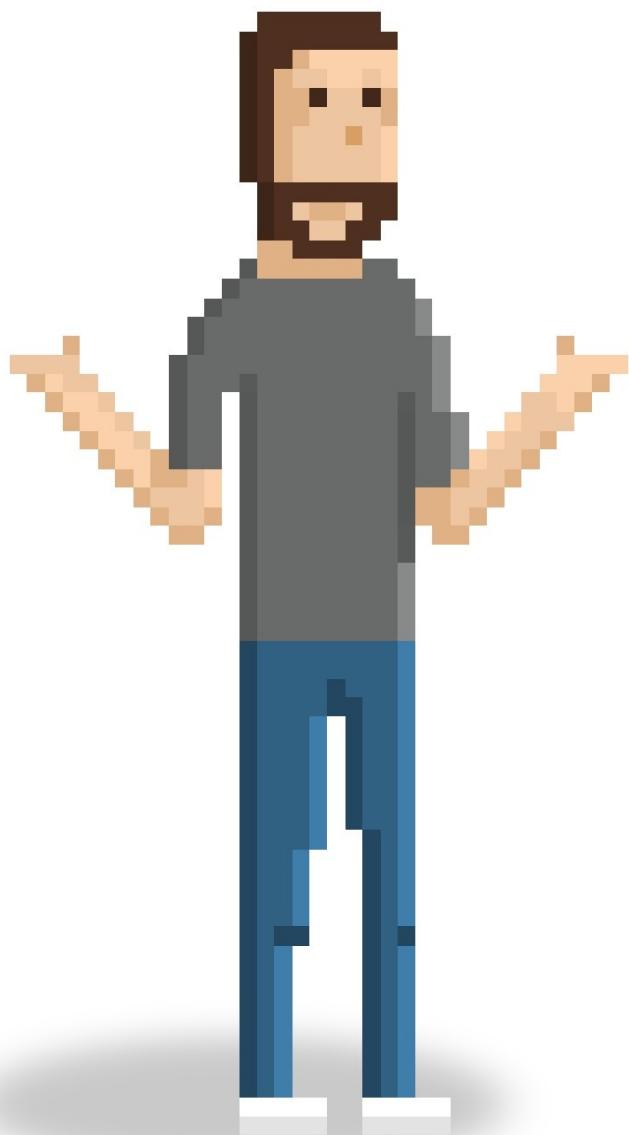


# Using Jelly In ServiceNow

Sal Costa



# Table of Contents

---

1. [Introduction](#)
2. [Jelly](#)
  - i. [Key Concepts](#)
3. [Jelly and ServiceNow](#)
  - i. [UI Pages](#)
    - i. [Page Template](#)
  - ii. [UI Macros](#)
    - i. [Formatters](#)
  - iii. [Other Jelly Scriptable Areas](#)
4. [Jelly Scripting](#)
  - i. [Jelly Tag Library](#)
    - i. [Variables](#)
    - ii. [Output](#)
    - iii. [JEXL Expressions](#)
    - iv. [Conditions](#)
    - v. [Looping](#)
    - vi. [Undocumented Jelly Tags](#)
  - ii. [Glide Tag Library](#)
    - i. [Using References](#)
    - ii. [Scripting Tags](#)
      - i. [Running JavaScript](#)
      - ii. [Scripting Helpers](#)
      - iii. [Using Functions and Macros](#)
      - iv. [Miscellaneous Functions](#)
    - iii. [UI Tags](#)
      - i. [HTML Elements](#)
      - ii. [Form Elements](#)
      - iii. [List Elements](#)
      - iv. [Choice Lists](#)
      - v. [Content Management Components](#)
      - vi. [Knowledge Base Components](#)
      - vii. [Chart Components](#)
      - viii. [Extended UI Components](#)
      - ix. [Other ServiceNow Components](#)
    - iv. [Undocumented Tags](#)
    - iii. [Debugging](#)
      - i. [Debugging Tags](#)
  5. [Advanced Usage](#)
    - i. [Glide and Jelly JavaScript Objects](#)
      - i. [Executing Jelly](#)
      - ii. [Render Properties](#)
      - iii. [Jelly Contexts](#)
      - iv. [Evaluating Expressions](#)
    - ii. [Creating UI Extensions](#)
      - iii. [Decorators](#)
      - iv. [Overriding Default Templates](#)
      - v. [System Properties](#)
  6. [Jelly Examples](#)
    - i. [Core Concepts](#)

- i. [Multiple Phases](#)
  - ii. [Fizz Buzz](#)
  - iii. [Fibonacci Numbers](#)
  - iv. [Bubble Sort](#)
  - v. [Calling Functions](#)
  - vi. [Towers of Hanoi](#)
  - ii. [Static Page](#)
    - i. [Hello World](#)
    - ii. [Simple Calculator](#)
    - iii. [Output A List](#)
    - iv. [Simple Form](#)
    - v. [Form Using Macros](#)
  - iii. [Dynamic Page](#)
    - i. [Dynamic Calculator](#)
    - ii. [Advanced Form](#)
  - iv. [Multi-Page Application](#)
    - i. [Simple Portal](#)
    - ii. [Advanced Form](#)
  - v. [AJAX Application](#)
    - i. [AJAX Form](#)
    - ii. [Form Widgets](#)
    - iii. [AJAX Form Widget](#)
  - vi. [Custom Formatter](#)
    - i. [Custom Element](#)
    - ii. [Form Widgets](#)
  - vii. [Mobile Page](#)
    - i. [Mobile Only Page](#)
    - ii. [Responsive Page](#)
  - viii. [Frameworks and Libraries](#)
    - i. [Datatables List](#)
    - ii. [Twitter Bootstrap](#)
7. [Index](#)
  - i. [Jelly Language Reference](#)
  - ii. [JavaScript Objects Reference](#)
  - iii. [Licenses and Copyrights](#)

# Using Jelly In ServiceNow

---

Sal Costa

Copyright © 2015 Sal Costa

---

I never thought I'd ever write enough to fill a book, now that I have I am sure I never will again. So, I'd better make all my dedications now...

To my parents, for encouraging my geekiness, to my love, Jadie, for tolerating it, and to my son, Domenic, who hopefully inherited it.

---

## Introduction

I have been using ServiceNow and Jelly for well over 5 years and sadly there is as much documentation for Jelly now as there was when I started. It is because of this lack of documentation that many people avoid or completely dismiss creating custom pages and functionality in ServiceNow. This is the book I wish I had on day one.

It is my hope to turn the perception of the language around, to get more people to use it and hopefully revive some interest in the language.

## Using This Book

As I mentioned, this is the book *I* wanted. If you are like me you probably will want to use this book in different ways depending on the circumstance.

The first few sections can be read straight through. They are meant to give an overview of the language, its capabilities and best practices on how to use it correctly in ServiceNow.

I also have provided numerous full-page examples with explanations, which you may use as a base for your own pages.

Lastly, I am including a full language reference, which will provide an exhaustive list of tags and their usage, again with examples, for when you simply need a reference.

Throughout the book I will try to add my own best practices and commentary like this.

Please keep in mind that the only references I used are the language itself, the few pages within ServiceNow and examples found on the internet. Almost all of the examples in this book come through thousands and thousands of tests. I fully expect changes and errors to be discovered and will do my best to publish updates quickly.

## Code Samples and Usage

All code in this book is provided for educational purposes only. Most is of my own creation and may be used without permission or attribution. Hopefully it helps you create something awesome.

Some code examples cite or use external libraries. These belong to their respective owners and have their own licenses and may require attribution.

**Please make sure to take care when using these examples and never make changes directly on a production system!**

If you would like to use material from this book as part of a training course or you feel your use goes beyond fair use please contact me at costas0811@hotmail.com.

## Jelly Defined

---

Jelly is an engine developed by the Apache Foundation that uses Java and XML to execute XML code. It can take that source Jelly XML and generate HTML, text or XML. It is extensible and has a very robust expression language called JEXL built-in.

ServiceNow uses Jelly to generate it's User Interface. Almost all pages, widgets, reports, lists and forms within ServiceNow exist as Jelly XML somewhere on the file system or in the database. ServiceNow itself is extensible using Jelly to create custom Pages and Macros which can be used throughout the system.

# Key Concepts

---

## XML Based

The language itself (referred to sometimes as Jelly XML) is a set of XML tags which build up the functions of the language. Each tag might be considered a function and its attributes and body are its parameters. Unlike other programming languages there are no brackets to indicate different scopes, instead in Jelly a functions body may be its scope.

To compare to a language like JavaScript, invoking a function is done by using the name and passing parameters in parentheses `myFunction(name, age);`, in XML based languages invoking a function is done by using the tag and passing parameters either by attribute or within the body of the tag, `<myFunction name="John" age="21"></myFunction>`

### Anatomy of a Tag

Since all functions in Jelly are XML based tags it will be useful to discuss some of the vocabulary used to refer to parts of the tag.

Here is a simple tag:

```
<g:evaluate jelly="true">
    // Some Script
</g:evaluate>
```

- `g` - The namespace of the tag, this will match up to a defined namespace.
- `evaluate` - The tag name, this defines what the function will be
- `jelly` - This is an attribute, in the context of Jelly this is a way to pass parameters to the function
- "true" - This is a value and is what the attribute is set to. Sometimes values in Jelly can be expressions (discussed below and in later sections)
- `// Some Script` - This is the body of the tag and is another way parameters can be passed to a function. Not all tags support using a body while others require it.
- `..` - This is the closing tag.

I will use this vocabulary fairly frequently in this book so it is good to be fully familiar with it.

## Generates HTML, Text and XML

Jelly takes XML as an input and can generate HTML, plain text or even more XML, which can itself be executed in Jelly. Another way to think of this is when Jelly Tags are executed the result of the execution replaces the Jelly Tag itself.

For example, given a custom Jelly Tag `myName` that takes a parameter name and outputs the name wrapped in an `h1` tag it might look like this.

Jelly XML:

```
<myName name="John"></myName>
```

Result after executing:

```
<h1>John</h1>
```

Many people incorrectly assume Jelly is an equivalent or replacement for HTML. This is definitely not the case, it can only generate it and is not something a Web Browser will recognize.

## Jelly Runs on the Server

The environment to run Jelly only exists on the ServiceNow server. Although the result may be returned to a user as part of a request, the original XML code will not.

Because it generates HTML and runs on the Server, Jelly can be compared to other Server Side scripting languages like PHP. PHP however is the language and it is executed with the PHP interpreter, with Jelly the language is XML and Jelly is the interpreter.

Jelly also has many similarities to templating libraries that other frameworks use. These libraries usually include a subset of what Jelly can do, simple loops and conditions are usually the extent of their built-in abilities. As you will see in this book, Jelly goes much further than just templating.

## Jelly Uses JEXL Expressions

In Jelly XML, JEXL is used to evaluate expressions. JEXL may be used within the body of a Jelly XML tag to output some value, or in an attribute to specify that value.

JEXL expressions in ServiceNow are wrapped in either  `${}` or  `$[ ]` depending on which phase the JEXL should be run. The actual expression within the brackets may be any valid JavaScript. The JavaScript be evaluated when it is encountered by Jelly and the result will replace the JEXL, very similarly to how Jelly Tags are replaced by their result.

## Jelly Is Extensible

Additional tags can be created and included as Tag Libraries. These are written in Java and must be deployed on the Server. ServiceNow has created the Glide Tag Library which includes many functions to simplify scripting, execute JavaScript in the ServiceNow context, create custom UI elements and more. At this point it is not possible for users to create and deploy these custom libraries but this is not a problem. ServiceNow has created methods to create your own re-usable components which we will learn about in later chapters.

## Angular Is Not A Replacement for Jelly

Angular is strictly a client-side framework that has many overlapping functions with Jelly, however this does not mean it is a full replacement for Jelly. Many other applications built around Angular still make use of server-side templating engines to do some of the heavy lifting. It is also important to keep in mind Angular is only usable when building AJAX driven applications. If the intent is only to build a simple page or application which does not need AJAX it is not necessary to use Angular and may actually negatively impact maintainability and increase development time.

## Jelly and ServiceNow

---

Jelly XML is used to generate almost all of the UI of ServiceNow. The source for the core UI of ServiceNow exists as files on the server but ServiceNow also provides ways to extend the platform by allowing developers to create Jelly XML on the platform itself and storing this in tables.

There are several ways to create Jelly in ServiceNow the most common being UI Pages, which are custom pages within ServiceNow, and UI Macros, which are re-usable blocks of Jelly XML that can be included in UI Pages and Forms. ServiceNow also uses some components of Jelly in Notifications, CMS Blocks and even has a way to programmatically run Jelly so developers can use Jelly anywhere they want.

In addition to the creation of the Glide Tag Library mentioned in the last section, ServiceNow has also customized the way Jelly executes by creating the concept of multiple phases. Jelly XML is passed through the Jelly engine two times, the result of the first pass is used in the second pass. This allows for static portions of the page to be cached resulting in better performance. It also allows for Jelly XML in the first pass to generate Jelly XML for the second pass.

Within Jelly XML all Jelly tags use a namespace. The namespace denotes which phase but also whether or not the tag is custom or part of the core Jelly Standard Tag Library (JSTL). Tags with the `j` and `j2` namespace are JSTL tags, tags with `g` and `g2` (`g` meaning Glide) are custom tags created by ServiceNow.

JEXL expressions can be used in specific phases as well. In Jelly, JEXL expressions are started with \$ and wrapped in curly braces,  `${}`. In order to allow the JEXL expressions to be used in a specific phase, ServiceNow looks for expressions wrapped in square braces  `${[]}` and converts them after the second pass.

One easy way to remember which phase the brackets are associated with is that curly brackets come to a single point `->{`, and so they are Phase one.

## UI Pages

---

A UI Page is a record in ServiceNow that uses Jelly and JavaScript to generate a custom page. It has three main components: HTML, Client Script and Processing Script.

The HTML is actually Jelly XML. When a user makes a request for the UI Page, ServiceNow takes that Jelly XML and executes it twice (once for each phase). The Client Script, which is just client side JavaScript is added to the result of the Jelly execution. That complete page is returned to the user.

The reason for breaking up the components is Jelly must be valid XML in order to be executed. JavaScript has many characters such as < and & which will break the XML. In order to make it easier to code JavaScript (and to separate your concerns) the Client Script is a separate field and is added after the Jelly execution.

JavaScript may still be used directly in the HTML portion but extra steps must be taken to avoid breaking the XML. These techniques will be covered in the Debugging section of this chapter.

The Processing Script is actually server side JavaScript and can be used to process a form submission from the page. Processing Scripts can access all submitted form values and are useful for simple forms. In the Jelly Examples chapter, we will cover using Processing Scripts and Jelly itself to process form submissions.

## Access Controls and Public Pages

---

There are two ways to access a UI Page, the first is to use the Try It button on the UI Page record which redirects to `instance.service-now.com/ui_page.do?sys_id=abc123...`.

The page may also be accessed directly by its name by going to `instance.service-now.com/page_name.do`. Which is why it is very important to never use spaces or special characters in the UI Page name.

Because UI Pages may show sensitive data or provide admin-like abilities, they have Access Controls applied to them to prevent unauthorized use. To do this an ACL of type `ui_page` must be created and the name of the page used. The rest of the ACL can be configured like a standard ACL using Scripts and Roles to restrict or grant access.

Normally to view a UI Page a user must at least be logged into the system, but sometimes it is desirable to create a public facing page. In this case Public Pages (`sys_public`) may be used to allow ServiceNow to serve the page to unauthenticated users. To do this navigate to the `sys_public` table and create a new entry with the page name as the page.

## Creating a UI Page

---

To create a UI Page you must be logged in as an Administrator. Navigate to the UI Pages module and click Create New. Remember to use only letters, numbers and underscores in the name.

The basic template of the page is automatically generated in the HTML field for you.

# Basic Page Template

This is the basic template that gets generated with every UI Page and Macro.

## UI Page

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
    Page Content Goes Here
</j:jelly>
```

## Walkthrough

Since Jelly is XML it must start with:

```
<?xml version="1.0" encoding="utf-8" ?>
```

This is called the Prolog and it declares the document as XML and specifies the encoding. It must come before the root element `j:jelly` and be the first element in the page. You may add comments after the prolog but nothing can be added before it, even a comment, or the XML will be invalid and the page will not process.

The `jelly` tag indicates the start of where Jelly Processing begins and is usually directly after the prolog.

```
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
```

The `jelly` tag specifies which namespaces are used in the document and to which Tag Library they reference . In almost all cases these should be left as is. Even though the phase 2 namespaces appear to be set to `null` , they are still valid. They only need to be set to make the XML document valid.

Additional name spaces could be specified as well but they would still need to map to an available library. Jelly ships with many Tag Libraries however they are disabled and are not accessible.

The `jelly` tag also takes an attribute `trim` which removes whitespace from the result of the execution. In practice that is any space at the beginning or end of the body of any tag.

```
<span> This has extra whitespace </span>
```

You may have noticed that changing the value of `trim` doesn't have any effect and that is because the value is not passed on to the second phase. The default value is true not false, so any whitespace ends up being removed in the second phase.

After the `jelly` tag is the page content. This can be any mix of Jelly, HTML and JavaScript.

```
Page Content Goes Here
```

Finally, after the page content the closing Jelly tag. This tells Jelly where to stop executing.

```
</j:jelly>
```

## UI Macros

---

A UI Macro is a re-usable piece of Jelly. They can be included in Jelly such as UI Pages and other UI Macros and they can be added to standard ServiceNow forms as a Formatter. Just like UI Pages they have the same standard setup and execute Jelly in two Phases.

To put it in JavaScript terms a macro is like a function. Any code that is ever repeated such as page headers or footers, forms or UI Elements can be built as a UI Macro to make creating pages easier.

Like functions, parameters can be passed to UI Macros so re-usable code can be generalized and used in many situations. In addition to parameters entire sections of Jelly can be passed into a Macro, the result being the content is wrapped in the Macro.

*These methods will be covered in the Using Functions and Macros section of this chapter*

In ServiceNow most UI Macros, like UI Pages, live as records in the UI Macros Table. Macro names are actually used to call the macro (similar to how a UI Page's name is used as its URL) and so it's a best practice to name them as if they were a file, replacing spaces with '\_' and avoiding special characters.

Some UI Macros that are created by ServiceNow live as files on the server, it is possible to use these Macro files but not change them. It is also possible to override these Macro files by creating a UI Macro record with the same name but this is extremely dangerous and if care is not taken you may end up breaking core system functionality.

ServiceNow also has many UI Macros included out-of-box that can be called and used in custom pages and macros, these will be covered in the Language Reference Chapter.

## Creating a UI Macro

---

To create a UI Macro (like a UI Page) you must be logged in as an Administrator. Navigate to the UI Macros module and click Create New. Remember to use only letters, numbers and underscores in the name.

The basic template of the macro is automatically generated in the XML field for you. It is exactly the same as the UI Page Macro.

# Formatters

Formatters link UI Macros to ServiceNow forms. Creating them is fairly simple, once you have a UI Macro made you can create a Formatter by going to the "Formatters" Module (you must be logged in as an administrator) and creating a new Formatter.

- The Name of the form should be a friendly name and identify the Macro.
- The Formatter should be the name of the UI Macro with `.xml` appended to the end.
- Table should be the table you intend to use the Macro on.
- Type should be Formatter
- Active should be true

UI Formatter	
Name:	<input type="text" value="Friendly Name"/>
Formatter:	<input type="text" value="my_macro.xml"/>
Table:	<input type="text" value="Incident [incident]"/>
Type:	<input type="text" value="Formatter"/>
Active:	<input checked="" type="checkbox"/>
<b>Submit</b>	

Once the Formatter is created you may add it to your tables Form layout in the Personalize Form editor. Formatters will be located after the field names in the list.

If the Formatter record itself ever changes it needs to be removed and re-added to the form. This means especially the name of the UI Macro. Changing the content (Jelly XML) does not require you to remove and re-add the macro.

**The Custom Formatter Section in the Examples Chapter will have several types of Formatters**

## Other Jelly Scriptable Areas

---

Aside from UI Pages and Macros Jelly and especially JEXL are used in other areas of the system.

Within the CMS Jelly is used in **Dynamic Content Blocks**. These are similar to UI Macros as they are only meant to be portions of pages and not entire pages itself (although you technically could). One interesting aspect of Dynamic Content Blocks is their ability to disable the Second Phase.

Another area where Jelly is used is in the **Content Type** record, also under the CMS. This record uses Jelly for both the **Summary Template** and **Detail Template** fields. **List Definitions** are also another variation on UI Macros used in the CMS.

JEXL is used outside of Jelly Pages in **Email Notifications**, both in the Notifications and in Templates. Each expression using  `${}` and  `${[]}` are JEXL expressions and are the same as would be found in Jelly Pages.

JEXL is also used for **Outbound Web Service Requests** as a way to populate variables in the body of the request.

## Scriptable Extensions

Using UI Macros you can also extend Jelly and create your own UI Components. This allows you to create components you can use across many pages and other macros. This is covered in the Advanced Usage section.

Another advanced technique covered is Field Decorations, or using UI Macros to extend the functionality of standard field types. *Similar to the VIP functionality on Incident.*

## Jelly Scripting

---

This chapter covers the basics of Jelly Scripting and also documents the Tag Libraries available in ServiceNow.

A quick search will reveal that Jelly has dozens of available libraries but in ServiceNow we are limited to the core Jelly tags and the Glide Tag library. Each has it's own dedicated section here.

## Jelly Tag Library

---

This section covers basic scripting abilities in Jelly XML which are part of the Jelly core Tag Library. This includes creating and using variables, looping, checking for conditions and output. Glide specific functions will be covered in the next section.

Remember all these tags are prefixed with a `j` or `j2`

# Variables

## Setting Variables

There are several ways to create variables in Jelly, the first which I will cover here is the `set` tag. `set` will set a variable to the result of a given expression.

