row, a contact is put in its data dictionary with the key 'object.' So if you call `reportData.object` on the data row, you would get a single contact.

Going deeper, the data row has a Dynamic Text Area acting as a cell. In that Dynamic Text Area, you have a token with a 'keypath' rule. This rule has the following keypath: `parent.reportData.object.firstname`.

- 'parent' gets us the data row (in the hierarchy, it is the parent of the dynamic text area in this case)
- 'reportData' gives us the data dictionary on the row
- 'object' gets us the contact
- 'firstname' gets us the firstname on the contact



## Data Tokens

Users expect their output to consist of data fields (as opposed to Objects, Rows, etc). You may want to style that field to be bold, left justified with a specific date format and surrounded by other text. We accomplish all of this with the Token Merging Engine inside a Dynamic Text Area. You can think of a Text Area as a mini TextEdit (the Application). You style the text in a Text Area exactly the same way you would in TextEdit. When you want a bit of data from the application to be inserted in some text, you use a Token.

This is a bit of text with a Detail token in it.

There are currently 4 types of tokens.

Token Type	Description
KeyPath	This is the most common token type. Basically, it pulls data from anywhere in element hierarchy. You can name them anything you want, but the keypath must be valid. An example keypath is: parent.reportData.object.firstname or parent.parent.reportData.objects.@sum.totalCachedAmount
F-Script	You can write a complex f-script inside this type of token. You can access the Object Context, Generator, User Input Dictionary and the Element (thus the whole hierarchy is available to you). Within the token, you can compose complex NSStrings or NSAttributedString. This token gives you immense power and flexibility.
Sub-Merge	With this token, you can use the old merge commands such as <code>&lt;\$foreach note element.parent.reportData.object.notes do\$&gt;</code> . You can learn more about these commands in the Daylite Developer Kit.
Current Date	This token simply returns the current date. You can apply your preferred date format.

### Token Result and Formatting

The results of your token must be a property (NSString, NSNumber, NSAttributedString or NSDate). If it is not, 'description' is called on the object and that is returned. You can then apply a format such as currency, short date, long date etc. You must be careful to choose the right kind of formatter such that it can work on your result. For example, choosing a currency format when a NSDate is returned will cause an error.

# Elements

We will cover the elements available in the Report Engine. Some elements are 'leaves,' they cannot accept other elements as children. Other elements are 'branches' and can accept other elements as children.

## Report

The report itself is an element and it is the root of all elements. Its direct children can only be Page Elements (one or more). This is where you define what type of data is coming in (including 'Objects' for Print Layouts) and page size. You should never edit the incoming data types as the setup assistant does that for you.

## Page

A Page and Master Page are synonymous. You must have at least one Page in a Report, no matter what type (behavior) you specify. A page's immediate children can be: Body, Text Area, Page Number, Box, Line, Image, Label Sheet.

Text Area, Page Number, Box, Line and Image are considered 'decorations' at this level. They will be copied for each new page the generator adds during the generation process. It is with decorations that you achieve header, footer, and background styles.

## Body

A body defines the area in which sub-elements can extend into as they grow. You can think of it as defining page margins. As a convenience, you can hide the body (and everything contained within) to work on the decorations (i.e background image). Body can only have Strips as children.

## Strip

Within a body, you must have at least one strip. A strip defines an area that can grow in height. If it grows, it will push other strips (after it) further down (see Element Overruns above). If you have more than one strip, you can 'pin' the last strip to the bottom of the page. This is handy for estimates and invoices where you need an area at the bottom of the last page for totals, taxes etc.

## Table

A table allows you to display rows of data. It can have 3 types of children: Header Row, Data Row and Summary Row. Header and Summary are optional, Data Row is mandatory. You can turn them on or off from the Table attributes inspector.

Attribute	Description
Automated Layout	In Manual mode, you have to manage the 'cells.' In Automated mode, when you add a cell, it is automatically sized. If you have a header and/or summary row, in automated mode, cells are added there as well. You can think of it as adding 'columns.' The benefit of manual mode is that you can set up pretty complex layouts.
Act as Live Table	There are times when you want to simply print the columns that are displayed in the main UI. Act as Live Table enables you to do this. Only the attributes of the first column are relevant—additional columns are ignored. Border, background and text attributes are respected while creating the dynamic columns.
Stretches Last Row	This works in conjunction with the minimum render height (sizing). Useful for Invoices and Estimates. Takes the border attributes of the last row and stretches them to meet the height of the table.
Alternate Row Color	You can specify a background color for alternate rows.
No Data Text	Specify the text you want to display when the data rules fetches no data. The text takes the styling of the first cell (left most).
Has Header Row	Specifies whether the table has a header row. A header row is repeated on every page should a table have more rows than can fit on a page.
Has Summary Row	A summary row is useful when you want to perform calculations such as a sum on a column.

## **Text Area**

Handles all the text layout for the Report Engine. In static mode, text is edited in place. In dynamic mode, you can perform token merging or mathematical expressions. A Text Area has the same stylistic behavior as TextEdit that include: left, right, center alignment, line spacing, kerning, ligature, tab stops and more. Text areas handle Unicode characters and all fonts supported by Mac OS X. See Data Tokens above and Mathematical Expressions below. Text Areas cannot have children elements.

## **Box**

A container that typically has borders turned on. Allowable children: Box, Line, Text Area, Table, Group By, Image.

## **Line**

A vertical or horizontal line. You can control the thickness, color and basic dash pattern.

## **Image**

Handles all image formats supported by Mac OS X including PDF, EPS, PSD, TIFF, GIF, JPEG, PICT, PNG and over 30 more.

Attribute	Description
Source	<p>Specifies where the image is coming from.</p> <p><b>Static:</b> Image dragged in.</p> <p><b>Path from Key:</b> A keypath returns the file path where the image resides.</p> <p><b>Absolute Path:</b> Specify the absolute file path where the image resides.</p> <p><b>Image from Key:</b> A keypath returns an UIImage object.</p> <p><b>Named:</b> Use one of the application's internal images.</p> <p><b>Mapped Name:</b> A Keypath returns a string or a number. That value is mapped to named images. (see Mapped Names below)</p>
Mapped Names	The source kind is Mapped Name. This list represents the image names and the value that make that image visible.
Default Image Name	An image is used if any of the sources don't return anything useable.
Scaling	Determines scaling behavior.
Alignment	Determines image alignment.

## Logo

Logo is a special kind of image. The keypath for source of the image is set in the Application's preferences. Once the user sets their logo choice, that path will feed the Logo element at render time.

## Label Sheet

A label sheet provides the ability to create Avery style sheets. We have provided some 800 presets (some of which are outmoded). You can also define your own settings by modifying the attributes in the Label Matrix. A Label Sheet can only work in the place of a body (it cannot be in a body because the sheet controls the margins), and its only immediate child is a Label Matrix (where you define the parameters). The only allowable child of a Label Matrix is a Label Cell. In a Label Cell, you can have children of type Box, Text Area, Line and Image. When you define a new Label Sheet, all these elements are created for you. You just have to edit the contents of the Label Cell.

## **Group By**

Group By is a complex element. Once the data source has been identified, you use a Keypath to identify which field you want to group by (i.e. Category). During the generation process, all objects with the same key are grouped together and that result is put on the Group By Strip. In that strip you put your other elements (most likely a table).

## **Checkmark**

The checkmark element is used when you want to display a checkmark or ballot X for 'true' value. You set a keypath as the data rule on a checkmark. If the result of keypath resolves to NULL, nil, NSNull, empty string, empty array, zero or false, no checkmark or ballot X will be displayed.

## **Color Chip**

A color chip is used to display something like a category color. Its data rule is a simple keypath. The return value of the keypath must be a color property—specifically NSColor.

# Mathematical Expressions

There are two types of mathematical evaluation mechanisms in the report system. The first is built into the keypath mechanism and the second is provided by F-Script.

## Keypath Mathematical Evaluation

You can do four basic operations using the Keypath mechanism. Sum (@sum), average (@avg), minimum (@min) and maximum (@max).

So given an Array of Opportunities (in Daylite), we would get the sum like this:

```
opportunities.@sum.cachedTotalAmount
```

You would get the minimum like this:

```
opportunities.@min.cachedTotalAmount
```

In this case, 'cachedTotalAmount' is called on all the opportunities in the array and the sum or minimum is returned.