```
<j:set var="jvar.firstName" value="Domenic" />
<!-- jvar.firstName is set to Domenic -->
```

ServiceNow has modified the way `set` works so that the variable name in `var` must begin with `jvar_`. This is most likely to avoid colliding with existing variables in the scope, but it is important to remember.

In the first example the value was set to a literal string but the `value` property can be a **JEXL** expression.

```
<j:set var="jvar.firstName" value="${current.first_name}" />
<!-- jvar.firstName is set to the value of current.first_name -->
```

The documentation for `set` indicates additional properties such as, `target`, `property`, `scope` that do not work in ServiceNow. `target` and `property` were intended to set a property on an object, however these do not work, and it is possible to do the same function using Glide Tags (covered in the next section).

`scope` is intended to set the scope where the variable will be set, but since there is only one scope available this is not needed.

Like most tags `set` can be used in either phase. Variables set with the `set` tag are available in their current phase only. This is not true of other variables in the scope such as those created globally or using an `evaluate` tag.

In this example the second `j:set` copies the value of `jvar.firstName` but the `j2:set` fails to set its variable correctly because `jvar.firstName` is not in Phase 2.

```
<j:set var="jvar.firstName" value="Domenic" />
<j:set var="jvar_copyFirstName" value="${jvar.firstName}" />
<!-- jvar_copyFirstName is set to the value of jvar.firstName -->
<j2:set var="jvar.firstName2" value="[jvar.firstName]" />
<!-- jvar.firstName2 is set to null -->
```

The `set` tag can also set its value from the body of the tag instead of the `value` attribute.

```
<j:set var="jvar.fullName">
${{
    var firstName = "John";
    var secondName = "Smith";

    firstName+ " " + secondName
}}
</j:set>

<!-- jvar.fullName is set to John Smith -->
```

Lastly, when setting the value of a variable using the body of the `set` tag it is possible to control whether or not text is escaped, converting `<` and `>` to `&lt;` and `&gt;`. This is used in cases where the output should not be treated as HTML but rather plain text. For instance, when trying to display HTML code instead of rendered HTML.

This can be tricky because the System Property `glide.ui.escape_text` will force the parser to use its value. If `glide.ui.escape_text` is true when the value is output `<` and `>` will always be escaped. This makes it difficult to output and render HTML from JEXL Expressions in UI Pages.

The `set` here is working properly, however before the text is responded `<` and `>` are escaped, effectively ignoring the `encode` attribute.

```
<j2:set var="jvar_string" encode="false">
    <h1>Hello World</h1>
</j2:set>

${jvar_string}
```

If `encode` was set to `true` and `glide.ui.escape_text` set to `false` this code would still output a rendered header tag with Hello World.

In almost all circumstances I set `glide.ui.escape_text` to `false` without issue. It is meant to provide an extra layer of security in preventing Cross Site Scripting attacks but these types of attacks are rare and can be prevented in other ways. To me the ability to return HTML is useful enough to allow disabling this. As with any security related property you should do a full analysis of how it may affect your system. Any areas where users are able to specify code that will be returned to other users needs to be locked down to prevent any malicious activity.

## Creating Lists

Jelly has the ability to create Array Lists which is just a Array that can be used natively in Jelly. This is done using the `j:useList` tag. It takes a `var` attribute like `set` and an `items` attribute which should contain an expression returning the list.

This example creates an Array List and then adds an item to the list using `j:invoke` (which calls a method on an object) and `j:arg` (which passes an argument to `invoke`).

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
    <j:useList var="jvar_animals" items="${['mouse','cat','dog']}"/>
    jvar_animals has ${jvar_animals.size()} items <br/>
    <j:invoke method="add" on="${jvar_animals}">
        <j:arg value="rabbit" />
    </j:invoke>
    jvar_animals has ${jvar_animals.size()} items
</j:jelly>
```

Lists have many built in functions which can be used in Jelly. The full documentation is at: [ArrayList](#)

## Output

```
jvar_animals has 3 items
jvar_animals has 4 items
```

## Deleting Variables

In some circumstances you may want to remove a variable that was created within the Jelly context. To do that Jelly has the `j:remove` tag. This tag takes an attribute `var` which is the name of the var to remove.

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
    <j:set var="jvar_name" value="Domenic" />
    First: ${jvar_name} <br/>
    <j:remove var="jvar_name" />
    Second: ${jvar_name}

</j:jelly>
```

## Output

```
First: Domenic
Second:
```

## Creating Temporary Scopes

Jelly has the ability to create scopes with the `j:scope` tag. All variables created within the body of a `scope` tag will exist only within that tag.

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
    <j:scope>
        <j:set var="jvar_name" value="Domenic" />
        In Scope: ${jvar_name} <br/>
    </j:scope>
    Out of Scope: ${jvar_name}

</j:jelly>
```

## Output

```
In Scope: Domenic
Out of Scope:
```

## Using Scriptable Classes

`j:invokeStatic` will call a method on a Class as long as the class does not require the use of its constructor. For instance the class GlideUser works because you can call methods on it directly but GlideRecord would not since you must initialize it first with the constructor then call methods on it. `invokeStatic` takes three attributes, `var` which is the variable to store the result in, `scriptableName` which is the Class Name and `method` which is the method on the class to call.

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
    <j:invokeStatic var="jvar_by_name" scriptableName="GlideUser" method="resolveNameFromSysID">
        <j:arg value="6816f79cc0a8016401c5a33be04be441"/>
    </j:invokeStatic>
    Name: ${jvar_by_name}
</j:jelly>
```

## Output

Name: System Administrator

# Output

---

## Static Text

Any text, HTML or XML that is not a Jelly Tag within a file will be returned as the output of the file when it is run. The only restrictions are that it must be valid XML, meaning the characters &, < and > must be escaped if they are in the output and not part of an HTML or XML tag.

### Jelly

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
    <h1>Hello World</h1>
    This is some sample output.
</j:jelly>
```

### Output

```
<h1>Hello World</h1>
This is some sample output.
```

---

If `glide.ui.escape_text` is `false` the following example would be invalid because in XML `&` is the beginning of an entity, which is not specified here.

### Jelly

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
    This is some sample output & this is some more sample output.
</j:jelly>
```

### Output

```
Error Message Error during script processing
Error Message The entity name must immediately follow the '&' in the entity reference.
Error Message line: 5, column: 33
Error Message This is some sample output & this is some more sample output.
```

The XML parser is expecting a code after the `&` to indicate which entity to use. The XML specification predefines 5 entities which may be used.

Name	Character	Description
quot	"	Double Quotation Mark
amp	&	Ampersand
apos	'	Apostrophe
lt	<	Less-than Sign
gt	>	Greater-than Sign

The characters & and < will immediately break XML since the parser would be expecting an entity and tag name respectively. Characters ',' and

Here is the same script with the full entity specified.

### Jelly

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
    This is some sample output & this is some more sample output.
</j:jelly>
```

Again the output is not as expected.

```
The entity name must immediately follow the '&' in the entity reference.
```

This is not an error from Jelly but most likely the response object was expecting valid XML, because the & was converted to & before responding the output to the user, the response XML is not valid.

With `glide.ui.escape_text` is `false` in order to output an & symbol you must create an entity which can persist through the Jelly Execution. Since the & gets converted to & we can add amp; to the end, &amp;amp;. After the Jelly Execution this will get converted to just & and we will finally get the correct output;

### Jelly

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
    This is some sample output &amp;amp; this is some more sample output.
</j:jelly>
```

### Output

```
This is some sample output & this is some more sample output.
```

## Predefined Variables

ServiceNow has created some predefined Jelly variables to simplify outputting special characters. They are \${AMP}, \${GT}, \${LT}. Notice the use of the Phase 2 square brackets to evaluate these expressions. They will work in Phase 1 as \${AMP}, \${GT}, \${LT} but may have unexpected results. If the result of the Phase 1 execution is not valid XML an error will be thrown.

### Jelly

```
Here are some tricky characters ${AMP}
```

### Output

```
The entity name must immediately follow the '&' in the entity reference.
```

**Jelly**

```
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
    Here are some tricky characters $[AMP], $[LT], $[GT]
</j:jelly>
```

**Output**

```
Here are some tricky characters &, <, >
```

## HTML Entities

Finally, what if we want to actually output an HTML Entity, that is some special character in the browser that is specified with `&####;`. We already know that to return an & we should use `$[AMP]`, so to use an entity such as ©, the output should contain `$[AMP]copy;`. Fully expanded out this would be `&amp;copy;`

**Jelly**

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
    I need to use a copyright symbol: &amp;copy;
</j:jelly>
```

Walking through the execution of this code after the Jelly Execution the output is:

```
I need to use a copyright symbol: &copy;
```

After the response object parses the Jelly response:

```
I need to use a copyright symbol: &copy;
```

What the user sees:

I need to use a copyright symbol: ©

Another common HTML entity to use is `&ampnbsp` and there is a Jelly shortcut for this as well, `$[SP]`.

It is important in Jelly where whitespace is added directly before or after a tag. This code will incorrectly trim the whitespace (even though we told Jelly not to) before the `<strong>` tag.

**Jelly**

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
    Hello <strong>World</strong>
</j:jelly>
```

**Output**

```
Hello<strong>World</strong>
```

So to fix it you would add \${SP] where necessary.

## Jelly

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
    Hello${SP]<strong>World</strong>
</j:jelly>
```

## Output

```
Hello<strong>World</strong>
```

## Preserving Whitespace

Whitespace can be important especially when including a space in between HTML tags. As we saw in the `jelly` tag there is a `trim` attribute which will trim all whitespace globally in a page. We also know that the default of `trim` is true and so, even though it is configurable for Phase 1 in the root `j:jelly` tag, unless it is specified for the second phase tag (`j2:jelly`) whitespace will be removed. To get around this there is a `j:whiteSpace` tag which will preserve whitespace in its body. If you run this in the first phase the whitespace will still be removed in the second phase, so this tag only makes sense to use in the second phase.

This example shows, normal whitespace treatment, using `j:whiteSpace` and `j2:whiteSpace`.

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">

    Whitespace <strong>trimmed</strong>. <br/>
    <j:whiteSpace>
        Trimmed <strong>here</strong> too. <br/>
    </j:whiteSpace>
    <j2:whiteSpace>
        Not <strong>here</strong>.
    </j2:whiteSpace>

</j:jelly>
```

## Output

Whitespacet**trimmed**.

Trimmed**here**too.

Not **here**.

# JEXL Expressions

Jelly also has support for JEXL, an expression language that lets us combine some context with an expression and return some result. These JEXL expressions can be used as output, within the body of a tag, or as the value of an attribute. The expressions use JavaScript although not all variables in context will have been created using JavaScript, some might have been created with Java and so not all the functions are always available.

Here is the basic example of outputting a variable using JEXL.

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
    <j:set var="jvar_message" value="Hello World!" />
    <h1>
        ${jvar_message}
    </h1>
</j:jelly>
```

ServiceNow allows JEXL expressions to be used in both phases. For Phase 1 the standard curly brackets, `{}`  can be used to indicate a JEXL Expression. For Phase 2, square brackets `[]`  are used.

## Context

The context in JEXL is all of the object which are available to it. In ServiceNow this includes things like the Glide API, `current`, `RP` (Request Parameters) and Global Business Rules. Variables already created in a Jelly Page are also part of the context and are usable in JEXL Expressions.

Remember since we are using JavaScript the `this` variable at a global level will return every object and function available.

Since the context will also include Jelly Variables the Phase will be important when knowing which variables are available. Variables created in Phase 1 will be available in both phases. Variables created in Phase 2 will only be available in Phase 2.

Understanding the brackets and their associated Phase is not usually what causes issues. Issues are caused when code is written and then the phase of Tags are changed but the brackets are not. This will be covered more in the Debugging Section of this Chapter.

## Expressions

The actual expression may be any combination of literal strings, numbers, operators and variables in context. Expressions may contain some JavaScript and although it is possible to make expressions multi-line, it is best to keep them simple and do complicated work elsewhere. Expressions are expected to return a result and whatever the result of the expression is what will be returned.

Here are some example expressions that use some of the features of JEXL.

### String Expressions

```
Literal String: ${ 'Hello World' }
String Concatenation ${ "Foo" + "Bar" }
Compare String ${ jvar.firstName.equals("John") }
Another Compare String ${ jvar.firstName == "John" }
```

## Variable and Function Expressions

```
Output a variable ${ jvar.firstName }
Output the result of a function ${ gs.getSession() }
Output the length of a variable ${ size(jvar.firstName) }
Check for empty value ${ empty(jvar.empty) }
```

`size()` is a function of JEXL, if you try to do `.length` you will find it does not always work. This is because JEXL is a combination of languages and not exactly JavaScript. Variables created in JavaScript will have `.length`, variables created in Jelly will not and will need to use `size()` to check length.

`empty()` is also a function of JEXL and it checks for empty variables.

## Boolean Expressions

```
Comparisons Equal ${ true == true }
Comparisons Not Equal ${ true != true }
Comparisons Greater Than ${ 10 > 100 }
Comparisons Greater Than or Equal ${ 10 >= 100 }
Comparisons Less Than ${ 10 < 100 }
Comparisons Less Than or Equal ${ 10 <= 100 }
Not ${ !true }
Or ${ true || false }
And ${ true && false }
```

In Jelly Pages the Less than and Less than or equal to breaks the page and must be inverted (`a < b` becomes `b > a`) since XML will expect `<` to begin a tag. In documentation for JEXL it is indicated that `gt`, `gte`, `lt`, `lte` are available but this seems to be for a later version than what is included in ServiceNow because those operators fail to return a result.

Likewise the And operator `&&` of course breaks XML as well and any comparison using this operator will need to be re-written to accomodate this.

Of course a function could always be written and added globally to compare two values if it is necessary.

## Mathematical Expressions

```
${ 2 + 2 }
${ 31 - 14 }
${ 10 / 2 }
${ 80 * 14 }
${ 42 % 5 }
```

These should look familiar.

## Multi Line Expressions

While it is possible to split up an expression to multiple lines, it decreases readability and there are much better ways, as we will see, to do this. Additionally part of the reason to break up an expression on to multiple lines would be to use `if` and `for` blocks. JEXL supports these types of functions but the closing `}` will close the expression and the expression will not evaluate properly.

```
Multi-line Expression ${ 
    var myVars = [];
    myVars.push("foo");
    myVars.push("bar");
```

```
myVars.push("frob");
myVars.push("nozzle");
myVars.join("-");
}
```

## Other Documented JEXL Features

There are a number of other features documented about JEXL but not everything works in ServiceNow. This may be partly due to the fact that the version in ServiceNow does not match the documented version and partly that those features just simply do not make sense in ServiceNow. Functions such as the `new` function is part of JEXL but doesn't really work in ServiceNow nor would it even be useful.

## Evaluating Expressions

Jelly also has and `expr` tag built in that evaluates an expression and effectively does the same thing as outputting the value on a page.

```
<j:set var="jvar_initial" value="${1}" />
${jvar_initial}
<j:expr value="${jvar_initial+1}" />
<j:out value="${jvar_initial+2}" />
```

### Output

```
1
2
3
```

In ServiceNow `expr` is synonymous with `out`, they each map to the same Tag.

## Conditions

We have already seen how JEXL expressions can be used to make comparisons. Those expressions can be combined with conditional tags to create conditionalized blocks within a Jelly Page. The first tag which we can use to make comparisons is the `j:if` tag.

The `j:if` tag has an attribute `test` which expects a JEXL expression. The result of that expression controls whether or not the body of the `if` tag gets executed.

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
    <j:set var="jvar_showHello" value="true" />
    <j:if test="${jvar_showHello == true}">
        Hello World.
    </j:if>
</j:jelly>
```

This example does a simple check to see if `jvar_showHello` is true and if so it displays "Hello World." Any text within the `if` would be output as well as any Jelly Tags, executed.

The `if` in Jelly does not have any concept of additional cases or a default else condition, so it is good if you need to check one condition but if you have multiple cases you can use the `j:choose` or `j:switch` tags.

`j:choose` should contain one or more conditions as `j:when` tags. Each `j:when` tag takes a `test` attribute, which expects an expression just as `j:if` does.

You may have as many unique `when` blocks as you want in a `choose` but only the first true block will be executed. There may also be a `j:otherwise` tag which specifies a default block to be executed if no other condition matches.

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">

    <j:set var="jvar_showMessage" value="Hello" />

    <j:choose>
        <j:when test="${jvar_showMessage=='Hello'}">
            Hello
        </j:when>
        <j:when test="${jvar_showMessage=='Goodbye'}">
            Goodbye
        </j:when>
        <j:when test="${jvar_showMessage=='Hello'}">
            This will never be displayed.
        </j:when>
        <j:otherwise>
            Not Sure
        </j:otherwise>
    </j:choose>

</j:jelly>
```

`j:switch` is very similar to `choose` however the setup is different and with `switch` you may only compare the value in each case, you may not have unique comparison for each. `switch` will also continue to check the various cases unless a `j:break` is encountered.

The `j:switch` tag takes an attribute `on` which contains the expression to compare to the value of each case. Within the `switch` tag you may have an unlimited amount of `case` tags which contain `value` attributes on which to compare the `on` value to. There may also be a `j:default` block as well which will execute if no other cases where executed.

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">

    <j:set var="jvar_showMessage" value="Hello" />

    <j:switch on="${jvar_showMessage}">
        <j:case value="Hello">
            Hello
        </j:case>
        <j:case value="Goodbye">
            Goodbye
        </j:case>
        <j:case value="Hello">
            Hello Again.
        </j:case>
        <j:default>
            Not Sure
        </j:default>
    </j:switch>

</j:jelly>
```

## Output

```
Hello Hello Again.
```

In this example each case was evaluated and every value which matched was executed. As with other languages, switch cases may be grouped together so multiple cases do not need to be re-written for each matched value. This is done with the `fallThru` attribute, which when set to `true` will instruct Jelly to keep executing cases until either a break or `fallThru` is set to `false`. For instance let's rewrite this JavaScript `switch` in Jelly.

## JavaScript

```
switch (greeting) {
    case "Hello":
    case "Hi":
    case "Hey":
        response = "Hey there!";
        break;
    case "Goodbye":
    case "Bye":
        response = "Goodbye!";
        break;
    default:
        response = "How are you?";
}
```

## Jelly

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">

    <j:set var="jvar_greeting" value="Hi" />

    <j:switch on="${jvar_greeting}">
        <j:case value="Hello" fallThru="true" />
        <j:case value="Hi" fallThru="true" />
        <j:case value="Hey">
            Hey There!
            <j:break />
        </j:case>
        <j:case value="Goodbye">
            Goodbye!
            <j:break />
```

```
</j:case>
<j:default>
    How are you?
</j:default>
</j:switch>

</j:jelly>
```

## Output

```
Hey There!
```

# Looping

## While Loops

Jelly provides a couple options for looping the first being `j:while`. This will continue to loop as long as a condition is true. `while` takes a `test` attribute which is an expression that will be evaluated on each loop.

This example will run through the loop once, output the message, then sets the variable `jvar_showMessage` to false, so the loop will not run an additional time.

```
<?xml version="1.0" encoding="utf-8" ?>
<j:set var="jvar_showMessage" value="true" />

<j:while test="${jvar_showMessage}">
    Hello World
    <j:set var="jvar_showMessage" value="false" />
</j:while>

</j:jelly>
```

You can also use a while to execute a loop a finite number of times. In this example the loop runs 10 times.