The most common expression of this concept in the Report Engine would look like this:

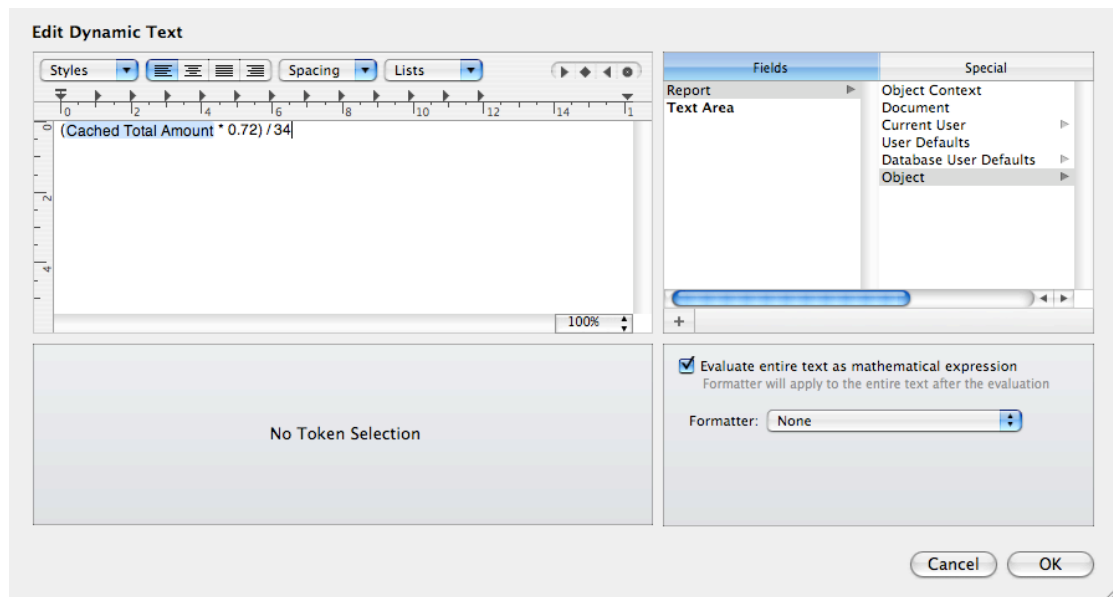
```
parent.parent.reportData.objects.@sum.cachedTotalAmount
```

and you would typically see this in Summary Row of a table. It is parent.parent because we need to get to the Table (where the 'objects' are) and not just the Row (where the 'object' is).

## F-Script Mathematical Evaluation

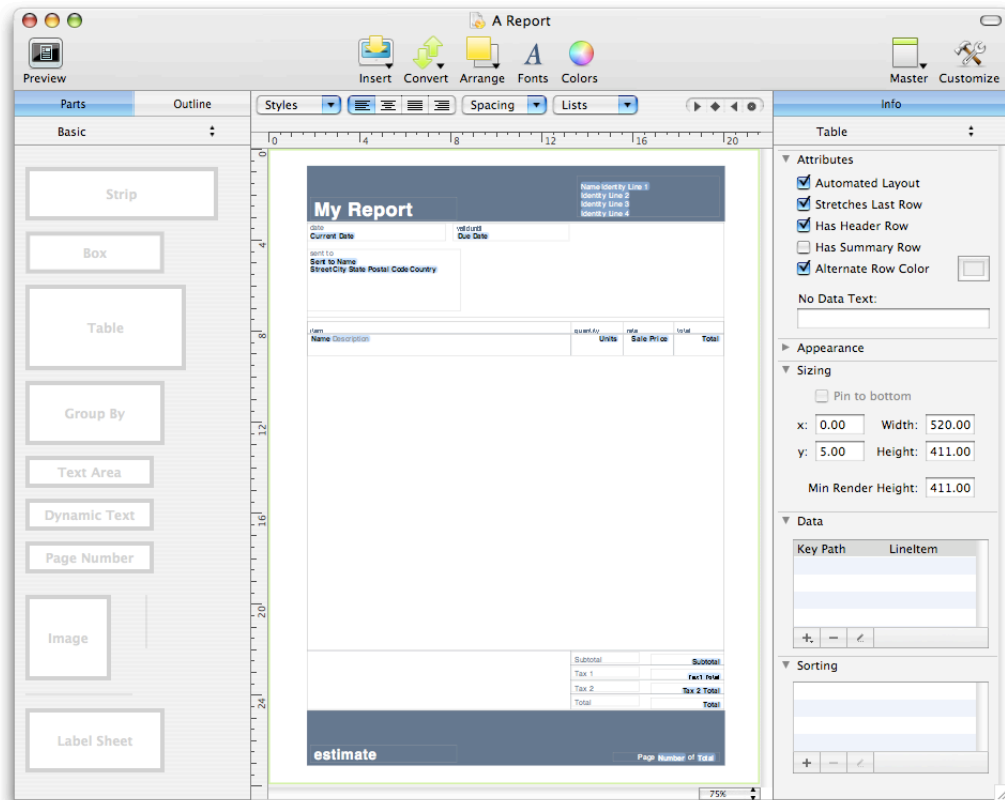
F-Script provides many common mathematical operators such as: addition, multiplication, division, subtraction, abs, arc cosine, arc sin, arc tan, cosine, negation, sin, square root, remainder, ceiling, floor, truncation, etc. Look at the F-Script guide to learn about more mathematical operators.

You can use these operators anywhere F-Script is used. You can also turn an entire Dynamic Text Area into a mathematical expression. The beauty here is that you can extract some values from the database using Tokens, then use the results that the Tokens return as part of a mathematical expression. You must specify that the Dynamic Text Area is evaluated as an expression by ticking the 'Evaluate entire text as math...' checkbox.

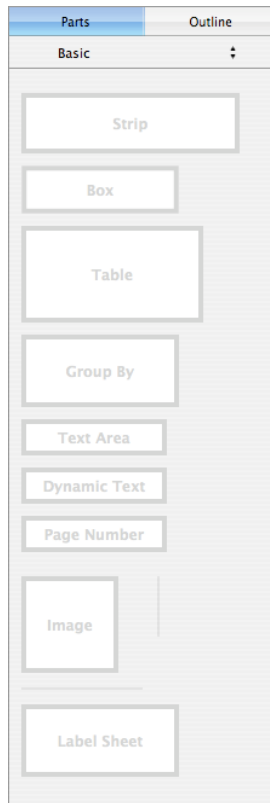


# Design Window

The design window is where you define and edit either Print Layouts or Reports. The design window is embedded in Daylite 3 (or higher) and Billings 2 (or higher). Each application may add its own extensions (in fact, any developer could add extensions).

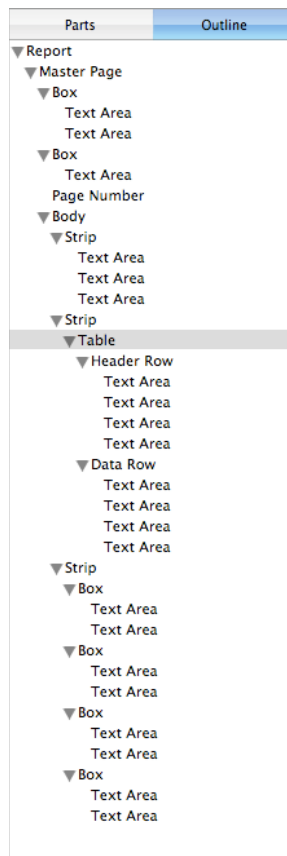


## Parts Area



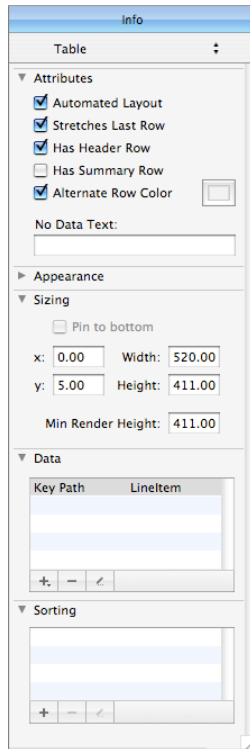
- The Parts area is where you can drag elements from and drop them on the canvas. Depending on the hierarchy rules, some elements may not be dragged into or on some elements. You can also use the 'Insert' toolbar item.

## Elements Outline

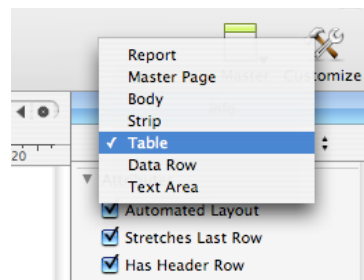


- When you expand the outline using the contextual menu, you can see the entire structure of a Report or Print Layout. You can change the selection on the canvas by changing the selection in the outline. In some cases, it may be very difficult to select some elements on the canvas and you can use the outline in those cases. Use the checkbox to hide elements.