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">

<j:set var="jvar_showMessage" value="0" />

<j:while test="${jvar_showMessage != 10}">
    Hello World
    <j:set var="jvar_showMessage" value="${jvar_showMessage + 1}" />
</j:while>

</j:jelly>
```

Notice the `set` tag. It may seem clumsy but all that tag is saying is `jvar_showMessage = jvarShowmessage + 1`.

## For Loops

The second loop type in Jelly is the `j:forEach` loop which will loop over an Array or other collection for every item in the list. You can also use a List created with `j:useList`

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
    <j:set var="jvar_values" value="${[1,2,3,4,5]}" />
    <j:forEach items="${jvar_values}" var="jvar_value">
        Hello World: ${jvar_value} <br/>
    </j:forEach>
</j:jelly>
```

## Output

```
Hello World: 1
Hello World: 2
Hello World: 3
Hello World: 4
Hello World: 5
```

You may also leave out the `values` attribute and specify `start`, `end`, and `step` attributes to create a standard `for` loop.

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
    <j:forEach begin="1" end="100" step="2" var="jvar_index">
        ${jvar_index}
    </j:forEach>
</j:jelly>
```

## Output

```
1 3 5 7 9 11 13 15 17...
```

Sometimes it is helpful to know what index you are on in addition to the value. To get that information you can specify a `varStatus` attribute to store the value of the index.

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
    <j:set var="jvar_items" value="${['mouse','cat','dog','rabbit','snake']}' />
    <j:forEach items="${jvar_items}" var="jvar_item" varStatus="jvar_status">
        ${jvar_item} is ${jvar_status} in the list. <br/>
    </j:forEach>
</j:jelly>
```

## Output

```
mouse is 0 in the list.
cat is 1 in the list.
dog is 2 in the list.
rabbit is 3 in the list.
snake is 4 in the list.
```

# Undocumented Jelly Tags

---

Since Jelly is not specifically for ServiceNow there are many tags which are part of the language but have no documented use in ServiceNow. Here is a short list of those tags and their descriptions.

## j :catch

Catches a thrown exception and it may work in ServiceNow, however it seems when an exception is thrown the page will assert that exception anyways, so this is not usable in UI Pages or Macros.

## j :new

Creates a new object of a specified type. Since Packages cannot be used this is unusable in ServiceNow. Objects can be created in `g:evaluate` blocks

## j :parse

Requires an XMLReader which is not accessible in ServiceNow.

## j :setProperties

Sets a property on an object created with `new` which is not usable in ServiceNow.

## j :thread

Runs the Jelly in the body of the tag in a separate thread. This has no effect on UI Pages and Macros.

## j :useBean

Creates an instance of a JavaBeans component. Since no instances of JavaBeans are used it is unknown if this tag is usable or not.

## Glide Tags

---

This section will cover the Glide Tag Library. These are custom tags created by ServiceNow which extend the Jelly to add scripting functionality, introduce code-reuse techniques and output UI Elements. It is as exhaustive as can be based on existing usage and documentation in ServiceNow.

Some tags simply do not have any documentation or usage. For these tags I have experimented and tested as much as possible and include the sample code in the **Undocumented Tags** section. Almost every tag in this section has a very specific purpose that can be achieved using other tags and scripting.

Remember all these tags are prefixed with `g` or `g2`.

# Using References

In the following sections, some tags will be dependent on a "reference" being created. The reference should point to a record (GlideRecord) or a field (GlideElement) and exist in the JavaScript context (Rhino). In order to be properly referenced by the tags.

Some tags allow setting the reference in a `ref` attribute others will simply depend on `ref`. In either case `ref` should be the name of the variable which is a GlideRecord.

For example here I set up a reference and store it as "incident".

```
<g:evaluate var="incident" copyToRhino="true" object="true">
    var gr = new GlideRecord('incident');
    gr.query();
    gr.next();
    gr;
</g:evaluate>
```

In tags where `ref` is available you can specify `ref="incident"` directly like that and when the reference gets evaluated, it will be pointing to the variable "incident".

In cases where the reference is not definable it will be assumed the variable `ref` will hold the name. So it becomes necessary to set that variable before the tag.

It is also very common that instead of directly setting the variable name "incident" that an expression be used.

```
<g:evaluate var="${jvar_ref_name}" copyToRhino="true" object="true">
    var gr = new GlideRecord('incident');
    gr.query();
    gr.next();
    gr;
</g:evaluate>

<j:set var="ref" value="${jvar_ref_name}" />
```

This works because `ref` doesn't actually store the Glide Record object, only the name of the object in context. Each tag will evaluate this reference on its own.

## Using References Between Phases

One commonly used technique used to use references is to output Jelly code from the first phase to execute in the second.

Remember that the reference is just a string, you cannot append methods, for instance this would not work because `ref` is a string. It stores the name of the variable, but not the variable itself.

```
 ${ref.getDisplayValue()}
```

`getDisplayValue` is obviously not defined on a String and this will fail. But if it were modified to output the referenced object in the first phase and the method was called in the second phase, it will work because the referenced object is in both phases.

```
<strong>$[$ref].getDisplayValue()</strong>
```

## Phase 1 Output

```
<strong>$[incident.getDisplayValue()]</strong>
```

This is Phase 2 so you may use `j2` and `g2` tags with this technique.

## Phase 2 Output

```
<strong>INC0010001</strong>
```

## Scripting Tags

---

These are tags that extend the scripting ability in Jelly. They may change the way Jelly works, add new programming functionality or simplify other scripting tasks. They do not necessarily output anything but are used to control the way a page is run.

# Running JavaScript

One of the most used tags in ServiceNow is `g:evaluate` which allows you to run JavaScript in Jelly. The context is server side and the same as the Jelly page itself. This means you have access to all the standard server side objects such as GlideRecord, GlideSystem as well as objects specific to Jelly such as RP (Request Parameters) and any variables created in the Jelly page.

## Evaluating With a Result

The result of `g:evaluate` can be used in a few ways. First an `expression` attribute can be passed to the tag which will be evaluated and stored in a variable which is passed to the `var` attribute. The variable specified in `var` must begin with `jvar_`.

`evaluate` and `set` seem very similar, which should I use? The answer in most cases is `evaluate`. This is because the expression or script will be JavaScript and even though 9 times out of 10 the JEXL expression may look the same it is not. JavaScript will definitely be maintainable and easier to read.

This example will set a variable `jvar_user_id` to the result of `gs.getUserID()`.

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
    <g:evaluate var="jvar_user_id" expression="gs.getUserID()" />
    ${jvar_user_id}
</j:jelly>
```

### Output

```
6816f79cc0a8016401c5a33be04be441
```

If the result of the expression is an object, you must set an attribute `object` to true on the `evaluate` tag.

This example initializes a GlideRecord to a variable `incident_rec` and then uses the same variable in a second `g:evaluate` and finally outputs the `sys_id` of the first record.

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
    <g:evaluate var="jvar_incident_rec" object="true" expression="var incidentRec = new GlideRecord('incident'); incidentRec.query(); incidentRec.next(); incidentRec.sys_id;" />
    <g:evaluate var="jvar_incident_id" jelly="true">
        jelly.jvar_incident_rec.query();
        jelly.jvar_incident_rec.next();
        jelly.jvar_incident_rec.sys_id;
    </g:evaluate>
    ${jvar_incident_id}
</j:jelly>
```

In the second `g:evaluate` we see the second way to pass a script in, the body of the tag. The output variable is set to the last line of the script which should be an expression that returns a value. In this case the variable `jvar_incident_id` is set to the result of `incident_rec.sys_id`.

## Accessing Jelly From `evaluate`

When specifying `jelly=true` in the `g:evaluate` tag you will have read and write access to Jelly variables. These will be

default values and anything created with `set` or another function.

In the last example we saw in the body of the `g:evaluate` that these are referenced within script under the `jelly` object.

## Carrying Variables Forward

Normally variables created through `g:evaluate` do not carry through to the second Phase. You can force this by adding the attribute and value 'copyToPhase2="true"'. This will copy the variable to the second phase.

Re-writing the previous script again to take advantage of this feature and output in Phase 2 instead.

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
  <g:evaluate var="jvar_incident_rec" object="true" expression="var incidentRec = new GlideRecord('incident'); incidentRec.query(); incidentRec.next(); incidentRec.sys_id;">
    <g:evaluate var="jvar_incident_id" jelly="true" copyToPhase2="true">
      jelly.jvar_incident_rec.query();
      jelly.jvar_incident_rec.next();
      jelly.jvar_incident_rec.sys_id;
    </g:evaluate>
    ${jvar_incident_id}
  </g:evaluate>
</j:jelly>
```

The actual mechanism which copies the values to Phase 2 is a tag `g2:copy_to_p2`. The `evaluate` tag will automatically do this but it is possible to use this tag to add a variable to Phase 2. The only problem is the tag requires that the value be compressed with `StringUtil`. Access to this function has been restricted with the Package Call Deprecation change.

If the function was not restricted it might work like this.

### JavaScript

```
new GlideStringUtil().compress("This is a compressed String.");
```

### Output

```
H4sIAAAAAAAAAvJyCxWAKJEheT83IKi10Li1BSF4JKizLx0PQB3a91mHAAAAA==
```

That compressed string can be used as the value of the `g2:copy_to_p2` tag with the `var` attribute set to the intended variable.

### Jelly

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
  <g2:copy_to_p2 var="jvar_message">H4sIAAAAAAAAAvJyCxWAKJEheT83IKi10Li1BSF4JKizLx0PQB3a91mHAAAAA==</g2:copy_to_p2>
  ${jvar_message}
</j:jelly>
```

### Output

```
This is a compressed String.
```

## Evaluating Without a Result

It is possible to run a script which does not return a Jelly variable. This example re-writes the previous script to do just that.

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
    <g:evaluate var="jvar_incident_id" jelly="true">
        var incidentRec = new GlideRecord('incident');
        incidentRec.query();
        incidentRec.next();
    </g:evaluate>
    ${incidentRec.sys_id}
</j:jelly>
```

What is interesting is that even though no specific variable is returned, variables initialized in the `evaluate` tag are usable in the page. Notice the expression `${incidentRec.sys_id}` which references the object created in the `evaluate` tag. These variables are also carried forward automatically, it is not necessary to specify copying them to Phase 2.

## Security with The `evaluate` Tag

It is extremely important to remember that the output of Phase 1 is going to be executed in Phase 2. If you are outputting variables that are sourced from some sort of user input it is important to sanitize the output or only use Phase 2 for that output. In the following example, the Short Description of a record is being output.

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
    <g:evaluate var="jvar_incident_id" jelly="true">
        var incidentRec = new GlideRecord('incident');
        incidentRec.query();
        incidentRec.next();
    </g:evaluate>
    ${incidentRec.short_description}
</j:jelly>
```

This code has a serious flaw however. If a User were to enter Jelly in the Short Description, that Jelly would be output and executed in Phase 2.

Say a User enters "" as the Short Description.

The output of the first Phase is:

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
    <j2:set var="jvar_insecure" value="${'Uh oh!'}" />
</j:jelly>
```

The `j2:set` will get executed in the second Phase. This is known as an Injection Attack and it is easily preventable by making it a practice that outputting user input must occur in the second Phase. That way even if a malicious user tries to inject Jelly it will never get executed in Phase 2 because the output is not what is executed, only the source.

# Scripting Helpers

---

This section covers tags that extend the Jelly language to have more native programming functions and simplify some of the core functionality.

## Iterating Over Records

Using the core Jelly Tag `while` we can iterate over a GlideRecord but there is custom tag `g:for_each_record` which can simplify iterating over GlideRecords. The tag takes in a GlideRecord object in the `file` attribute as an expression. It can also take a `max` attribute to specify the maximum number of iterations to run.

Here is a simple example which initializes a GlideRecord to the Incident Table, adds a query and then executes the query. The record set is then iterated over using `for_each_record`. The body of the `for_each_record` tag becomes the template which is used for each iteration.

The `for_each_record` tag also sets several variables to show the original table queried (`jvar_list_file`), the query used (`jvar_list_query`), the current record index (`jvar_active_row`) and the total number of records (`jvar_list_rows`). Unlike simply using a GlideRecord in JavaScript or using a while, `for_each_record` will apply the `canRead` ACL to each row. The total skipped will be stored as well (`jvar_skipped_rows`).

```
<?xml version="1.0" encoding="utf-8" ?>

<g:evaluate>
    var incidentRec = new GlideRecord('incident');
    incidentRec.addQuery("active","true");
    incidentRec.query();
</g:evaluate>

<ul>
    <g:for_each_record file="${incidentRec}" max="10">
        <li>${incidentRec.number} is ${jvar_active_row} of ${jvar_list_rows} on table ${jvar_list_file} with filter ${jvar_list_query}</li>
    </g:for_each_record>
</ul>

Skipped ${jvar_skipped_rows} Rows
```

## Output

- INC0000001 is 1 of 33 on table incident with filter active=true
- INC0000002 is 2 of 33 on table incident with filter active=true
- INC0000003 is 3 of 33 on table incident with filter active=true
- INC0000005 is 4 of 33 on table incident with filter active=true
- INC0000007 is 5 of 33 on table incident with filter active=true
- INC0000014 is 6 of 33 on table incident with filter active=true
- INC0000015 is 7 of 33 on table incident with filter active=true
- INC0000016 is 8 of 33 on table incident with filter active=true
- INC0000017 is 9 of 33 on table incident with filter active=true
- INC0000018 is 10 of 33 on table incident with filter active=true

Skipped 0 Rows

## Importing Variables

It is possible to easily parse a URL style parameter directly into Jelly variables using the `g:import_vars` tag. It takes one attribute `vars` which should be a string with key value pairs separated by `&` and `=`. If using a literal string `&` needs to be replaced with  `${AMP}`.

This example imports variables from a string then outputs them. Remember that the variables will need to be prefixed with `jvar_`.

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
    <g:import_vars vars="jvar_test=Hello World${AMP}jvar_name=Jelly Programmer" />
    ${jvar_test}
    ${jvar_name}
</j:jelly>
```

### Output

```
Hello World Jelly Programmer
```

## Getting Preferences

`g:preference` will get a User Preference. It takes three attributes, `name` which is the name of the preference to get, `var` is the variable to set the result to, `default` is the value to use if no explicitly set preference is found.

```
<?xml version="1.0" encoding="utf-8" ?>

${jvar_homepage}
```

### Output

```
My Default Homepage is: 4da11b86c611229701f1d31a48c9245d
```

## Setting a Variable in the JavaScript Context

When executing a Jelly page there is a context for each phase that exists with all the Jelly variables. Those variables exist in Jelly or are set using `j:set` or other native methods. There is also a JavaScript context. This is the context used when using `g:evaluate`. Using `g:setjs` you can set a variable using Jelly that exists in that JavaScript context. This makes it easy to use JEXL Expressions to set variables.

`g:setjs` requires a `var` attribute like `set` and either `value` to set a specific value or `expression` to use a JEXL expression.

This example illustrates the difference between `set` and `setjs`. You can see the variable set using `setjs` is available within the `evaluate` block.

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
    <j:set var="jvar_random" value="${Math.random()}" />
    Output: ${jvar_random}
    <g:setjs var="random" expression="Math.random()" />
    Output: ${random}

    <g:evaluate>
```

```
gs.log(jvar_random)
gs.log(random)
</g:evaluate>
</j:jelly>
```

## Log Output

```
Warning    org.mozilla.javascript.EcmaError: "jvar_random" is not defined.
Caused by error in <refname> at line 2
  1:
  ==> 2: gs.log(jvar_random)
  3: gs.log(random)
  4:

Information   0.8260521038111401
```

## Set A Value Conditionally

The Glide Tag Library has a shorthand for conditionally setting a value where only a True/False condition is used called `g:set_if`. The tag requires four parameters, `var` which is the var to set the result to, `test` which is the condition to check and `true` and `false` attributes whose values will be assigned to the variable depending on the test.

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
  <j:set var="jvar_isHelloWorld" value="true" />
  <g:set_if var="jvar_message" test="${jvar_isHelloWorld=='true'}" true="Hello World." false="Not Hello World."/>
  ${jvar_message}
</j:jelly>
```

## Output

```
Hello World.
```

## Creating Lists (Arrays)

To create a List from text based on a delimiter the `g:tokenize` tag can be used. It takes in two attributes, `var` which is the variable to store the List in and `delim` which is the delimiter to be used.

Special characters such as newline, tab and return can be used as well.

Here is an example which creates a List from the body of the tag using a newline as the delimiter.

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
  <g:tokenize delim="\n" var="jvar_test">
    this
    is
    a
    test
    of
    tokenize
  </g:tokenize>

  ${jvar_test.size()}
  ${jvar_test}

</j:jelly>
```

## Output

```
7 [this, is, a, test, of, tokenize, ]
```

The last line before the `</g:tokenize>` is treated as a valid input and so the last value in the list is `blank`.

## Preventing Escaping

`g:no_escape` will prevent text from being escaped. The body of the tag will be output without escaped text. This tag will also be affected by the system property `glide.ui.escape_text`.

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
    ${LT}
    <br/>
    <g:no_escape>
        ${LT}
    </g:no_escape>
</j:jelly>
```

## Output

```
&lt;
<
```

# Using Functions and Macros

ServiceNow has added some very useful ways to re-use code throughout the system. Keeping in mind that Jelly is typically used in a context that has access to a file system but in ServiceNow that's not possible. All the files are records in the database. So these functions have the ability to use those records in different ways, making re-use possible.

This section covers those tags which allow for that code re-use in a few different ways. As functions, including files or as templates.

## Calling Functions

ServiceNow has three methods to call "functions". These can be references to Templates on the file system or UI Macros. They may actually return or set a value and they may also directly output as well as wrapping other output.

The tags are `g:inline`, `g:insert` and `g:call`. The differences between the three are how they are called and how they interact with scopes of variables.

`insert` runs in a new context and inserts the result inline. According to the documentation you will not be able to read or edit variables in your outer scope (the Jelly Page) from within the function, however, I have found this to not be true.

`inline` is the equivalent of taking the contents of a template and placing it on the parent page.

`call` is the most like an actual function, taking parameters and returning values.

## Inserting Functions

`insert` is intended for those functions which are not meant to take in parameters or return values. Essentially static pieces of the page. It takes the name of the function or template as `template`.

The documentation on the Wiki incorrectly states that the function within the template will not have access to the outer scopes variables. In the following example we can see that the functionality is not as documented.

This example uses a UI Macro and UI Page and shows a function using `insert` that accesses the outer variables scope and sets it.

### UI Page

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
    <j:set var="jvar_message" value="Hello World" />
    Message Outside Before: ${jvar_message}<br />
    <g:insert template="my_function.xml" /><br />
    Message Outside After: ${jvar_message}<br />
</j:jelly>
```

### UI Macro - my\_function

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
    Message Inside: ${jvar_message}
    <j:set var="jvar_message" value="Changed!" />
</j:jelly>
```

## Output

```
Message Outside Before: Hello World
Message Inside: Hello World
Message Outside After: Changed!
```

While `insert` does seem to function like `inline` it actually is different. It is its own function, and it does in fact create a separate context, but the context seems to reference the outer context. If using `insert` I would try not to change or set variables from the outer scope within functions.

## Inlining Functions

`g:inline` is similar to `g:insert` but instead of executing a function separately and inserting the result, the function (or template) is included in the Jelly file. This means the function should be expected to be able to read and write all variables.

### UI Macro - my\_function

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
    <j:set var="jvar_message" value="Hello World" />
    Message Outside Before: ${jvar_message}<br />
    <g:inline template="my_function.xml" /><br />
    Message Outside After: ${jvar_message}<br />
</j:jelly>
```

### UI Page

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
    Message Inside: ${jvar_message}
    <j:set var="jvar_message" value="Changed!" />
</j:jelly>
```

## Output

```
Message Outside Before: Hello World
Message Inside: Hello World
Message Outside After: Changed!
```

You can see the output is exactly the same as the `insert` example.

## Calling Functions

`g:call` will make the equivalent of a function call to a UI Macro or Template. The scope of the function will be separate from the parent scope that calls it. Only variables passed as attributes on the tag will be available to the function.

`g:call` expects at least a `function` attribute which is the name of the Macro or Template with `.xml` appended.

### Simple Function Call

```
<?xml version="1.0" encoding="utf-8" ?>
```

```
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
<g:call function="glide_user.xml" />
</j:jelly>
```

This example will function exactly like using inline. Let's try using call for the same example as we did before with `insert`.

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
<j:set var="jvar_message" value="Hello World" />
Message Outside Before: ${jvar_message}<br />
<g:call function="my_function.xml" /><br />
Message Outside After: ${jvar_message}<br />
</j:jelly>
```

## Output

```
Message Outside Before: Hello World
Message Inside:
Message Outside After: Hello World
```

This output is different because when within the function there truly is a different context. In order for the function "my\_function" to have access to `jvar_message` it must be passed to it.

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
<j:set var="jvar_message" value="Hello World" />
Message Outside Before: ${jvar_message}<br />
<g:call function="my_function.xml" message="${jvar_message}" /><br />
Message Outside After: ${jvar_message}<br />
</j:jelly>
```

Notice in this example the `jvar_` is omitted from the name of the parameter, Jelly will automatically add this, so it is not necessary to specify it in the name of the parameter. The value is also not a literal value but a JEXL expression which references the same variable in the current context. This will ensure that the same value will be passed to the function.

## Output

```
Message Outside Before: Hello World
Message Inside: Hello World
Message Outside After: Hello World
```

`call` still does not function exactly like `insert` or `inline` because the change to `jvar_message` within the function does not overwrite the outer scopes message. This is intentional and necessary. What happens within the call can be trusted not to overwrite or clobber existing variables.

## Mandatory Fields and Default values

If we want to extend the concept of a function even further we can have required and default variables in our function parameters. This is done by adding the `g:function` tag to our function macro.

The `function` tag should be added to the top of your function macro right after the `jelly` tag.

Any attributes specified in this tag will be considered parameters to the function. They can have the value of `REQUIRED` making them required.

Here is the "my\_function" macro re-written to include parameters.

### UI Macro - my\_function

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
  <g:function message="REQUIRED" />
  Message Inside: ${jvar_message}
  <j:set var="jvar_message" value="Changed!" />
</j:jelly>
```

### UI Page

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
  <j:set var="jvar_message" value="Hello World" />
  Message Outside Before: ${jvar_message}<br />
  <g:call function="my_function.xml" /><br />
  Message Outside After: ${jvar_message}<br />
</j:jelly>
```

### Output

```
Message Outside Before: Hello World

Message Outside After: Hello World
```

The function call is missing here because the required parameter was not passed. In the error log is the exception thrown.

### Error Log

```
You must define an attribute called 'message' for this tag.: org.apache.commons.jelly.MissingAttributeException: null:@
```

The tag is expecting message as we defined it and it was not present. To fix this error simply add `message` to the `g:call` tag in the UI Page.

Now, let's add a default value for another field. The documentation on the Wiki states that attributes added to `function` can specify a default value. In practice this seems to not work, however to set a default you can still specify the default value before the `function` tag.

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
  <j:set var="jvar_first_name" value="default" />
  <g:function message="REQUIRED" />
  Message Inside: ${jvar_message}<br/>
  Name Inside: ${jvar_first_name}
</j:jelly>
```

`first_name` will be set initially to the default and if the same value is passed as a parameter it will be used instead.

## UI Macro - my\_function

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
  <g:function message="REQUIRED" name="John Smith" />
  Message Inside: ${jvar_message}<br/>
  Name Inside: ${jvar_name}
  <j:set var="jvar_message" value="Changed!" />
</j:jelly>
```

## UI Page

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
  <j:set var="jvar_message" value="Hello World" />
  <g:call function="my_function.xml" message="${jvar_message}" /><br />
</j:jelly>
```

## Default Values

The documentation on the ServiceNow wiki states that attributes in `function` may specify a default value. This actually seems to not work as intended. The intent is that any variable not passed in `call` can use a value specified in the `function` but this does not actually happen. As a work-around you can conditionally set the variable in the body of the `function` using `set_if`.

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
  <g:function message="REQUIRED" />
  <g:set_if var="jvar_test" test="${jvar_test==null}" true="Default Value" false="${jvar_test}" />
  ${jvar_test}
</j:jelly>
```

The `set_if` compares the target variable to `null`. If it is `null` the `true` condition is used otherwise just set it back to itself.

This may be intentional and the documentation is wrong or the documentation is wrong and it's a bug. This will be verified in newer versions.

## Returning Values

To further make Macros more "function like" they can return values instead of directly outputting. To do this you must first specify a `return` attribute on the `call` tag. This should contain the name of the variable to store the result in.

Second, in your function script you must specify a variable `jvar_answer`, most likely at the end of the script.

## UI Macro - my\_function

```
<?xml version="1.0" encoding="utf-8" ?>

var answer = jelly.jvar_message.toUpperCase();
```

## UI Page

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
  <j:set var="jvar_message" value="Hello World" />
  <g:call function="my_function.xml" message="${jvar_message}" return="jvar_new_message" />
```

```
New Message: ${jvar_new_message}
</j:jelly>
```

## Output

```
New Message: HELLO WORLD
```

In this example "my\_function" takes an input (`message`) and converts it to upper case then returns the result, which is output in the parent.

This is the equivalent in JavaScript as doing

```
var new_message = my_function(message);

function my_function(message){
    if(!message) throw "Missing Parameter"

    var new_message = message.toUpperCase();
    return new_message;
}
```

## Invoking Macros

Yet another way it seems to call macros is the `g:macro_invoke` function. Which takes at least `macro` as a parameter. Additional parameters may be passed as necessary by the macro. It is not clear at this time how this differs from the other methods of calling macros.

These are some documented usages from the system.

```
<g:macro_invoke macro="show_cascade_del_objs_table" obj_list="${jvar_delObjList}" />
```

```
<g:macro_invoke macro="index_creator_information" />
```

## Including UI Scripts

`g:include_script` allows you to easily include UI Scripts from the system. It takes a `src` attribute which is the name of the script include.

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
    <g:include_script src="WelcomeWidgetSelectControl.jsdbx"/>
</j:jelly>
```

## Requiring File System Scripts

`g:requires` is similar to `include_script` and creates a `script` tag to include a file specified in `name`.

```
<g:requires name="scripts/classes/TreeNode.js"/>
```

## Output

```
<script type="text/javascript" src="/scripts/classes/GroupTreeNode.jsx?v=12-12-2014_1622"></script>
```

## Including Forms (Deprecated)

`g:form` seems to include a template. It takes two attributes `template` and `mode`. By default the template is "form" which is dependent on may variables and inevitably causes an error.

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
    <g:form template="welcome_content.xml" />
</j:jelly>
```

## Rendering Components

`g:render_component` takes `componentName` as a parameter and renders it immediately. There are only 2 documented usages known however.

```
<g2:render_component componentName="com.glideapp.servicecatalog.VEditor"/>
<g2:render_component componentName="com.glideapp.questionset.DefaultQuestionEditor" />
```

Which are both used in UI Macros within the system. It is unknown if additional components exists that can be used this way.

## Including Client Scripts

To include Client Scripts you can use the `g:client_script` tag. There are a few configurations.

First sets a `type` attribute to "user". This will return the `g_user` object.

Second, is to use `g:client_script` to output Client Scripts for a particular form. This is done by specifying a `tableName` and either `type` or `file`. Form Client Scripts are dependent on having a reference set up and setting the reference to `ref`.

The following script outputs the Client Scripts for `g_user` as well as those for the Incident Form. Of course these will be dependent on the exact form fields existing on the page.

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
    <g:evaluate var="incident" copyToRhino="true" object="true">
        var gr = new GlideRecord('incident');
        gr.query();
        gr.next();
        gr;
    </g:evaluate>

    <j:set var="ref" value="incident" />

    <g:client_script type="user"/>
    <g:client_script tableName="${ref}" type="phase1"/>
    <g2:client_script tableName="${ref}" file="${[${ref}]}" />

</j:jelly>
```

# Miscellaneous Functions

This section covers other tags that do not fit into the previous categories.

## Doing Nothing

The Glide Tag Library comes equipped with its own "nop" function `g noop` which seems to just take in any attributes and a body, but performs no function. NOPs have plenty of use in the world of compiled code but here I can only see this being useful as a delay in a loop.

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
    <g noop noopAttribute="">
        Nothing going on here.
    </g noop>
</j:jelly>
```

## Getting A List of Filters For A Table

`g:get_filters` will get the list of filters for a table and set them to `jvar_filter_list` and the count of items to `jvar_filters_count`. It takes one attribute `tablename` which should be set to the table name.

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
    <g:get_filters tablename="incident" />

    There are ${jvar_filters_count} filters.
    <br/>

    <j:forEach items="${jvar_filter_list}" var="jvar_item">
        ${jvar_item.getDisplayTitle()} - ${jvar_item.getFilter()}<br/>
    </j:forEach>
</j:jelly>
```

## Output

```
There are 5 filters.
Active - active=true
Active - Unassigned - assigned_to=NULL^active=true
Assigned to me - active=true^assigned_to=javascript:gs.user_id()
Closed - active=false
My Open Incidents - caller_id=javascript:gs.getUserID()^active=true
```

## Getting a List of Decorations

`g:decorations` gets the list of UI decorations that are displayed in the top right of the ServiceNow Header. This will most likely just include the Home button.

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">

    <g:decorations var="jvar_decorations" />
    ${jvar_decorations}
</j:jelly>
```

## Output

```
[{title=Home, onmouseout=contextTimeout(event, 'homepages');, img=images/icons/home_16.gif, class=header, __tag_name=de
```



## Getting a List of Files

If you'd like to list the images directory or the uploads folder you can do so with the `file_browse` tag. The tag takes two attributes `directory` and `exts` which allow you to specify a directory and the file types you'd like to list in the directory.

To see images specify `directory="images"` or to see uploads specify `directory="images/UPLOADS"`. This example lists out files in the UPLOADS folder.

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
    <g:file_browse directory="images/UPLOADS" exts="jpeg,jpg,png,gif" />

    <j:set var="jvar_file_counter" value="0"/>
    <j:while test="${jvar_file_counter != jvar_available_files.size()}">
        <j:set var="jvar_img" value="${jvar_available_files.get(jvar_file_counter)}"/>
        <div>
            ${jvar_img}
        </div>
        <j:set var="jvar_file_counter" value="${jvar_file_counter + 1}"/>
    </j:while>
</j:jelly>
```

## Output

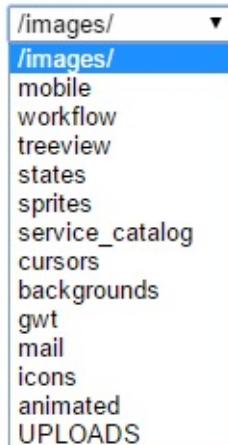
```
ui-bg_glass_95_fefiec_1x400.png
ui-icons_f29a00_256x240.png
ui-bg_flat_0_aaaaaa_40x100.png
ui-bg_glass_75_d0e5f5_1x400.png
...
```

You can't see or edit the contents of files on the system so images and uploads are the only things that are useful here.

To see the available Images directories you can use the `g:image_structure` tag.

```
<select>
    <g:image_structure />
</select>
```

## Output



## Getting Groupable Fields For a Table

`g:get_groupby_list` will generate a list of groupable fields for a table. It can take two parameters, `table` which is the table to generate fields for and `related` which can be set to "false" to hide related fields.

This example creates a choice list of groupable fields (including related) on the incident table.

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
    <g:evaluate var="incident" copyToRhino="true" object="true">
        sysparm_view="test";
        var gr = new GlideRecord('incident');
        gr.query();
        gr.next();
        gr;
    </g:evaluate>
    <g:get_groupby_list table="incident" />

    ${jvar_groupby_count} Total Options<br />

    <select>
        <j:forEach items="${jvar_groupby_list}" var="jvar_group_field">
            <option>${jvar_group_field}</option>
        </j:forEach>
    </select>
</j:jelly>
```

## Output

617 Total Options

Active : active
<b>Active : active</b>
Approval : approval
Assigned to : assigned_to
Assignment group : assignment_group
Business resolve time : business_stc
Caller : caller_id
Category : category
Caused by Change : caused_by
Change Request : rfc
Child Incidents : child_incidents
Close code : close_code
Closed by : closed_by
Company : company
Configuration item : cmdb_ci
Contact type : contact_type
Correlation ID : correlation_id
Correlation display : correlation_display
Created by : sys_created_by
Custom : u_custom
Delivery plan : delivery_plan

each value is actually "Display : field\_name". You could separate these further using a `g:evaluate` in the `forEach`.

## UI Tags

---

These tags that directly output some result. The result may be something as simple as an HTML comment or something as complex as the full Calendar functionality.

# HTML Elements

These are functions which output basic HTML tags only. In most cases you can simply output HTML right from Jelly but there are a couple special circumstances which require the HTML to be output through a function.

## Comments

This seems redundant since you are able to just use comments in your code, but actually Jelly will remove comments. This may be to save bytes that are transferred or to prevent accidentally returning comments that are intended to be private.

If you actually need to return a comment you can use the `g2:comment` tag. It is necessary to use `g2` since the comments are removed in each phase, meaning a comment output in Phase 1 (`g`) will be removed in Phase 2. Jelly does not distinguish between a comment typed by the user and one generated by a tag.

Beyond just documenting code comments are important in HTML because in some cases you may need to do conditional includes using comments for CSS. This is done using comments and is necessary to conditionally include Style Sheets for IE6-9.

Another important use for comments is with frameworks like KnockoutJS and AngularJS. You can embed functions in these comments instead of the HTML tag you are outputting. These are known as *Virtual Elements* because they perform some function but are not output to the user.

In this example the first comment is stripped out while the second comment makes it to the response.

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
  <g:comment>
    This comment will not be returned.
  </g:comment>
  <g2:comment>
    This will be returned to the user.
  </g2:comment>
</j:jelly>
```

## Output

```
<!-- This will be returned to the user. -->
```

## Setting The Doctype

There is a Glide Tag `g:doctype` which will return a `doctype` declaration to an output but it cannot be used in UI Pages or Macros because it will not be output as the first element of the page, resulting in it having no effect.

Trying to set the `doctype` in Phase 1 will result in an error **A DOCTYPE is not allowed in content..** So the `doctype` tag should only be used in Phase 2. `g2:doctype` takes a parameter `name` which is the doctype to be used. In almost every case this would just be "html" but can also use any full doctype, just enclose the string in single quotes since the doctype will include double quotes.

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
  <g2:doctype name="html" />
</j:jelly>
```

## Output

```
</div><!DOCTYPE html>...
```

### The `doctype` appears in the body, and so it does not affect the page.

There are many doctypes but the two that are most commonly used in Web Development are `html` and "none". Declaring no doctype will result in Internet Explorer using Quirks Mode, a special mode that has the functionality of IE6, which typically breaks many custom pages. This has a double negative effect of users trying to 'fix' the page by using Compatibility Mode. Once Compatibility is enabled on a page the doctype is effectively ignored. Even setting a `html` doctype will not affect the rendering of the page, it must be explicitly turned off by the user. Chrome and Firefox are not affected by this so if you develop exclusively in either of these browsers remember to test in IE.

To set a doctype for a custom page you must use the `sysparm_doctype=true` parameter in the URL or in ServiceNow versions that support UI14 you can set a system property `glide.uidoctype` to set this doctype at a system level.

# Form Elements

## Forms

While it is usually okay to create your own forms from scratch there is a `g:ui_form` function which will do that for you and has some additional benefits. `ui_form` takes `id`, `form_class` and `onsubmit` as parameters. The body of the tag will be wrapped with an HTML `form` tag and some additional code will be added to automatically set the `action` of the form. It also adds the `sysparm_ck` hidden input which may be necessary to prevent cross-site-request-forgery.

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">

    <g:ui_form id="my_form" form_class="form-horizontal" onsubmit="return someFunction('ok');">
        <g:ui_date name="birthday" id="birthday" value="2014-07-11" onchange="datePicked(this.value);"/>
    </g:ui_form>

</j:jelly>
```

## Headers

Pass some text to `g:ui_header` to create a form header.

```
<table>
    <tr>
        <td>
            <g:ui_header>My Form</g:ui_header>
        </td>
    </tr>
</table>
```

## Output

 My Form

## Form Fields

To output any field on a particular record you can use the `g:element` tag. It will automatically display the field using the correct UI element, for instance an `input` for a String field or dropdown for a Choice field.

`g:element` expects a "reference" to already be created. It requires an attribute `ref` which should contain the reference and the field to output as a string (ex. `incident.short_description`).

In this example a reference to Incident is created in an `g:evaluate` tag and set to the variable `incident`. `g:element` will expect `incident` to exist and be a GlideRecord object.

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">

    <g:evaluate var="incident" copyToRhino="true" object="true">
        var gr = new GlideRecord('incident');
        gr.query();
        gr.next();
        gr;
    </g:evaluate>
```

```
<g:element ref="incident.short_description" />
<g:element ref="incident.priority" />

</j:jelly>
```

## Output

Short description:  
Can't get to network file shares

Priority: 1

Notice that `ref` is just a String. `g:element` evaluates the reference in the Rhino Context. This is why it's necessary to set up the reference as in the `g:evaluate` block.

## Labels

`g:form_label` creates a label element which is basically the same thing as just doing a `label` directly except it will set the text direction based on the system internationalization settings, either left-to-right (default) or right-to-left.

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
    <g:form_label for="my_field"> Field:</g:form_label><br/>
    <label for="my_field">Field</label>
</j:jelly>
```

## HTML Output

```
<label for="my_field" dir="ltr">Field:</label>
<br>
<label for="my_field">Field</label>
```

## Currency Fields

Outputting currencies can be tricky, values will automatically convert to a users currency and you can use `getDisplayValue` to get the value in the currency format but if you need to do calculations and output the value it is necessary to do the formatting yourself. To make this easier there is the `g:currency_format` tag.

It takes the value as a number and will automatically convert the output to the correct format.

In this example the User Country has been set to the UK so the output will be in British Pounds. The tag will not convert the currency and assumes the value is already in the target currency.

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
    <g:currency_format value="100.99" />
</j:jelly>
```

## Output

£100,99

## Text Input

Standard single-line text fields can be output using `ui_input_field`. It takes `label`, `name` and `value` as parameters. It can also take JavaScript for events `onkeyup` and `onchange`. `readonly` can be set to "false" to disable the field and `style` can be used to pass style rules to the element.

```
<g:ui_input_field label="My Input Field" name="my_field" value="Hello World"/>
```

## Output

**My Input Field** Hello World

## Reference Fields

Reference fields are complex components which have many hidden parts, luckily there is a Macro `ui_reference` to create these for us.

It takes many parameters, `name` which is the name of the field, `table` the underlying table, `value` an existing value for the field (`sys_id`), `displayvalue` the Display value of the field, `show_popup` and `show_lookup` are booleans which control the additional search and find options of the Reference field. `completer` is the type of Autocompleter to use, `columns` are the selected display columns for the AC. `order_by` and `query` are both used to control the AC search.

Examples:

```
<g:ui_reference class="text"
    name="newAgendaTaskUser"
    id="newAgendaTaskUser"
    table="sys_user"
    query="active=true"
    size="50"/>
```

```
<g:ui_reference name="renewal_contact"
    table="sys_user"
    value="${renUser.sys_id}"
    displayValue="${renUser.name}" />
```

```
<g:ui_reference name="renewal_contact" table="sys_user" />
```

```
<g:ui_reference name="item_ref_field"
    query="sys_class_nameNOT IN${gs.getProperty('glide.sc.item.cannot_add_to_request')}"
    table="sc_cat_item" />
```

## Textareas

While it is possible to use the standard `textarea` tag it sometimes causes issues when the code is displayed in a textarea because the browser will end the outer textarea when the inner textarea tag is closed. To fix this there is a `g:textarea` tag. Any attributes passed to the tag will be output as attributes on the generated textarea and the body of the tag will become the value of the textarea. Be careful to use whitespace correctly when using multiple lines, the text area will not trim whitespace at the beginning of the new line so any tabs or spaces will appear in the textarea.

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">

    <g:textarea id="my_textarea" class="form-control">
Heres some text
Heres another line
    </g:textarea>
</j:jelly>
```

Heres some text  
Heres another line

## Multi-line Input Field

Close to a `textarea` is the `ui_multiline_input_field`. It creates a textarea input with a label and additional functionality. It takes in `name`, `rows` which is the number of rows to display, `size`, `label`, `value` if there is one, `mandatory` which is a boolean. It can also take JavaScript functions for `onchange`, `onkeyup` and `oncontextmenu`. `readonly` can be set to true to disable the input and `style` can be passed to control the style of the textarea.

## Buttons

To output a form button use `g:ui_button` which takes a parameter `action` which actually becomes the `id` of the final element and also matches to the *Action* value of a UI Action. The body of the tag becomes the body of the `button` tag.

```
<g:ui_button action="save">${gs.getMessage('Save')}</g:ui_button>
```

Save

## Checkboxes

`ui_checkbox` takes `id`, `name` and `value` as parameters and outputs a standard HTML checkbox input.

```
<g:ui_checkbox id="my_checkbox" name="my_checkbox" value="false" />
```

## Choice Lists

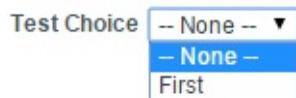
Choice Lists are output using `g:ui_choice_input_field`. The function takes `id`, `name`, `label`, `mandatory`, `for`, and `onchange` as parameters. The body of the tag should be `option` elements either entered directly or generated by another Jelly Function.