## Inspector Area

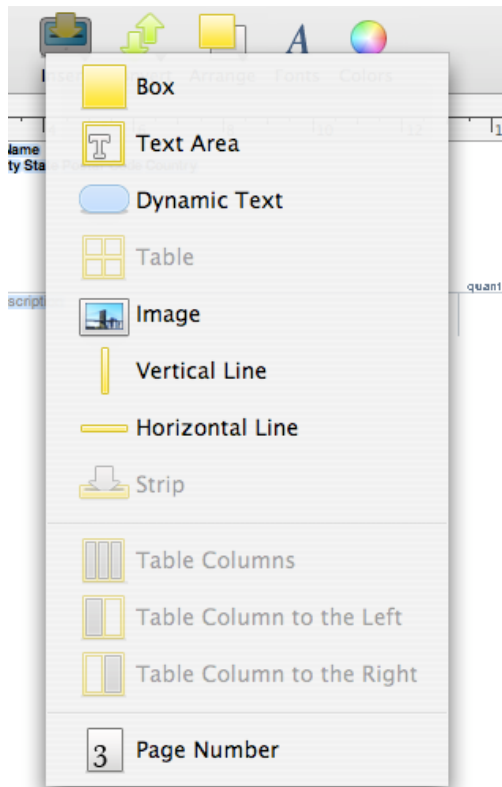


- The Inspector Area is where you will set all attributes of an element. The area changes based on the selected element. You can expand and collapse inspectors. Some inspectors are common to almost all elements (Appearance, Sizing) and some are unique to the selected element.
- You can also inspect attributes of a parent element without changing the selection. Click the menu item at the top of the Inspector Area to navigate to a parent.

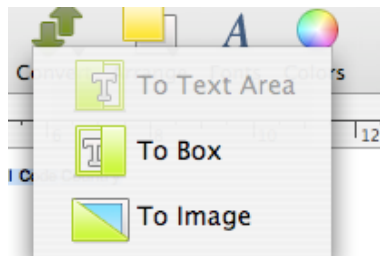


## Toolbar

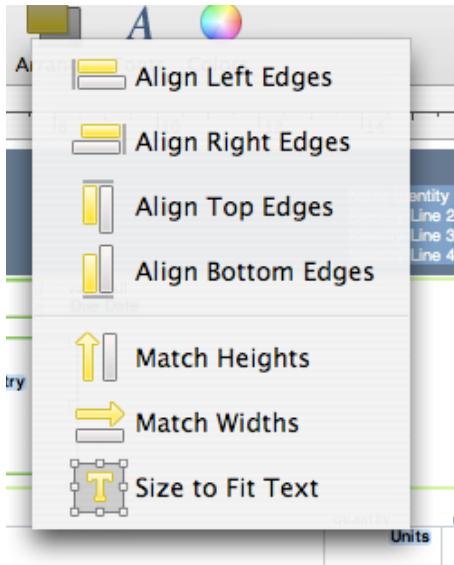
Since the Report Editor is meant to be embedded in another application, it cannot have too many regular menu items of its own. Instead, all functions that you would expect in a menu are in the toolbar. You can customize the items in the toolbar just as you would in any Cocoa based application with a toolbar.



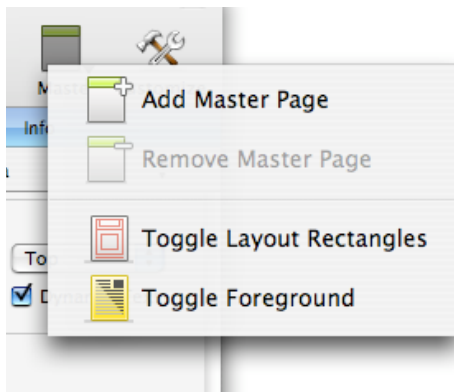
- The Insert toolbar item has all the basic elements. You can use this toolbar instead of dragging elements from the Parts area. You will notice that the item becomes enabled and disabled based on your current selection on the canvas. If an item is disabled, then it cannot be added in the parent element. For example, you cannot add a table column to the left unless you have a table row selected.



- The Convert toolbar item is handy when you want to convert a text area into a box or vice versa. The most common usage is inside tables, when you want to change a 'cell.'



- The Arrange toolbar item is handy when you need to line up elements or match their sizes. You need at least two elements selected for most of this menu to work.

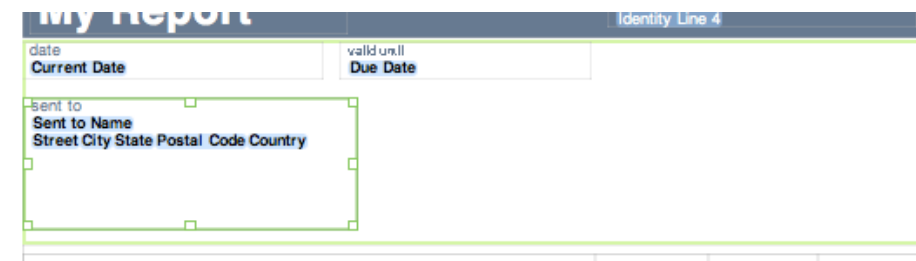


The Master toolbar item is for miscellaneous report wide actions such as:

- Add/Remove Master Pages (useful if you need a cover or closing page for a report or print layout).
- Toggle Layout Rectangles (see below).
- Toggle Foreground (you can hide the 'body' so that you can work with the background page decorations unimpeded).