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">

    <g:ui_choice_input_field id="test_choice" name="test_choice" for="test_choice" label="Test Choice" mandatory="true">
        <option value="none">${gs.getMessage('-- None --')}</option>
        <option value="First">First</option>
    </g:ui_choice_input_field>

</j:jelly>
```

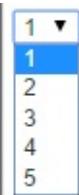
## Output



`ui_choicelist` is another way to output a Choice list for a table and field. It takes `class`, `name` and `table & field` which should point to a Choice List field on a table.

```
<g:ui_choicelist class="form-control" name="myquantity" table="sc_cart_item" field="quantity" />
```

## Output



To generate options for the choice list you can use `ui_select_option`. It takes `text`, `value` and `selected` a boolean to control whether it is selected.

```
<g:ui_select_option text="My Option" value="my_option" selected="true" />
```

## Date and Time Pickers

`ui_date` and `ui_date_time` will each output the standard ServiceNow date and time pickers. They take `name`, `id`, `value` and `onchange` as - mostly optional - parameters.

```
<g:ui_date name="birthday" id="birthday" value="2014-07-11" onchange="datePicked(this.value);"/>
```

```
<g:ui_date_time name="start_time" value="${start_time}"/>
```

## Output



## Glide Lists

`lightweight_glide_list` is an easy to use function that generates a Glide List. It depends on several variables. `jvar_list_data` is the current, comma-separated list of values, `jvar_reference` is the table to reference, `jvar_can_write` enables the list for editing, `jvar_control_name` is the name of the control and `jvar_not_focused` controls whether the element should have focus when the page loads.

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">

<j:set var="jvar_list_data" value="" />
<j:set var="jvar_reference" value="sys_user" />
<j:set var="jvar_can_write" value="true" />
<j:set var="jvar_control_name" value="Favorite Users" />
<j:set var="jvar_not_focused" value="true" />
<g:lightweight_glide_list />

</j:jelly>
```

### Output



`lightweight_glide_list` may also use `jvar_dependent` to set a dependent field for the reference and `jvar_ref_qual_elements` to set a reference qualifier. Although no examples could be referenced.

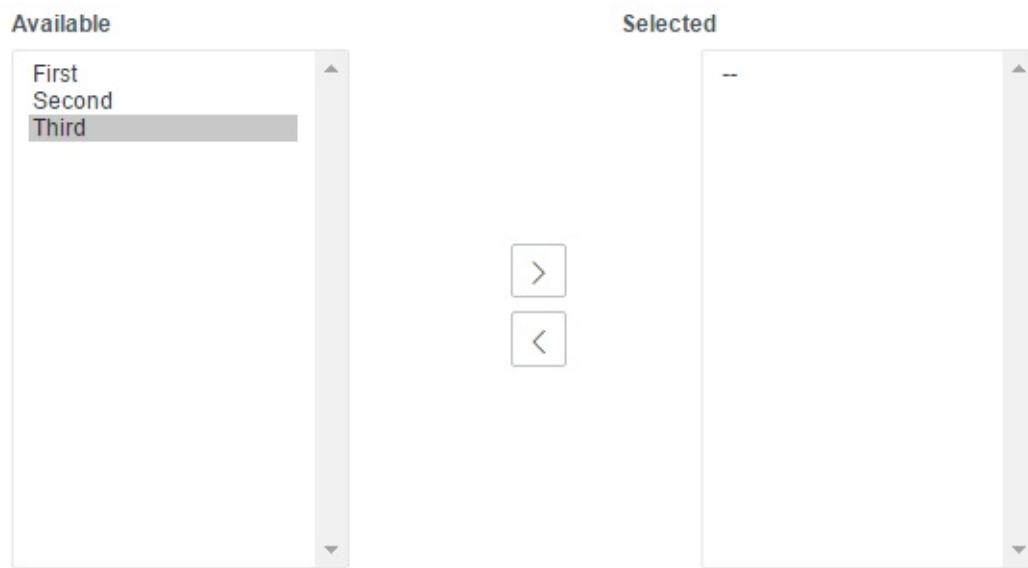
## Slush Buckets

Generating a Slush Bucket can be easily done using the `g:ui_slushbucket` function. It takes a parameters `up_down`, a boolean, which will control showing the scroll up and down buttons.

The body of the tag should be a set of HTML `option` tags.

```
<g:ui_slushbucket up_down="false">
<option>First</option>
<option>Second</option>
<option selected="true">Third</option>
</g:ui_slushbucket>
```

## Output



## Sections

To create a ServiceNow Style section (like the ones in Forms) you can use the `g:section` tag. The section title can be set by using a variable `jvar_section_caption` to the name of the section.

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
    <j:set var="jvar_section_caption" value="My Section" />
    <g:section>
        <h3>This is My Section</h3>
    </g:section>

    <j:set var="jvar_section_caption" value="My Other Section" />
    <g:section>
        <h3>This is My Other Section</h3>
    </g:section>
</j:jelly>
```



**This is My Section**



**This is My Other Section**



## Tables

`g:ui_table` calls a macro that wraps the body of table in the standard ServiceNow table. It takes no parameters.

```
<g:ui_table>
    <tr>
        <td>
            Here is my generated table.
```

```
</td>
</tr>
</g:ui_table>
```

**Output**

```
<table cell_padding="0" class="wide" cell_spacing="0">
  <tbody>
    <tr>
      <td>Here is my generated table.</td>
    </tr>
  </tbody>
</table>
```

**Form Splits**

Usually in ServiceNow forms you will have a two column layout. This same layout can be achieved in Jelly using the `g:vsplitter` and `g:vsplit` tags. Each tag expects a body that will be converted to a tabular layout.

You can even nest the splits to create 4 and 8 column layouts, but each split should only have a maximum of two columns, any more and the layout will start to wrap.

Here is an example which demonstrates creating 4 columns using two nested splitters.

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
  <g:vsplitter>
    <g:vsplit>
      <g:vsplitter>
        <g:vsplit>
          <h4>Column 1,1</h4>
          <p>Some content...</p>
        </g:vsplit>
        <g:vsplit>
          <h4>Column 1,2</h4>
          <p>Some content...</p>
        </g:vsplit>
      </g:vsplitter>
    </g:vsplit>
    <g:vsplit>
      <g:vsplitter>
        <g:vsplit>
          <h4>Column 2,1</h4>
          <p>Some content...</p>
        </g:vsplit>
        <g:vsplit>
          <h4>Column 1,2</h4>
          <p>Some content...</p>
        </g:vsplit>
      </g:vsplitter>
    </g:vsplit>
  </g:vsplitter>
</j:jelly>
```

**Output**

Column 1,1	Column 1,2	Column 2,1	Column 1,2
Some content...	Some content...	Some content...	Some content...

## Field Debugging Message

To output the Field Specific debugging messages you can use the `g:reportmessage` tag. It depends on a reference and an attribute `field` which specifies a field on the reference (example - `incident.short_description`).

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">

    <g:evaluate var="incident" copyToRhino="true" object="true">
        var gr = new GlideRecord('incident');
        gr.query();
        gr.next();
        gr.short_description.setError('Short Description should not contain Jelly.');
        gr;
    </g:evaluate>

    Message:
    <g:reportmessage field="incident.short_description" />
    <br/>
    Element:
    <g:element ref="incident.short_description" />

</j:jelly>
```

### Output

The screenshot shows a ServiceNow interface with a message and an element. The message is "Message: Short Description should not contain Jelly." with an error icon. The element is "Element: Short description:" followed by a code editor containing "<j2:set var='jvar\_insecure' value='\${'Uh oh!'}' />" and an error icon. Below the message and element are two small circular icons.

As you can see in the example, `g:element` already includes the debugging output so if you use that it's not necessary to use `reportmessage` again.

## Spacers

Might be necessary - although it is highly discouraged - to use a spacer. `g:ui_spacer` will insert a small wedge using `margin` on a `span`. It takes `pixels` as a parameter which will default to "5".

```
<g:ui_spacer pixels="20" />
```

## Outputting Page Parameters

The `g:emitParms` tag will output all page parameters as hidden inputs on the page. This tag takes no parameters.

The `emitParms` tag will only output parameters starting with:

- `sysparm`
- `sysparm_collection`
- `sysparm_userpref`

The `emitParms` tag will not output the following parameters:

- `sysparm_this_url`
- `sysparm_referring_url`
- `sysparm_field`
- `sysparm_field_list`
- `sysparm_collection`
- `sysparm_userpref`

This example shows emitting the parameters on a custom page passing in `sysparm_foo=bar`.

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
    <g2:emitParms />
</j:jelly>
```

## Output HTML

```
<input type="HIDDEN" name="sysparm_foo" id="sysparm_foo" value="bar"></input>
<input type="HIDDEN" name="sysparm_base_form" id="sysparm_base_form" value="ui_page_render"></input>
<input type="HIDDEN" name="sysparm_view" id="sysparm_view" value=""></input>
```

## Field Decorations

In an earlier section I talked about creating Decorations for fields `reference_decoration` and `reference_decoration_custom` are two built-in ways to make decorations for reference fields.

`reference_decoration` takes `onclick`, `id`, `field`, `image`, and `title` as parameters. `title` is optional but the rest are required.

`reference_decoration_custom` takes `onclick`, `id` and `field` but instead of `image` it uses the body of the tag as the actual decoration, so it can be something other than an image.

Here are two examples from the system. The first one outputs the Related Incidents decoration and the second outputs the Show Workflow decoration. Both are used from UI Macros that are themselves used as field decorations for references fields.

```
<g:reference_decoration_custom id="${jvar_n}" field="${ref}"
    onclick="showRelatedList('${ref}'); ">
    <span class="tab_square_button tab_header_button tab_ref_contribution icon-tree-right"
        aria-label="${HTML:gs.getMessage('Show related incidents')}">
    </span>
</g:reference_decoration_custom>
```

```
<g:reference_decoration id="show_workflow.${ref}" field="${ref}"
    onclick="showWorkflow('${ref}'); " title="${gs.getMessage('Show Workflow')}"
    image="images/workflow/wkfl_definition.gifx" />
```

# List Elements

## Generating Lists

There are two functions which make creating list views easy, `g:list_custom` and `g:list_default` both create the necessary references to tables that can be used to create list views.

Both accept `table` and `view` set to the value of the intended table to create a list for and its view respectively. `list_custom` also accepts an `elements` attribute which can be a comma separated list of field names.

After each function it is necessary to call the template `list2_default.xml` to actually output the list.

The template will also use `ListProperties`, `sysparm_query`, `sysparm_fixed_query`, `sysparm_view`, `sysparm_first_row`, `sysparm_rows_per_page`, and `sysparm_list_css` if they are set.

A complete list of `ListProperties` properties and methods is available in the JavaScript Objects Reference

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">

    <g:list_custom table="incident" view="" elements="number,short_description" />
    <g:inline template="list2_default.xml" />

</j:jelly>
```

## Output

<a href="#">INC0010802</a>	Test Short Description
<a href="#">INC0010801</a>	Test Short Description
<a href="#">INC0010800</a>	
<a href="#">INC0010599</a>	
<a href="#">INC0010598</a>	
<a href="#">INC0010597</a>	Test Short Description

## Generating Related Lists

Related Lists also follow a similar template for normal lists, `g:list_related` takes `parentTable` and `related` as parameters. `parentTable` is of course the parent table, `related` is the related table and the field to relate on.

`list_related` also depends on `ListProperties` and another function `list_record_default` which uses `properties` and `query` to set up the query for the list template. `properties` is a serialized list of `ListProperties` and `query` is an encoded query to relate the parent and related table.

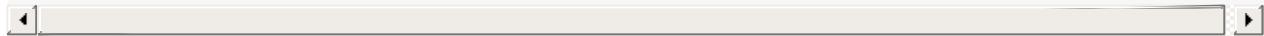
```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">

    <g:list_related parentTable="task" related="task_sla.task" />

    <g:evaluate>
        ListProperties.setHasHeader(true);
    </g:evaluate>
```

```
<g2:list_record_default properties="${ListProperties.serialize()}" query="task=3dd5fda90f2fb1005c95059ce1050e70">
    <g:inline template="list2.xml" />
</g2:list_record_default>

</j:jelly>
```



## Output

SLA	Stage	Start time	Planned end time	Actual elapsed time	Actual elapsed percentage	Actual time left	Business elapsed time	Business elapsed percentage
Priority 3 ● resolution (5 day)	Achieved	2008-07-23 12:25:19	(empty)	3 Minutes	0.06			

Another type of list is the **Relationship** list driven by a configured query that links two tables as opposed to a reference/sys\_id relationship as in standard Related Lists.

Like the other functions `list_relationship` follows the same template. It takes `parentTable` which is the parent table, `related` which is set to the sys\_id of the System Relationship, prefixed with `REL:`, `view` which can be blank or set to the view and finally `isEmbedded` which is not clear but may set some styling or List Properties to display as an embedded list.

Similar to `list_related` there is a function to help set up the query, `list_record_relationship` which takes `properties` (same as `list_record_default`), `parentID` which is the sys\_id of the parent record, and `query`, which may be left blank and probably should in this case.

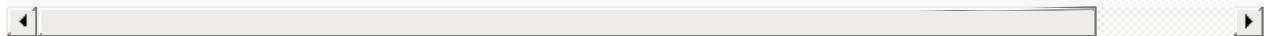
```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">

    <g:list_relationship parentTable="sys_db_object" related="REL:4344f6f5bf1320001875647fcf0739ad" view="" isEmbedded='

        <g:evaluate>
            ListProperties.setHasHeader(true);
        </g:evaluate>

        <g2:list_record_relationship properties="${ListProperties.serialize()}" parentID="845f7e720f3231005c95059ce1050e33'>
            <g:inline template="list2.xml" />
        </g2:list_record_relationship>

    </j:jelly>
```



## Output

Column label	Type	Reference	Max length	Default value	Display
● Active	True/False		40	true	false
● Activity due	Due Date		40		false
● Approval	String		40	not requested	false
● Approval history	Journal		4,000		false
● Approval set	Date/Time		40		false
● Assigned to	Reference	User	32		false
● Assignment group	Reference	Group	32		false

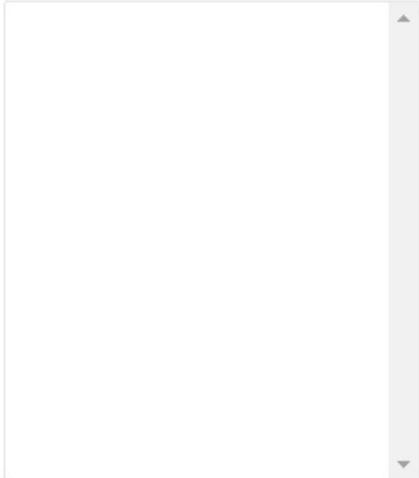
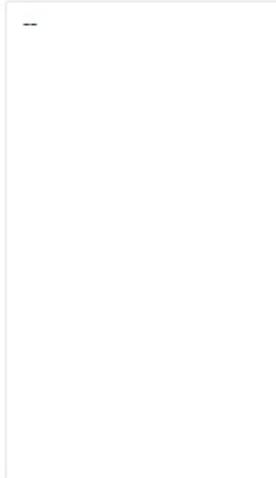
## Additional List Functions

`g:list_mechanic2` and `g:list_mechanic_inner` both seem to render part of the List Mechanic feature however simply placing them in breaks the page immediately.

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">

    <g:list_custom table="incident" view="" elements="number,short_description "/>
    <g:evaluate>
        ListProperties.setHasListMechanic(true);
    </g:evaluate>
    <g:list_mechanic2 />
    <g:list_mechanic_inner />
    <g:inline template="list2_default.xml" />
</j:jelly>
```

**Output**

**Available** **Selected**

Wrap column text  Compact rows  Active row highlighting  Modern cell coloring  
 Enable list edit  Double click to edit



The `list_mechanic2` function is dependent on `ListProperties.setHasListMechanic(true)`

# Choice Lists

## Generating Choice Lists Options

Choice lists values can be retrieved by using the `g:choice_list` tag. The tag takes `table` and `column` attributes to specify the source of the choices. There is an optional attribute `value` to specify the selected value.

The choice list tag returns an object in the variable `jvar_choices` that contains all the choices, their labels and values and whether or not it is the selected value.

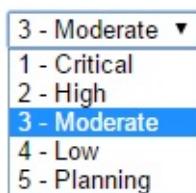
### Example Choice List Object

```
[
  {
    label : '1 - Critical',
    value : 1,
    selected : false
  },
  {
    label : '2 - High',
    value : 2,
    selected : true
  }
  ...
]
```

Here is an example that pulls the choices from the Priority field on Incident and generates a choice list with the selected value of 3. Then creates a dropdown with those choices, automatically selecting the correct item.

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
  <g:choice_list table="incident" column="priority" value="3" />
  <select>
    <j:forEach var="jvar_choice" items="${jvar_choices}">
      <j:if test="${jvar_choice.selected=='true'}">
        <option value="${jvar_choice.value}" selected="true">${jvar_choice.label}</option>
      </j:if>
      <j:if test="${jvar_choice.selected=='false'}">
        <option value="${jvar_choice.value}">${jvar_choice.label}</option>
      </j:if>
    </j:forEach>
  </select>
</j:jelly>
```

### Output



## Generating Options From a Reference

`g:options` can be used to generate choice lists from a reference

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
    <g:evaluate var="incident" copyToRhino="true" object="true">
        var gr = new GlideRecord('incident');
        gr.query();
        gr.next();
        gr;
    </g:evaluate>

    <select>
        <g:options ref="incident.priority" choiceValue="3" />
    </select>
</j:jelly>
```

## Generating Options With Scripts

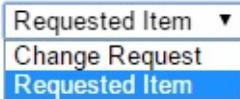
Another way to generate a dropdown list is using a Script Include. `g:scripted_options` will allow you to specify a Script Include and function to generate a list of options. This is especially useful if you want to re-use the same logic generating the list in other scripts or Jelly Pages.

The tag accepts a few options, `script` and `function` should be set to the Script Include and function within the Script Include to run respectively. `selected` should be the selected value.

This example uses the OOB Script "WFStageSet" to generate a list of options and sets the selected value to "Requested Item".

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
    <select>
        <g:scripted_options selected="Requested Item" script="WFStageSet" function="getSetNames" />
    </select>
</j:jelly>
```

### Output



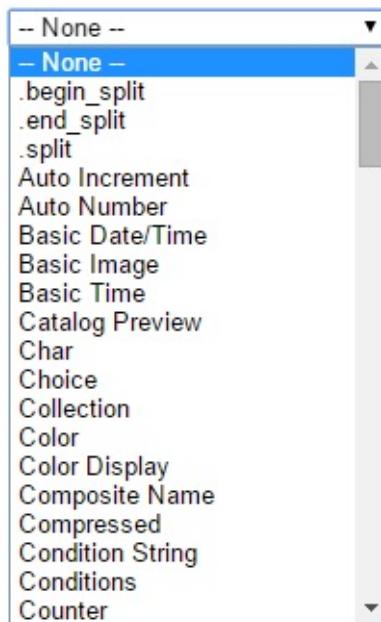
## Generating a List of Internal Types

`g:internal_type_options` will return a Choice List of Internal Types (**String, Integer, Reference, etc...**). It has one configurable attribute `includeContainerType` which will include the containers (`.begin_split`, `.end_split`, `.split`) that can be true or false.

This example shows the full list of Internal Types.

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
    <select>
        <g:internal_type_options includeContainerTypes="true" />
    </select>
</j:jelly>
```

## Output



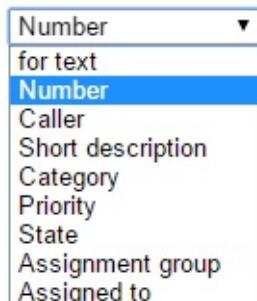
## Generate a List of Searchable Column Fields

To get a list similar to what is presented in the List View search you can use the `g:columnoptions` field. It will take a reference in `ref` and also has an attribute `fields` in which you can specify a comma separated list of the fields to display.

This sample will generate the list of searchable columns for the Incident table.

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
    <g:evaluate var="incident" copyToRhino="true" object="true">
        var gr = new GlideRecord('incident');
        gr.query();
        gr.next();
        gr;
    </g:evaluate>
    <select>
        <g:columnoptions ref="incident" />
    </select>
</j:jelly>
```

## Output

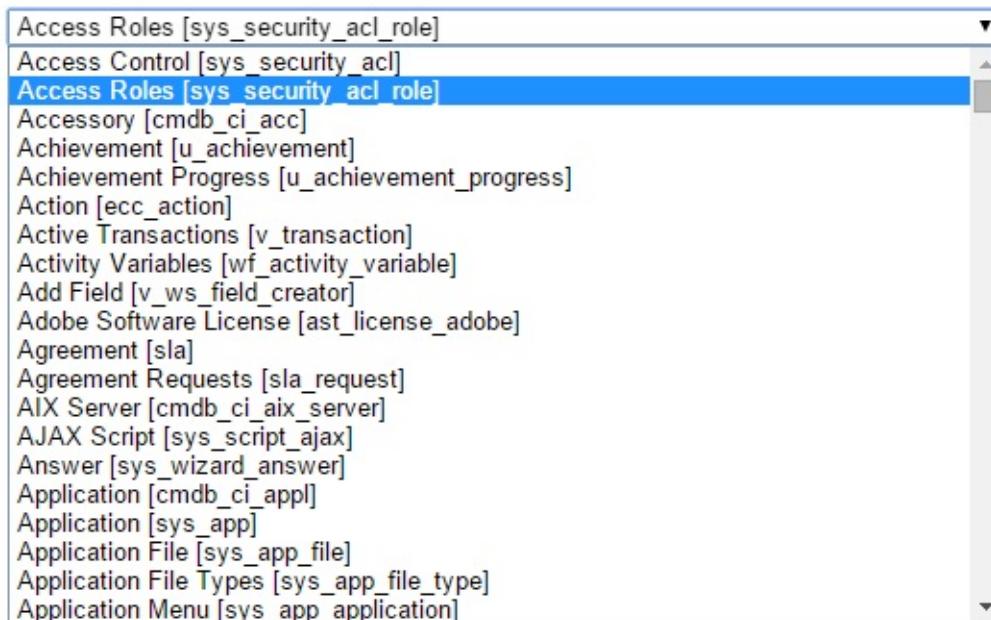


## Generating a List of Tables

To create a choice list of all tables (with friendly names) you can use the `g:table_options` tag.

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
  <select>
    <g:table_options />
  </select>
</j:jelly>
```

### Output



Another similar tag `g:update_table_options` will generate a list of tables that are capturable in Update Sets.

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
  <select>
    <g:update_table_options />
  </select>
</j:jelly>
```

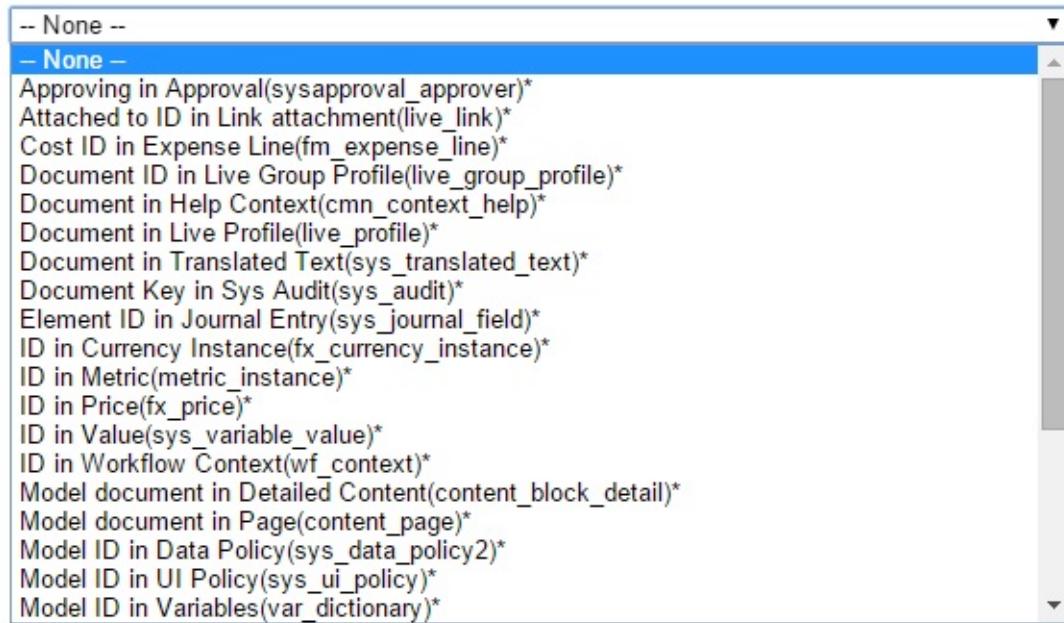
Lastly `g:search_table_options` will output a list of tables that can be searched through.

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
  <select>
    <g:search_table_options />
  </select>
</j:jelly>
```

## Generating a List of Document ID Fields

`g:reference_options` lists all fields of `document_id` types in all tables. It takes an attribute `table` which seems to be ignored.

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
<select>
    <g:reference_options />
</select>
</j:jelly>
```

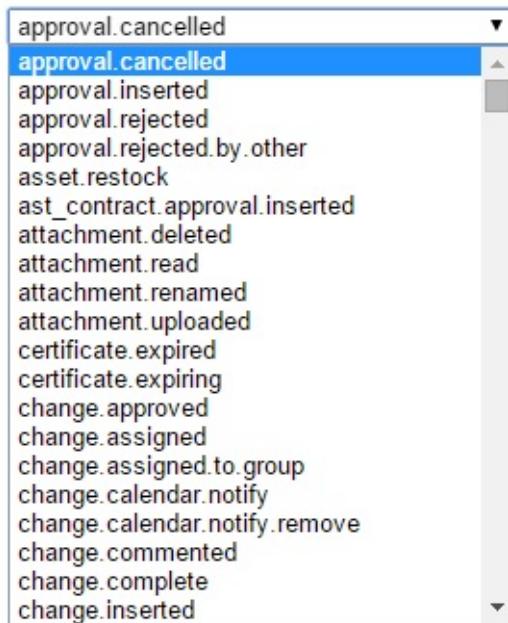


## Generating a List of System Events

To generate a choice list of System Events, use `g:sysevent_name_options`. This tag takes no attributes.

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
<select>
    <g:sysevent_name_options />
</select>
</j:jelly>
```

### Output

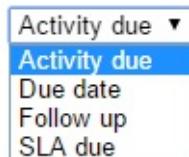


## Generating a List of Reminder Fields

In ServiceNow **due\_date**, **follow\_up**, **sla\_due**, **activity\_due** are referred to as Reminder Fields. These can be listed for a particular table by using the `g:reminder_field_name_options` tag. It takes `tableName` as an attribute that should be the table name and returns a choice list of the specific fields in this class.

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
  <select>
    <g:reminder_field_name_options tableName="incident" />
  </select>
</j:jelly>
```

### Output

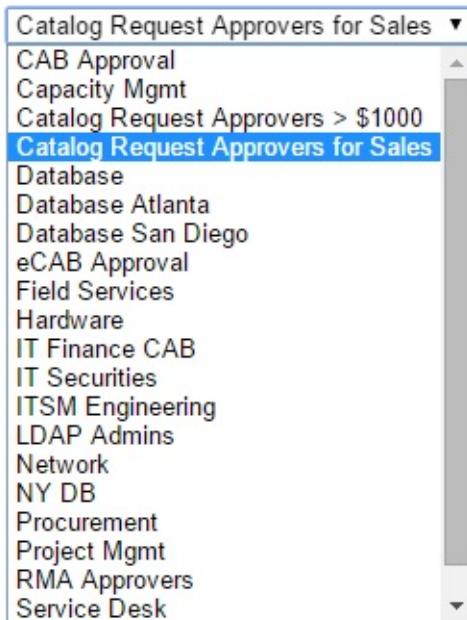


## Generating a List of User Groups

`g:my_groups` will output a choice list of the current users Groups. If the user is an Administrator all groups are output.

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
  <select>
    <g:my_groups />
  </select>
</j:jelly>
```

## Output



`g:filter_groups` is very similar but only lists groups but for those users who can create Global Filters.

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
    <g:filter_groups />
</j:jelly>
```

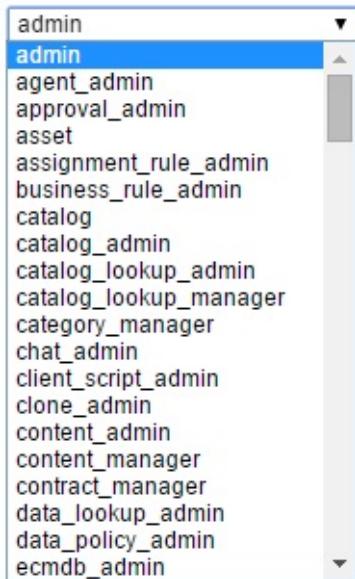
## Generating a List of Roles

`g:get_roles` can be used to get a dropdown of available roles in the system. It takes at least a `type` and `record` attribute. `type` can be "available" or "selected", `record` should be a list of roles that are omitted from the list. There is also a `readonly` attribute which will output the values in `record` (not particularly useful).

This example shows creating a dropdown with the available roles.

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
    <select>
        <g:get_roles type="available" record="" />
    </select>
</j:jelly>
```

## Output



## Generating A List of Child Tables

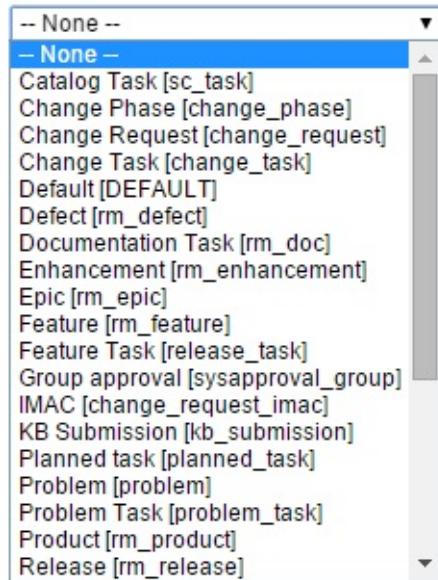
Outputting the child tables of a specific table is done by using the `g:child_tables` tag. The tag will output a choice list of tables that inherit from a root table. You can specify the root table with the `root` attribute. `skip_root` will prevent outputting the root table to the choice list.

There are a few other attributes: `includeDefault` will add a "Default" value to the list. `skipTables` will accept a list of tables to prevent from being output in the list. Finally, `choiceType` can be set to "1" which will add a "None" option to the list, useful if you are adding the choice list to a form.

Here is an example which outputs tables inherited from task, skipping "incident", including a Default Value and adding a None Value.

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
<select>
    <g:child_tables root="task" skipRoot="true" includeDefault="true" skipTables="incident" choiceType="1" />
</select>
</j:jelly>
```

### Output



## Generating a List of Fields

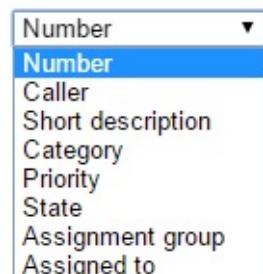
To output a choice list of fields associated with a particular List view of a table you can use the `g:get_list` tag. `get_list` depends on a reference being set to `ref`. You may also specify a particular type of list: `list`, `related_list`, `section`, `activity`, `sys_choice`, `variable_set`, `record_producer` in the `form` attribute to pull the fields from that specify type of list.

This example pulls the fields from the List view of Incident (what you would normally see on "incident\_list.do").

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
    <g:evaluate var="incident" copyToRhino="true" object="true">
        sysparm_view="test";
        var gr = new GlideRecord('incident');
        gr.query();
        gr.next();
        gr;
    </g:evaluate>

    <select>
        <g:get_list form="list" ref="incident" />
    </select>
</j:jelly>
```

### Output



## Generating a List of Field Values

`g:response_options` will generate a list of all values for a particular field in a table including translations. It requires two attributes `table` and `field` which should be set to the table and field to pull the values from.

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
    <select>
        <g:response_options table="incident" field="priority" />
    </select>
    <br/>
    <select>
        <g:response_options table="incident" field="short_description" />
    </select>
</j:jelly>
```

# Content Management Tags

## Generating Lists

Within the Content Management System, **List Definitions** may generate lists directly using Jelly within the List Definition record or they may use `g:content_summarizer` which uses a custom rendering definition from **Content Types**. This allows some lists to be styled for a particular CMS Site.

`content_summarizer` takes a single parameter `content` which should contain a reference to the record to be summarized.

Here is an example from `list_unordered_larger` which is a List Definition part of the core ServiceNow CMS.

```
<div style="font-size:larger;">
    <div class="content_list_title" style="width:98%;">${jvar_title}</div>
    <ul>
        <g:for_each_record file="${current}" max="${jvar_max_entries}">
            <li style="padding-top:4px;">
                <g:content_summarizer content="${current}" />
            </li>
        </g:for_each_record>
    </ul>
</div>
```

This function is dependent on the Content Management System and having a **Content Type** defined for the table you are generating a list for.

## Rendering Content Blocks

To render different types of Content Blocks in or out of a CMS Page you can use the `content_block` function. It takes two parameters `type` which should be a valid Content Block Type and `id` which is the `sys_id` of the Content Block record.

Valid types are each of the Content Management System *Block* and *Specialty Content* types.

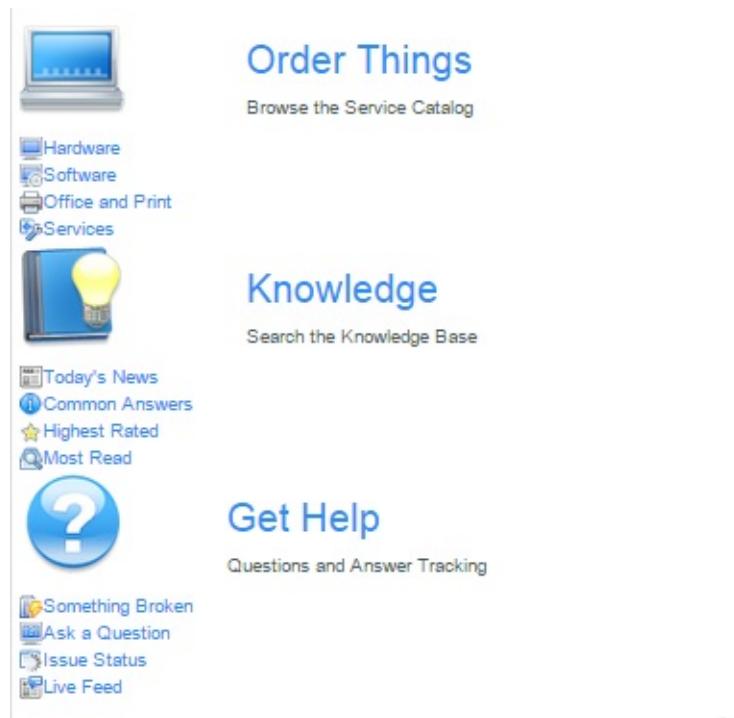
- `content_block_header`
- `content_block_menu`
- `content_block_programmatic`
- `content_block_lists`
- `content_block_static`
- `content_block_flash`
- `content_block_content_link`
- `content_block_iframe`

This example outputs a Content Management Menu.

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">

    <g:content_block type="content_block_menu" id="577043b4c0a8016b003cbacf363d38f3"/>

</j:jelly>
```



## Menus

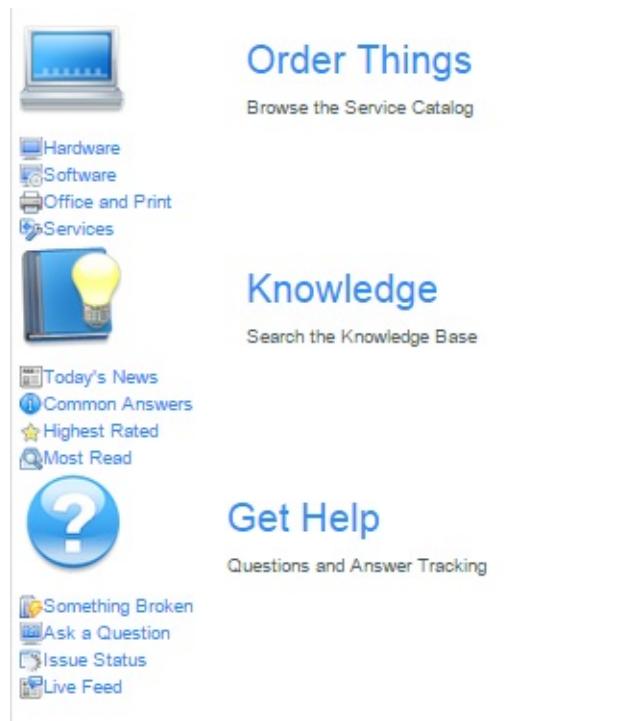
To output a menu in or out of a Content Management Page you can use the `g:cms_menu` function which takes a parameter `menu_sysid` which refers to a Content Management Navigation Menu record.

This example outputs one of the sample menus.

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">

<g:cms_menu menu_sysid='577043b4c0a8016b003cbacf363d38f3' />

</j:jelly>
```



`g:cms_menu_functions` is function used within a CMS Menu Macro which creates a Server Side function to check if a user can see the menu. This should always be included in any custom menus you create within the system.

`g:cms_menu_set_url_and_target` is another function used within CMS Menus and is used to generate the URL and target for a particular menu item. If a `link_sysid` parameter is passed to the function, it is treated as a Menu Section otherwise it is treated as a Menu Item. The result of the function is `jvar_link_url` and `jvar_link_target` will be set based on the section or item, to be used immediately after the function is called.

## Content Management Header Functions

These functions are really only useful as part of the rendering of Content Management Headers (UI Macro - `render_content_block_header`). Each outputs a specific part of the CMS Header and is dependent on being rendered in that context.

- `cms_header_chat`
- `cms_header_font_sizer`
- `cms_header_login`
- `cms_header_logo`
- `cms_header_search`
- `cms_header_text`

These are all accessible as UI Macros.

## Generating A Page Grid

Content Management Pages use layouts which have configurable "dropzones". Content Blocks are added to these drop zones and the configuration is saved in the CMS page. In order for these drop zones to work properly additional code is added around the sections forming a grid.

To generate a grid like the CMS you must first load a **Layout** then call `g:generate_grid` which takes two parameters `id` which is the Sys ID of the Content Page and `editable`, a boolean which controls whether the page is editable.

```
<?xml version="1.0" encoding="utf-8" ?>
```

```
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">  
    <g2:call function="layout_2_across.xml" name="test"/>  
    <g2:generate_grid id="0b7fda90ef202000914304167b22564c" editable="false"/>  
</j:jelly>
```

This example generates a page by first loading the Layout, then adding the grid and grid content.

# Knowledge Base Components

These functions are used in Knowledge Base pages. These functions often depend on some additional "setup" in their source pages, such as establishing a reference to an article or list.

## Knowledge Base Components

### Search Drop Down

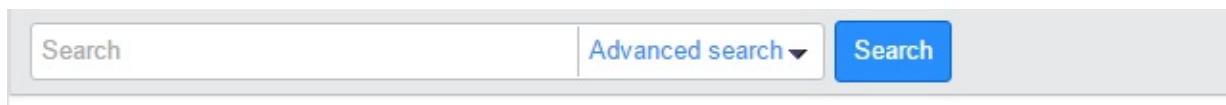
`g:kb_header_search` will produce the Knowledge Base Header Search form. The macro uses `g:kb_advanced_search`, which inserts the "Advanced Search" drop-down. Trying to use this on its own produces no output.

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">

<g:kb_header_search />

</j:jelly>
```

### Output



### Language Drop Down

Although it does not work on its own (it depends on set up within the KB Pages) `g:kb_languages` will produce the Knowledge Base Language Picker.

### Knowledge Attachment

If a Knowledge Article contains an attachment link `g:com.glideapp.knowledge_attachment` will find the attachment and store the ID in a given variable. `ref` stores the ID of the KB Article and `var` is the variable to store the result in.

```
<g:com.glideapp.knowledge_attachment var="jvar_attachid" ref="${kb.sys_id}" />
```

### Column Sets

`g:com.glideapp.knowledge_columns` seems to organize KB Articles for a given section into columns based on a known column count.

This is an example usage in `kb_home`

```
<g2:com.glideapp.knowledge_columns var="jvar_columnset" section="${jvar_view_topic}" columns="${jvar_kb_columns}" />
```

From its usage it produces a variable for each column with articles in each. Further down the page:

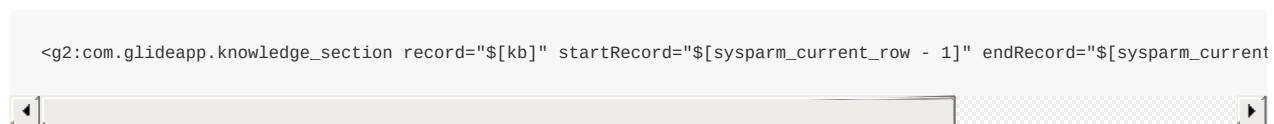
```
<j:while test="${jvar_now_column < jvar_kb_columns}">
  <td width="50%" id="dcolumn_${jvar_now_column}" dragging="true" valign="top">
    <j2:forEach var="jvar_kb_section" items="${jvar_columnset${jvar_now_column}}">
```

This will produce a loop for each column and the `forEach` will be looping over `jvar_columnset1`, `jvar_columnset2` etc...