### Canvas Selection Behavior

Because the Report Engine works on a nested concept, selection can be a bit tricky. You must double click to select a working region, then single click inside. To deselect the 'region,' you must double-click outside the element.



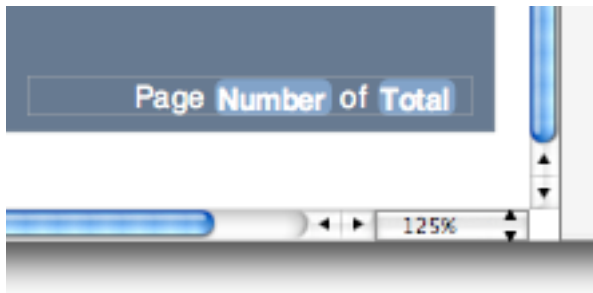
The illustration above shows that the 'send to' text area is the current selection within its parent region. Be careful not to move an element outside the visible range (sometimes it is useful to partially hide an element). If you 'lose' an element outside the visible range, you can use the sizing inspector to change the origin of the element and move it back into the visible area.

### Copy, Cut and Paste

You can cut, copy and paste elements. You can also option-drag elements to copy them (similar to Photoshop and Illustrator). It is safe to copy/paste elements within the same parent element in all cases, it is **not** safe however, if you have a keypath and you copy from one parent to another. The keypaths could become invalid and the report or print layout will have errors.

### Zoom In, Zoom Out

You can zoom the canvas area using the zoom popup menu at the bottom right of the window or by pressing the minus (-) and plus (+) keys (similar to Photoshop and Illustrator). The - and + keys only work while you have a selection in the canvas.



### Layout Rectangles

To help see the outlines of elements, you can toggle layout rectangles. There are 3 modes: none, light and heavy.

item	quantity	rate	total
Name Description	Units	Sale Price	Total

item		quantity	rate	total
Name	Description	Units	Sale Price	Total

You toggle these modes using the Master -> Toggle Layout Rectangles toolbar item.

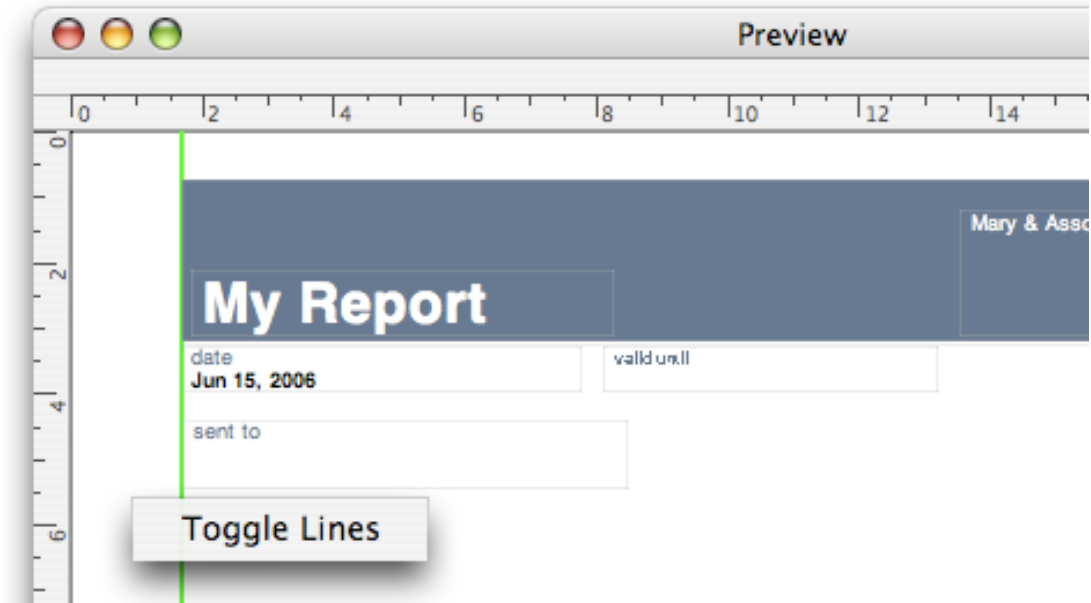
### Appearance Behavior

All elements can have borders and a background color. Each border can have different color and width. This is handy when you need to highlight a particular area in the table.