## Sections

It is unclear what exactly `com.glideapp.knowledge_section` does. From its context it seems to create and/or execute a query on the Knowledge Base for the purposes of returning some metrics to calculate the Section or return records directly.

`record` actually seems to be a reference to a FilteredGlideRecord pointed at `kb_knowledge`, `startRecord` and `endRecord` are integers establishing the parameters of the result.



```
<g2:com.glideapp.knowledge_section record="${kb}" startRecord="${sysparm_current_row - 1}" endRecord="${sysparm_current
```

## Pagination Controls

To output the KB Pagination Controls (also called VCR functions here) you can use `kb_page_vcr` or `kb_v3_vcr`. Both depend on some set up and are only used within the Knowledge Base pages. `kb_page_vcr` produces the older style controls and `kb_v3_vcr` produces the new style (aligned right, modern look).

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">

  KB Page VCR<br/>
  <g2:kb_page_vcr />

  <hr/>

  KB V3 VCR<br/>
  <g2:kb_v3_vcr />

</j:jelly>
```

## Output

### KB Page VCR



◀◀ □ to NaN of org.mozilla.javascript.Undefined@177421a ▶▶

### KB V3 VCR



◀◀ ◀ □ to NaN of ▶ ▶▶

## Buttons

### Mark as Solution

To output the "Mark as Solution" button you can use `solution_button`.

```
<g:solution_button/>
```

## Attach

The Attach button can be output using `kb_view_attach_button` takes one parameter `kb_search_table` which appears to be the table searching from.

```
<g:kb_view_attach_button kb_search_table="incident" />
```

## Edit

If the current user has write permissions, the Edit button can be output with `kb_view_edit_button`.

```
<g:kb_view_edit_button />
```

# Chart Components

---

## Setting A Diagram Scale

`diagram_scale` is a Macro which outputs a dropdown selector for the scaling percentage of diagrams. It takes a parameter `diagram` which is the name of the diagram and `after`, an optional parameter for a function to execute after scaling.

## JRobin Graph Controls

`jrobin_graph_controls` seem to be used when rendering the Performance Gadget controls seen on the Admin dashboards.

# Extended UI Components

These tags are UI Elements that extend the core set of Glide and Jelly tags as UI Macros. They are usable within Jelly Pages and Macros as if they were Jelly tags.

## Annotations

`annotation` calls the `annotation` UI Macro and adds different types of annotations. Through testing this seems to not work entirely as expected. The output HTML is missing the text and this tag may need a specific context to output correctly

```
<g2:no_escape>
  <g2:annotation text="${jvar_sc_category_infomsg}" annotation_type="Section Details"/>
</g2:no_escape>
```

### Output

```
<div style="padding:2px 0px 0px 4px; ; " class="annotation-wrapper">
  <input style="visibility:hidden; width:0px;" id="make_spacing_ok">
</div>
```

Another type of annotation is the `yellow_annotation`. This macro produces a yellow "warning style" annotation on a form with some given text in the body.

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">

  <g:yellow_annotation>Here is my Yellow Annotation. This may be a warning.</g:yellow_annotation>

</j:jelly>
```

### Output

Here is my Yellow Annotation. This may be a warning.

## Generating A List of System Applications

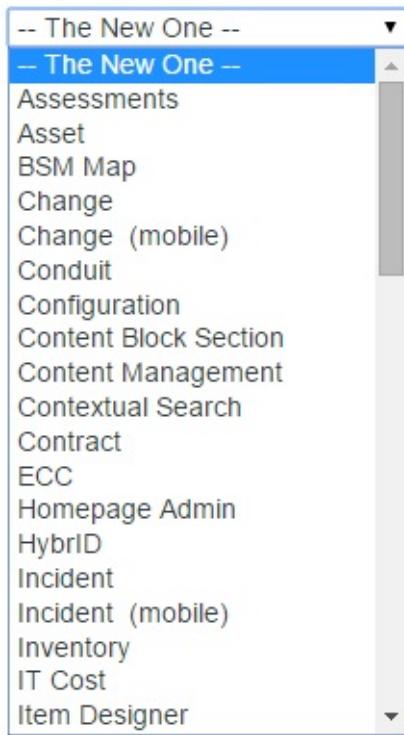
`g:application_selector` outputs a list of Applications in the system. This is the same list displayed on the "Create Table" form. It takes no additional parameters.

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">

  <g:application_selector />

</j:jelly>
```

### Output



## Generating Attachment Lists

To generate a list of attachments for a particular record there are two functions `g:attachment_list` and `g:attachment_entry` which will generate output similar to what is in the "Attach File" Dialog.

`attachment_list` takes two parameters `sys_id` and `table` which should be the `sys_id` and table of the target record.

`attachment_entry` is the function which outputs the filename as a link as well as the "Rename" and "View" actions.

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">

    <g2:attachment_list sys_id="62130bd50f10c6005c95059ce1050e9b" table="incident">
        <g2:for_each_record file_variable="sys_attachment" var="attachment">
            <g2:attachment_entry />
        </g2:for_each_record>
    </g2:attachment_list>

</j:jelly>
```

### Output

[my\\_attachment.txt](#) [rename] [view]

## Form Buttons

There are several UI extensions to easily generate form buttons.

`dialog_button`, `dialog_button_ok`, `dialog_buttons_ok_cancel` and `dialog_buttons_save_cancel_discard` will all output different

sets of buttons styled with the common ServiceNow styling.

The core macro is `dialog_button` which the other macros use to generate their buttons.

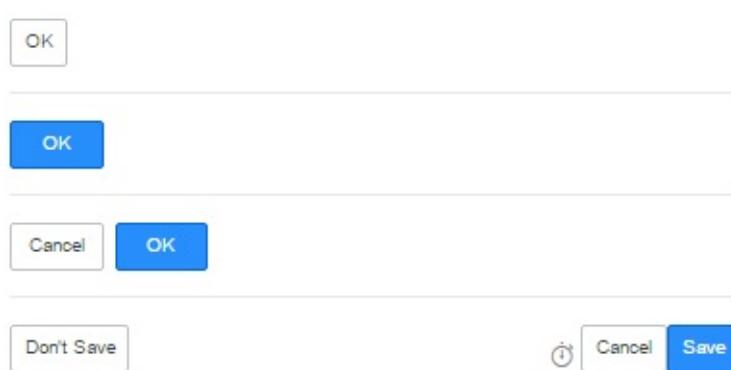
Here is an example which uses each function and also shows how to override the default values of the macros.

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">

    <g:dialog_button onclick="console.log('Dialog Button')" name="ok_button" id="map_button">${gs.getMessage('OK')}</g:di
    <hr />
    <g:dialog_button_ok ok="console.log('Dialog Button OK')" ok_type="button" />
    <hr />
    <g:dialog_buttons_ok_cancel
        ok="console.log('Dialog Button OK')"
        cancel="console.log('Dialog Button Cancel')"
        ok_id="ok_button"
        cancel_type="button" />
    <hr />
    <g:dialog_buttons_save_cancel_discard
        discard="console.log('Dialog Button Discard')"
        discard_type="button"
        cancel="console.log('Dialog Button Cancel')"
        cancel_type="button"
        save="console.log('Dialog Button Save')"
        save_type="button"/>

</j:jelly>
```

## Output



# ServiceNow Components

## Generating Record History

`g:history` can be used to generate the history view of a record. Its parameters are actually set not via attributes but as variables. This example shows the history of a record in the Incident table.

```
<j:set var="sysparm_table" value="incident" />
<j:set var="sysparm_sys_id" value="46ee8c2fa9fe198100623592c70d643e" />
<g:history />
```

### Output

#### Details for INC0000024

Created	2014-05-21 16:53:13 by david.loo
Last updated	2014-05-21 18:20:58 by glide.maint
Update count	3 (0 audited)

2014-05-21 16:53:13 Created by David Loo (188 Days 3 Hours 42 Minutes)

Field	Value
Active	false
Approval	Not Yet Requested
Assigned to	ITIL User
Business duration	16 Seconds
Business resolve time	16
Duration	16 Seconds
Resolve time	16
Category	Inquiry / Help
Closed	2014-05-21 16:53:08
Closed by	David Loo
Contact type	Phone
Escalation	Normal
Impact	3 - Low
Incident state	Closed
Knowledge	false
Location	San Diego
Made SLA	false
Notify	Do Not Notify
Number	INC0000024
Opened	2014-05-21 16:52:52
Opened by	David Loo
Priority	4 - Low
Severity	3 - Low
Short description	Issue with a web page
State	Closed
Task type	Incident
Domain	global
Urgency	3 - Low

## Record Activity

To display a records activty, similar to the Activity Formatter, you can use the `g:activity` tag. The tag will return a list of activity details dependent on the attributes passed to it.

The standard parameters to get a records activity are `sys_id`, which is the Sys ID of the record to get, `table` which is the table to get. `var` should be the variable to store the result in and `fields` which is the list of fields to retrieve.

```
<g:activity sys_id="9c573169c611228700193229fff72400" fields="" table="incident" type="changes" var="jvar_activity"/>
```

Would get all field changes for the record in the Incident table and stores the resulting list in `jvar_activity`. That variable can then be iterated over.

In the example below, you can see when iterating over the resulting object, you would use `.get(0).getValue("FIELDNAME")` to get information about the change.

Diving further you can use `g:activity` again to get Headers and Labels. Headers will store information about non-field activity, like outgoing emails. Labels will store the labels of the fields changed. The configurations is slightly different for these functions.

```
<g:activity type="header" records="${jvar_history}" var="jvar_field_header"/>
```

`type` should be either "header" or "labels", `records` should refer to the current record (the variable set in the loop). `var` again is the variable to store the result in.

In this example `jvar_history` stores each result, iterating over `jvar_history` with a `forEach` you can output the actual changes for each field.

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">

    <g:activity sys_id="9c573169c611228700193229fff72400" fields="" table="incident" type="changes" var="jvar_activity">
        <j:forEach items="${jvar_activity}" var="jvar_history">
            Created On: ${jvar_history.get(0).getValue('sys_created_on')}
            <g:activity type="header" records="${jvar_history}" var="jvar_field_header"/>
            Header: ${jvar_field_header}
            <br/>
            <g:activity type="labels" records="${jvar_history}" var="jvar_field_labels"/>
            <strong>Fields</strong>
            <ul>
                <j:forEach items="${jvar_field_labels}" var="jvar_h">
                    <li>${jvar_h}</li>
                </j:forEach>
            </ul>
            <strong>Updated Fields</strong>
            <ul>
                <j:forEach items="${jvar_history}" var="jvar_h">
                    <li>Source Table: ${jvar_h.getTableName()} - Field Name: ${jvar_h.getValue('fieldname')} - New Value: ${jva
                </j:forEach>
            </ul>
            <hr/>
        </j:forEach>
    </j:jelly>
```

## Output

Created On: 2014-12-02 19:18:41 Header:

### Fields

- Additional comments

### Updated Fields

- Source Table: sys\_audit - Field Name: comments - New Value: Test - Old Value:

---

Created On: 2014-12-02 19:14:16 Header:

### Fields

- Additional comments

### Updated Fields

- Source Table: sys\_audit - Field Name: comments - New Value: Test - Old Value:

## Outputting Notifications

Notifications are output as part of almost every page in ServiceNow but it is possible to force the notifications to output using the `g:session_notifications` tag.

This tag will add the notification boxes to the top of the page and flush all session notifications.

```
<?xml version="1.0" encoding="utf-8" ?>
gs.addInfoMessage("This is a message.");
```

```
<g:evaluate>
    gs.addErrorMessage("This is an error.");
</g:evaluate>
<g:session_notifications />
```

## Listing Record Attachments

To list a records attachments you can use the `g:attachment_list` tag. It takes `table` and `sys_id` as attributes which should point to a specific record with attachments. The tag will create a collection of attachments.

To output a specific attachment you use the `g:attachment_entry` tag, which takes an attribute `show_rename`.

This example outputs the list of attachments for a record.

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">

    <g2:attachment_list table="incident" sys_id="9c573169c611228700193229fff72400">
        <g2:for_each_record file_variable="sys_attachment" var="attachment">
            <g2:attachment_entry show_rename="true" />
        </g2:for_each_record>
    </g2:attachment_list>

</j:jelly>
```

### Output

test.txt [rename] [view]

This example will output the full Manage Attachments header.

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
    <g2:attachment_list table="incident" sys_id="9c573169c611228700193229fff72400">
        <j2:set var="jvar_attachment_url" value="javascript:saveAttachment('incident', '9c573169c611228700193229fff72400')"/>
        <g:inline template="attachment_list_body.xml" />
    </g2:attachment_list>
</j:jelly>
```

### Output

Manage Attachments (1): [test.txt](#) [rename] [view]

## Highlighting Text

`g:highlighter` will embolden search terms in a body of text. It accepts a search term in the `search` attribute. The body of the tag will be the text to search in. The result will be HTML with the search term wrapped in `<b></b>` tags. The result can either be directly output or stored in a result.

To store the result, pass a `var` attribute to the tag, to directly output omit the `var` attribute.

This example shows highlighting a word in a body of text, storing the result, then outputting the result.

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
    <g2:highlighter search="ham" var="jvar_highlighted_text">
        Bacon ipsum dolor amet venison kevin brisket beef ribs, bacon ground round chuck jowl meatloaf turkey cow. Ground round
        Pork loin tri-tip biltong fatback alcatra tongue cow shank bacon ham andouille. Strip steak short loin prosciutto sirloin
        </g2:highlighter>
        ${jvar_highlighted_text}
    </j:jelly>
```

### Output

Bacon ipsum dolor amet venison kevin brisket beef ribs, bacon ground round chuck jowl meatloaf turkey cow. Ground round **ham** hock jowl shank tenderloin doner flank sausage, alcatra corned beef... alcatra tongue cow shank bacon **ham** andouille. Strip steak short loin prosciutto sirloin beef ribs jowl

If outputting directly remember to use Phase 2 so the spaces between the bolded words are preserved.

## Dashboard Elements

You can output dashboard elements using the `g:dashboard` tag. It can take in the Dashboard Name in the `board` attribute and will output the Dashboard widgets in table cells.

This example shows outputting the "Overview" Dashboard.

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
    <table>
        <g:dashboard board="Overview"/>
    </table>
</j:jelly>
```

### Output

My Work			ITIL Summary Counts	
	▲ Number	Short description		
<input type="checkbox"/>	<input type="checkbox"/> RITM000007	A Blackberry Wireless Device	Critical Tasks	19
<input type="checkbox"/>	<input type="checkbox"/> RITM000009	HP Laserjet 4240n	Tasks that have Critical priority that are still active	
<input type="checkbox"/>	<input type="checkbox"/> RITM000010	A Blackberry Wireless Device	Overdue Tasks	13
<input type="checkbox"/>	<input type="checkbox"/> RITM000012	Loaner Laptop (T42)	The number of Tasks that are overdue	
<input type="checkbox"/>	<input type="checkbox"/> TKT0010001		Incidents Opened %3e 1 Week	33
<input type="button" value="Actions on selected rows..."/>			Active incidents that have been opened for longer than a week	
◀ ◀ 1 to 5 of 5 ▶ ▶				

Unassigned Work		
	▲ Number	Short description
<input type="checkbox"/>	<input type="checkbox"/> INC0000039	Routing to oregon mail server
<input type="checkbox"/>	<input type="checkbox"/> INC0000046	Can't access SFA software
<input type="checkbox"/>	<input type="checkbox"/> INC0010002	
<input type="checkbox"/>	<input type="checkbox"/> INC0010017	
<input type="checkbox"/>	<input type="checkbox"/> INC0010055	Reset the password for System Administrator on
<input type="checkbox"/>	<input type="checkbox"/> RF00000002	

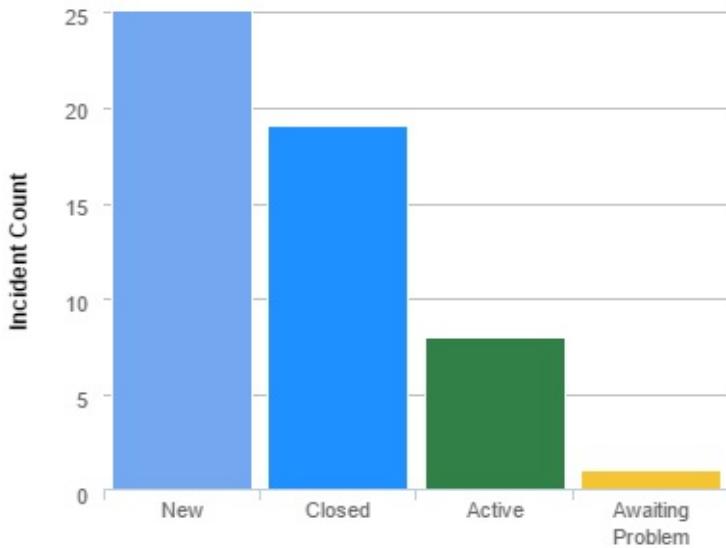
The table lacks any formatting and each element includes a "change" button, so this is probably no a good way to actually output Dashboard widgets.

## Rendering Gauges

Gauges, such as those generated on dashboards, can be rendered in Jelly using the `g:gauge`. It takes at least the Sys ID of the Gauge (from `sys_gauge`) as `id`. Other attributes are `height` and `width` to specifically set the width, although this is overridden in some cases and `type` which can also be omitted as it will be auto-detected from the Gauge record.

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
  <g:requires name="scripts/Glidev2ChartingIncludes.js" includes="true" />
  <g:gauge id="18d48b3fc6112282015f087653a9c24c" />
</j:jelly>
```

## Output



that Report types that have charts (pie,bar,etc..) require the "GlideV2ChartingIncludes.js" script be loaded as well.

## Calendars

I am including this tag because it exists but it is very specific to existing ServiceNow functionality, is not easy to use and is not configurable.

```
g:calendar will show calendar events in one of four views (Day, Week, Month, Year). The view is specified in the view.  

year, month, day can also be specified.
```

Calendar events does not seem to be configurable in this case and is limited to Field Changes.

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">

    <j2:set var="sysparm_view" value="year" />
    <j2:set var="sysparm_calview" value="year" />

    <j2:set var="sysparm_year" value="2014" />
    <j2:set var="sysparm_month" value="1" />
    <j2:set var="sysparm_day" value="1" />
    <j2:set var="sysparm_table" value="incident" />
    <j2:set var="sysparm_query" value="active=true" />
    <j2:set var="sysparm_cal_field" value="sys_updated_on" />

    <g2:calendar view="[$[sysparm_calview]" year="[$[sysparm_year]" month="[$[sysparm_month]" day="[$[sysparm_day]" />

    <j2:if test="[$[sysparm_calview == 'month' || sysparm_calview == '' || sysparm_calview == null]">
        <j2:set var="jvar_month" value="[$[jvar_calendar.firstMonth]" />
        <g2:inline template="calendar_month.xml"/>
    </j2:if>
    <j2:if test="[$[sysparm_calview == 'week']">
        <g2:inline template="calendar_week.xml"/>
    </j2:if>
    <j2:if test="[$[sysparm_calview == 'day']">
        <g2:inline template="calendar_day.xml"/>
    </j2:if>
    <j2:if test="[$[sysparm_calview == 'year']">
        <g2:inline template="calendar_year.xml"/>
    </j2:if>

</j:jelly>
```

## Output

January 2014							February 2014							March 2014						
Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun
1	2	3	4	5			3	4	5	6	7	8	9	3	4	5	6	7	8	9
10	11	12	13	14	15	16	15	16	17	18	19	20	21	17	18	19	20	21	22	23
24	25	26	27	28	29	30	24	25	26	27	28	29	30	24	25	26	27	28	29	30
April 2014							May 2014							June 2014						
Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun
1	2	3	4	5	6	7	1	2	3	4	5	6	7	1	2	3	4	5	6	7
8	9	10	11	12	13	14	8	9	10	11	12	13	14	9	10	11	12	13	14	15
15	16	17	18	19	20	21	15	16	17	18	19	20	21	16	17	18	19	20	21	22
22	23	24	25	26	27	28	22	23	24	25	26	27	28	23	24	25	26	27	28	29
29	30	31					29	30	31					29	30					
July 2014							August 2014							September 2014						
Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun
1	2	3	4	5	6	7	1	2	3	4	5	6	7	1	2	3	4	5	6	7
8	9	10	11	12	13	14	8	9	10	11	12	13	14	8	9	10	11	12	13	14
15	16	17	18	19	20	21	15	16	17	18	19	20	21	15	16	17	18	19	20	21
22	23	24	25	26	27	28	22	23	24	25	26	27	28	22	23	24	25	26	27	28
29	30	31					29	30	31					29	30					
October 2014							November 2014							December 2014						
Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun
1	2	3	4	5			1	2						1	2	3	4	5	6	7
8	9	10	11	12			8	9	10	11	12	13	14	8	9	10	11	12	13	14
15	16	17	18	19			15	16	17	18	19	20	21	15	16	17	18	19	20	21
22	23	24	25	26			22	23	24	25	26	27	28	22	23	24	25	26	27	28
29	30	31					29	30	31					29	30					

`g:calendar_event_id` seems like it should be able to add an event but there is no documented usage of it. The `calendar` tag does have event but it looks like there is no way to inject custom events using `calendar_event_id`.