This table shows different border widths and colors. The last row shows a row background color. The table itself could have a border and background color and so could the parent element containing it. You can edit these settings in the 'Appearance' inspector.

### Preview

When working with your Report or Print Layout, you can preview your work. Simply click the Preview toolbar menu item. You can toggle layout rectangles in the preview window by using the contextual menu (click in the margins area). The layout rectangles have additional lines that illustrate what the Report Engine thinks is the margin (green) and what it believes is the current end line (red) and what it thinks is the bottom (blue). This information is helpful when figuring out placement issues.



The preview mechanism will pull arbitrary data from the database as sample data. If you use Visual Query, F-Script or SQL filters, then those are run normally. You cannot preview reports that have User Inputs.

# User Input

In some Reports, you may want to ask the user for some criteria. You can do so by adding User Inputs to the Report. The User Input inspector is only available on templates with the 'Report' behavior.

The mechanism is pretty simple. You define an ordered set of inputs and specify a variable name. When the user inputs a piece of data (i.e. a date), that value is set a variable on Report element thus making the variable available to any script or Smart list criterion. You can specify whether an input is mandatory or not. If it is mandatory, the Report will not run unless the user enters something in the input field.

Each user input type has an editor—a sheet that drops—that allows you to edit it.

## Description

The Description input type is not really an input type—it is rich text that you use to describe something to the user (i.e. This report will do the following). The Description input type uses the standard rich text editing rules (i.e. bold, font, size, spacing, etc).

## Text Field

The Text Field input type asks the user for textual input (i.e. the name of a contact). You can specify a key such as 'theName.'

## Date

The Date input type asks the user for a date. You can specify a key such as 'theDate.'

## Date Range

The Date Range input type asks the user for a date range (start and end). You need to specify two keys such as startDate and endDate. The dates are given to you as the beginning of the day for the start date and the end of the day for the end date.

## Popup

The Popup input type asks the user to select an option from a menu list. You should specify whether the options are just strings or properties from objects. If you select 'Objects' as the kind, then you will need to specify the keypath for the title. If you want the object to be returned, don't put anything in the value key field. If you want a specific property to be returned (say the primary key), then put a keypath to the property in the value key field.

The script area is used to get the array of objects that populates the popup menu. At the moment, you must use F-Script.

## Element Choice

This input type asks the user which ‘parts’ of a report should be excluded. For example, if you have an activity report that includes tasks, appointments and notes, but the user may only want to see tasks, then you can give them option to exclude appointments and notes.

To hide an element, do the following:

1. Name the element—for example put ‘Tasks’ on a strip.
2. In the element choice editor, add a rule using the + button.
3. Enter in a description that makes sense to the user.
4. Enter the name of the applicable element (in this case ‘Tasks’).

The user will be presented with a checkbox list. They can choose to uncheck one or all the named elements.

## Accessing User Input variables

User input variables are put in a dictionary and you can access that dictionary using the ‘userInput’ reserved keypath.

### Accessing User Input variables from F-Script

For example, if we had a date range input with the start date key of ‘start-Date’ we would get that start date with the following code:

```
myStartDate := userInput valueForKey: 'startDate'.
```

You can also use:

```
myStartDate := element valueForKeyPath: 'userInput.startDate'.
```

This code is valid in tokens, in pre and post extraction scripts and in scripts’ filters.

### Accessing User Input variables from a sub-merge

If you use sub-merge tokens, you access the variable in the following way:

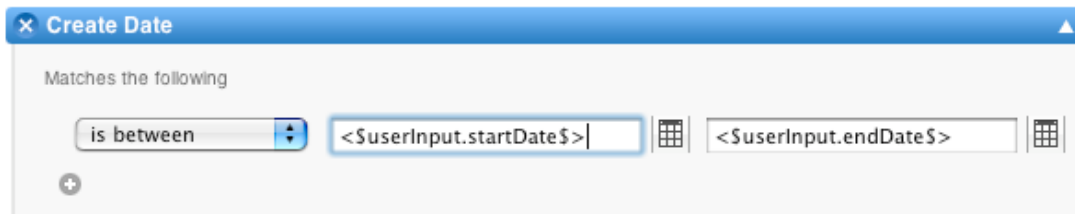
```
<$userInput.startDate$>
```

Those of you familiar with the ‘merge’ report format from Daylite 1.x will be familiar with this notation.

## Accessing User Input variables from a Standard Filter Criterion

You can substitute a user input variable as a value in a criterion using the same pattern as sub-merge.

`<$userInput.startDate$>`



# Scripting

The Report Engine supports F-Script as its core scripting language. F-Script is a smalltalk like, array based language layered on top of Cocoa. These underpinnings give F-Script the ability to access the whole Marketcircle application stack and it gives the Report Engine tremendous power and flexibility.

## Pre and Post Extraction Scripts

On any element, you can add a pre-extraction script and a post-extraction script. Every element is given the opportunity to fetch data (called data extraction). There are some elements that do nothing. Among the others, the element Table can get a list of objects, whereas, the element Text Area gets properties.

The pre-extraction script is executed before the data extraction phase. You can, for example, query the database, file or URL for something, or massage a user input variable. The post-extraction script is executed after an element extracts its data based on the data rule on it and the basic composition has been done. For example, on Dynamic Text Area, you may want to set the font color to red if certain values are out of scope or represent some kind of alert level.

## Script Filter

A script filter is executed during the data extraction phase of a Table or Group By. Its purpose is to get objects from somewhere—most likely the database.

## Script Token

A script token is executed during the data extraction phase of a Dynamic Text Area. The return value should be a property of type NSString, NSNumber, NSDate or NSAttributedString. Formatting and spacing rules are applied after the script has executed and returned its value.

## Dynamic Text as Mathematical Expression

You can turn a whole Dynamic Text Area into one mathematical expression. You can use embedded tokens to execute sub-scripts or fetch properties. In this case, all tokens are executed, the mathematical expression is evaluated, and then formatting and spacing is applied.

## Popup User Input Script

To populate a user input popup menu with choices, you execute a script. The script can be complex or as simple as a single line.

## Reserved variable names

When a script is executed, several special variables are set on it. Be careful not to override these in your scripts.

Variable	Purpose
element reportElement	A reference to the report element such as Table
objectContext	A reference to the Object Context, the gateway to all objects from the database
userInput	A reference to the dictionary of user input variables
arguments, keyPath, generator, previousRe- sults, sys	Reserved

# Variables

Each element can store variables. You can access the variables by calling `reportData` or `data` on an element. The variables container (a dictionary) will not be initialized on elements that do not have data rules (i.e. Box), but you can set a `NSMutableDictionary` on the element in a Pre-Extraction script should you want that Box to store variables. The following elements have the variables container by default (that may not be initialized until after the data extraction phase):

- Report
- Table
- Data Row
- Group By
- Group By Row
- Checkmark
- Color Chip

Dynamic Text Area is a special case as the tokens handle the data work and have no need for a variables container.

## Reserved Variable Names

The following variable names are reserved and must not be overridden:

- `object`
- `objects`
- `document`
- `objectContext`
- `userDefaults`
- `databaseUserDefaults`
- `currentUser`

## Accessing Element Variables

You would touch or access an element variable from a script typically. Since a template can have a tree of elements, you need to realize where the variable you are interested in is stored. For example, you can get to the `currentUser` variable from any element using the following:

```
theUser := element
  valueForKeyPath: 'enclosingReportLayout.reportData.currentUser'.
```

or

```
theUser := element
  valueForKeyPath: 'parent.parent.parent.reportData.currentUser'.
```

\* the number of 'parent' in the keypath is dependent on how deep you are in the hierarchy.

or

```
theUser := element valueForKey: 'currentUser'.
```

\* slower than the other techniques as it tests each element in the parent chain.

or

```
theUser := element enclosingReportLayout reportData
  objectForKey: 'currentUser'.
```

As you can see there are many ways. The preferred way is the first or the last.

# Help and other resources

There are a number of resources available to help you to learn more about the Report Engine:

- ▶ **Report certified gurus Blue Rock & Mozaic Services & Solutions:**  
Blue Rock's website <http://www.daylitehelp.com/> is full of free and paid custom report templates. Author Eric O'Connell of Mozaic Services has written a book for the 'do it yourself' audience and also provides custom report creation services. For more information, please visit [http://www.mozaicsands.com/daylite\\_reports.html](http://www.mozaicsands.com/daylite_reports.html)
- ▶ For a full list of Report certified partners, visit <http://partners.marketcircle.com/partners/index>. Training movies can be found at <http://www.marketcircle.com/help/movies/>.
- ▶ Visit <http://forums.marketcircle.com> to share ideas, tips, and questions with other users. Marketcircle engineers, designers, and support staff also share their knowledge on the forums.