## Form Section Lists

`g:section_list` outputs a standard ServiceNow form header. There are three attributes which can be **true** or **false**. `headerOnly` , `headerSpacer` , `headerSimple` which all configure how the header is output.

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
    <g:evaluate var="incident" copyToRhino="true" object="true">
        var gr = new GlideRecord('incident');
        gr.query();
        gr.next();
        gr;
    </g:evaluate>

    <j:set var="ref" value="incident" />
    <g:section_list headerOnly="true" headerSpacer="false" headerSimple="false"/>
</j:jelly>
```

## Output



## Privilege Escalation

`g:elevate_privilege` will output a button and dialog window to escalate privileges. The function `g:elevated_role_dialog` outputs the same dialog as `elevate_privilege` directly without a triggering button.

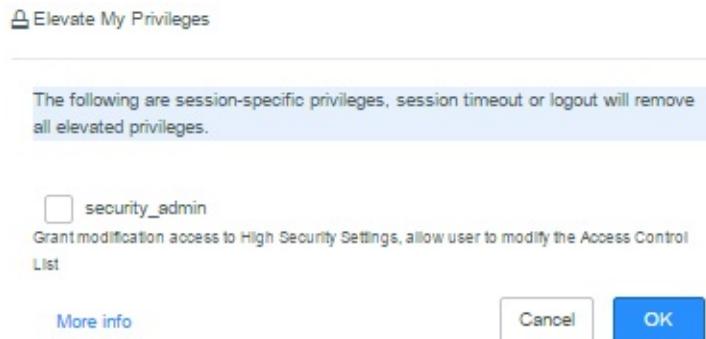
```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">

    <g:elevate_privilege />$[SP]Elevate My Privileges
    <hr />
    <g:elevated_role_dialog />

</j:jelly>
```

## Output

# ServiceNow System Functions



## Impersonation Functionality

Similar to privilege escalation there are three function which will output the UI elements to Impersonate Users.

A user will still need the have the ability to impersonate users.

To generate the button to impersonate use `g:impersonate`. To output the Impersonate User pop up dialog only use `g:impersonate_dialog`. To generate a list of impersonation choices use `g:impersonate_choices` which can take a parameter `history` whose value is a list of recently impersonated users.

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">

    <g:impersonate_button />
    <hr />

    <select>
        <g:impersonate_choices />
    </select>

    <hr />
    <g:impersonate_dialog />

</j:jelly>
```

## Output

## Domain Picker

To output the Domain Picker there are a couple functions `domain_select` and `domain_reference_picker` which depend on the Domain Separation Plugin being enabled.

These are always paired with a script which handles the event triggered when changing domains from the picker.

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">

<j:if test="${pm.isActive('com.glide.domain')}">
  <g:requires name="scripts/DomainChangeNotificationHandler.js" />
  <g:domain_select />
  <g:domain_reference_picker />
</j:if>

</j:jelly>
```

This requires the Domain Separation Plugin which I did not have when running this example.

## Encryption Picker

If the Encryption Plugin is enabled, use `encryption_select` to output the Encryption Context Picker.

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">

<j:if test="${pm.isActive('com.glide.encryption')}">
  <g:encryption_select />
</j:if>

</j:jelly>
```

This requires the Encryption Plugin which I did not have when running this example.

## Mobile Version Switch

To output the Mobile Version switch use `g:mobile_version_switch`.

This only displays when in Mobile to toggle to the Desktop version.

## Theme Switcher

To output the Theme Switcher use `g:ui_theme_changer`.

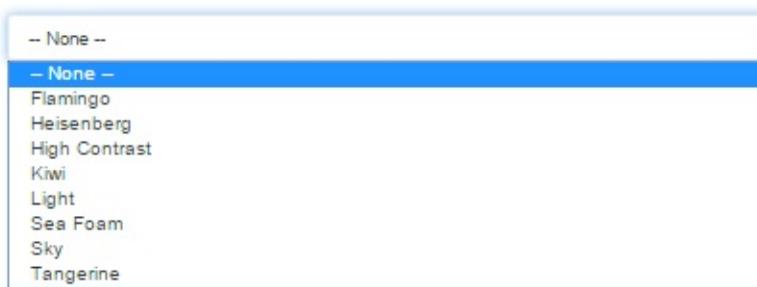
```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">

<g:ui_theme_changer />

</j:jelly>
```

## Output

## Theme



## Language Picker

If the Internationalization Plugin is enabled, `g:ui_language_select` will display the Language Picker.

```
<g:ui_language_select />
```

## Update Set Picker

`ui_update_set_picker` outputs the Update Set picker but does not seem to work properly on its own.

```
<g:ui_update_set_picker />
```

Styling is applied by default to hide the picker.

## Timezone Picker

`ui_timezone_changer` will output the Timezone Changer and like the Update Set Picker it will not work on its own.

```
<g:ui_timezone_changer />
```

Styling is applied by default to hide the picker.

An example usage of this function and all Header Decorations is in the UI Macro `decorate_welcome_header_stripe`.

## Context Menu

The macro `view_context` generates the View Context switcher used on forms and lists and is used in the UI Macro `context_form_header`.

It takes the parameters `id` which is the ID of the Context Header, `table` which is the current table, `type` (always set to "section" and `first_form_section` which is the id of the first section on the form.

`history_context`, `label_context` and `template_context` are all functions that output their respective options to the Context Menu.

## Full Text Search Components

Generating a Search Widget can be done using `g:text_search_widget` which takes a parameter `size` to indicate it's width. Although you can just use it, it should be wrapped in a form.

Globally the form submits to `textsearch.do`, within the CMS it submits to `content_search.do`

Here is an example modified to run on its own.

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
    <span>Enter your term</span>
    <form action="textsearch.do">
        <span>
            <input type="hidden" id="sysparm_tsgroups" name="sysparm_tsgroups" value="" />
            <input type="hidden" id="sysparm_view" name="sysparm_view" value="text_search" />
            <g:text_search_widget size="10"/><br/>
            <input type="submit" />
        </span>
    </form>
</j:jelly>
```

## Output

The screenshot shows a search interface. At the top, there is a text input field with the placeholder "Enter your term". Below the input field is a "Search" button with a magnifying glass icon. To the right of the search button is a dropdown menu icon. At the bottom of the form is a "Submit" button.

The `global_text_search_widget` has exactly the same implementation, and seems to output the same thing.

## Log File Browser Form

`g:log_file_browser` will output the top form from the Log File Browser page. It is only usable within the context of that page as it depends on it's Client Script.

## Miscellaneous Functions

### Process Flow Formatter

The `process_flow` macro uses the `flow_formatter` function which takes 3 parameters: `var` which stores the result of the function, `table` which is the table that is being references and `current` which is a reference to a record.

The function uses the `sys_process_flow` table as the source and returns a list of flow stages.

### Distribution List

This is a weird tag whose name has nothing to do with its actual functionality. `get_distribution_list` is used in generating Glide Lists. It takes `ref` as a reference to a GlideRecord and `record` which takes a list of values (comma separated). Depending on the usage it may also take `type` set to "current" for editable lists, `readonly` set to "true" if the element is disabled and `ensure` which is always set to "true" with `readonly`.

`get_distribution_list` is used in the `lightweight_glide_list` UI Macro.

## GWT Select List

`g:gwt_select_list` generates a multi-select list used in Slush Buckets and other similar areas. It takes `name` which is the control name, `dropzone` which should be set to true, `readonly` to control whether the control is editable, and optionally `unique` to control whether the list can contain duplicates.

The `gwt_select_list` is heavily dependent on Scripts. A good example is in the UI Macro `ui_dnd_slushbucket`.

Also used in conjunction with the GWT Select Lists is the `ui_select_list_addremove_control`, which generates a button set to manipulate the lists. `name`, `selecttable`, `leftzone` and `rightzone` are its parameters which should match up with what `gwt_select_list` generates

```
<g:ui_select_list_addremove_control name="source_fields" selecttable="selectTableSource" leftzone="source_fields_leftzc...
```



## Scrolling Area

To create a scrolling section (like with News) you can use `scrollable_area` which takes `height` as parameter. The body of the tag will scroll repeatedly.

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">

    <g2:scrollable_area height="100px">
        <p>Here Is Some Content</p>
    </g2:scrollable_area>

</j:jelly>
```

It would be impossible to show this scrolling in an output, so you will just need to take my word it scrolls!

## OLAP Cube Functions

*Possibly Deprecated*

Two functions exist that seem to be part of the OLAP Cube functionality that is no longer functional (or is being worked on?)

`g:olap_view` and `g:olap_cube_options` are used in the OLAP Pages, but are not useful anywhere else.

## CI Relationship Form

`g:ci_relationship_manage` will output the CI Relationship Manager form as used in the UI Page `ci_relationship_manage`.

```
<g:ci_relationship_manage/>
```

## System Properties

To create the System Properties pages the `g:system_properties` tag is used. It takes two parameters `includes` and `excludes`. Both must be included however the usage is tricky. If trying to display a specific set of Property categories, specify them in the `include` tag as a comma-separated list. Specifying individual property names in `exclude` seems to return the whole list of properties except the excluded ones.

This example will output all the CSS Properties.

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
  <g2:system_properties exclude="" include="CSS" />
</j:jelly>
```

## Output

Banner text color:

Banner and list caption background color.  
CSS color formats:

- name - predefined color names: red, green, blue, etc.
- RGB decimal - RGB(102, 153, 204)
- RGB hex - #223344

Font used in forms and lists:

Base font size (points):

Button background color:

Button border color:

Button border width:

Button text color:

An even easier way to output properties by Category with the headers is to use `g:system_properties_categorized`. This accepts one attribute `category` which takes a comma-separated list of categories to include.

This example will output both the CSS and Cookie categories.

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
  <g2:system_properties_categorized category="CSS,Cookies" />
</j:jelly>
```

## Output

Customization Properties for overriding values in the CSS file

Banner text color:

Banner and list caption background color.  
CSS color formats:

- name - predefined color names: red, green, blue, etc.
- RGB decimal - RGB(102, 153, 204)
- RGB hex - #223344

Font used in forms and lists:

Base font size (points):

Button background color:

Button border color:

Button border width:

Button text color:

## Using Translated Messages

ServiceNow has the ability to store all text in multiple translations. These messages are stored in the "sys\_ui\_message" table. To use these translations in custom pages you can use the `g:messages` tag. The tag will return a script tag which sets

the values of the tags for use on the client side.

The particular values returned will depend on the users selected language and the presence of a proper value in the 'sys\_ui\_message' table. If no value exists for the users language the English version is used.

In this example the translated word for "Run" is returned and used on the page.

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
    <g:messages>
        Run
    </g:messages>
    <script>
        console.log( new GwtMessage().getMessage("Run"));
    </script>
</j:jelly>
```

## Output

```
<script>
    addLoadEvent(function() {var messages = new GwtMessage();messages.set('Run', 'Carrera');})
</script>
```

## Console Output

Carrera

## Getting Unique Field Values

`g:ref_element` gets a list of unique values for a particular field. It takes three attributes: `name` which should be the table and field separated by a period (`incident.short_description`), `var` which is the variable to store the result in and `query` which is the encoded query to be run. `query` may be blank.

In this example the list of unique Short Descriptions for Incident are returned and listed out.

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
    <g:ref_element name="incident.short_description" var='jvar_test' query="active=true" />
    <ul>
        <j:forEach items="${jvar_test}" var="jvar_item">
            <li>${jvar_item}</li>
        </j:forEach>
    </ul>
</j:jelly>
```

## Output

```
*
```

- \* Can't access SFA software
- \* Can't get to network file shares
- \* Can't launch X-Win32
- \* Can't log into SAP
- \* CPU load high for over 10 minutes
- \* EMAIL Server Down
- \* Exchange server appears to be down...lots of users impacted
- \* Having problems with performance on the Sales Tools
- \* How do I create a sub-folder
- \* I can't get my weather report

```
* I can't launch my game anymore
* I need more memory
* Issue with email
* JavaScript error
* missing my home directory
* My cubical phone does not work
* Need access to sales db for the west
* Network storage unavailable
* please remove this hotfix
* Rain is leaking on main DNS Server
* Request for a Blackberry
* Request for a new service
* Reset the password for System Administrator on
* Routing to oregon mail server
* Sales forecast spreadsheet is READ ONLY
* SAP Financial Accounting application appears to be down
* SAP Sales app is not accessible
* The SAP Human Resources application is not accessible
* There seems to be some slowness or an out to SAP Materials Management
* User can't access SAP Controlling application
* Wireless access not available on floor 3
```

## Getting Email Client Canned Messages

Yep theres a function for that `g:get_canned_messages` takes no parameters and will set a variable `jvar_message_list` to a list of all the Canned Messages in the System.

*Canned Email Messages are stored in the table `sys_email_canned_message`*

## Label Spacing

`g:label_spacing` is used throughout and the description for this function states "Provide spacing for labels via the use of a 0 pixel wide field."

This may be a technique for a clearfix or literally to just add a tiny bit of spacing to a label.

```
<g:label_spacing />
```

## Output

```
<input id="make_spacing_ok" style="visibility:hidden; width:0px;" title="" />
```

## Merge Tags Form

`g:merge_tags_input` has only a single use throughout the system and appears to be a form used to merge record tags.

## Progress Workers

`ui_progress_worker` will output an AJAX Progress Watcher for a given Worker ID. Its parameters are `worker_id` and `show_cancel` which is a boolean to control whether the underlying worker is cancellable.

```
<g:ui_progress_worker worker_id="${sysparm_pworker_sysid}" />
```

There is no indication on how to find the Worker ID.

`g:mid_tools_progress_worker` is a more specialized tag which outputs the AJAX Progress watcher when executing scripts

on the MID Server. It takes a parameter `ecc_queue_id` which should be set to the sys id of the ECC queue job record.

```
<g:mid_tools_progress_worker ecc_queue_id="${jvar_ecc_queue_id}" />
```

## Import Export Mapping

A couple tags exist specifically for the Import Export Mapping Feature. These are used exclusively in the Mapping Assist function. They are `g2:mapping_list` and `g2:mapping_sample_data`.

`g2:mapping_list` will return a choice list of fields. These fields are pulled from the Import Export map and are based off of the Target Table and the External Fields. The `source` attribute will set where exactly to pull from and can be "database" or "selected" and the type will return the subset of those fields either "selected" or "available"

`g2:mapping_sample_data` depends on the datasource actually having an attached file where it can pull sample data from. It takes no attributes and will store the sample data in the variable `jvar_import_fields`.

Here is an example that was set up using a test Import Export Map.

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
    <j2:set var="sysparm_sys_id" value="8da2f3db2b74b1009728f70a89da157d" />
    Available Fields
    <select>
        <g2:mapping_list type="available" source="database" />
    </select>
    <br/>
    Selected Fields
    <select>
        <g2:mapping_list type="selected" source="database" />
    </select>
    <br/>
    Selected External Fields
    <select>
        <g2:mapping_list type="selected" source="selected" />
    </select>
    <br/>
    Available External Fields
    <select>
        <g2:mapping_list type="available" source="selected" />
    </select>
    <br/>
    <g2:mapping_sample_data />
    <j2:forEach var="jvar_field" items="${jvar_import_fields}">
        ${jvar_field.name}
        <ul>
            <j2:forEach var="jvar_field_value" items="${jvar_field.value9}">
                <li>${jvar_field_value}</li>
            </j2:forEach>
        </ul>
    </j2:forEach>
</j:jelly>
```

## Output

Available Fields Active ▾

Selected Fields priority ▾

Selected External Fields pri ▾

Available External Fields ▾

col1

- hello
- N/A

col2

- world
- N/A

col3

- !
- N/A

In order to reproduce this you will need to set up a Data Source in the system and create an Import Export map. The settings in the Import Export map will drive these functions.

## Undocumented Functions

### `attachment_encrypt`

Appears in dialogs for attaching files. Directly running this has no output.

# Undocumented Tags

Tags in this section lack sufficient documentation to determine their usage or have been deprecated.

## Tags With No Documented Usage

- action\_filter
- help
- insert\_form
- hdrftr\_layout
- hdrftr\_cell\_data
- report\_choice\_list
- get\_action\_list
- get\_connectors
- get\_distribution\_list
- wizard

## Tags with Limited Documentation

### `scripted_table_options`

`scripted_table_options` is used in the UI Macro `document_selector`, it takes two parameters `selected` which seems to be the table, and `script`. While the table is obvious to find out there is no documented usage of `script`.

```
<g:scripted_table_options selected="${table_name}" script="${tableChoicesScript}" />
```

### `element_list`

```
<g:evaluate>
    var incidentRec = new GlideRecord('incident');
    incidentRec.query();
    incidentRec.next();
    incidentRec.next();
    incidentRec;
</g:evaluate>
<j:set var="ref" value="incident" />

<g:element_list section_id="991f88d20a00064127420bc37824d385" table="incident">
</g:element_list>

${jvar_p2flow_all_visible_fields}
```

## Creating A Partial Page

`partial_page` wraps a section in a "partialPage" span tag. This tag seems to have been used when creating charts.

```
<g2:partial_page name="myPartial">
    <h4>Hello World</h4>
</g2:partial_page>
```

## Menu Lists

`g:menu_lists` outputs a mostly broken Menu List. It takes as a parameter `device_type` which can be **mobile** or **browser**.

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
    <g:menu_lists device_type="browser" />
</j:jelly>
```

## List Controls

`g:list_control` sets some variables pertaining to List views. No documented usage could be found.

```
<g:evaluate var="incident" copyToRhino="true" object="true">
    var gr = new GlideRecord('incident');
    gr.addQuery("active","true")
    gr.query();
    gr.next();
    gr;
</g:evaluate>

<g:list_control ref="incident" />

Control ID: ${jvar_list_control_id}<br />
Sys ID: ${jvar_list_control_sys_id}<br />
Control Query: ${jvar_list_control_query}<br />
Use Relationship Banner: ${jvar_use_relationship_banner}<br />
Relationship Banner: ${jvar_relationship_banner}<br />
List Edit Type: ${jvar_list_edit_type}<br />
Ref Qualifier Tag: ${jvar_list_edit_ref_qual_tag}<br />
```

## Output

```
Control ID: 91a807a60a0a3c74012113e28b47ca2e
Sys ID: 91a807a60a0a3c74012113e28b47ca2e
Control Query:
Use Relationship Banner:
Relationship Banner:
List Edit Type:
Ref Qualifier Tag:
```

## Rendering a Page

`g:html_page` seems to be deprecated. It depends on two variables `jvar_form_name` and `jvar_html_template_name`. Running the function results in an error **The element type "table" must be terminated by the matching end-tag ""**. within the included template. No documented usage could be found.

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
    <j:set var="jvar_form_name" value="incident.9c573169c611228700193229fff72400" />
    <j:set var="jvar_html_template_name" value="form.xml" />

    <g:html_page />
</j:jelly>
```

## LDAP Details

`g:ldap_detail` does work but it was created specifically for the LDAP Browse functionality and has no use outside of that

page. It takes `server`, `ou` and `dn` as attributes.

## Sample Usage

```
<g2:ldap_detail server="${sysparm_server_id}" />
<ul>
    <j2:forEach var="jvar_attr" items="${jvar_ldap_attributes}">
        <li>${jvar_attr.label} - ${jvar_attr.value}</li>
    </j2:forEach>
</ul>
```

## Rendering Components

The only available documentation implies `g:render_component` should render an Abstract Component but gives no list or indication what this refers to. There is no documented usage.

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
    <g:render_component componentName="XXXXXX" />
</j:jelly>
```

## Wizard Choice List

`g:wizard_choice_list` has no documented usage. It seems as if it should pull choices for a Wizard Choice field or Interceptor Choices. The only attribute is `answer` which immediately causes an error "**Property 'answer' has no write method: java.lang.IllegalArgumentException**".

## Getting Collection Data

`g:get_collection_data` has limited documented usage.

```
<g:get_collection_data ref="incident" id="9c573169c611228700193229ffff72400" key="number" related_field="parent" related
```

## Remote Import

`remote_import` has no documented usage. The name would imply the function would import a remote record, possibly from another instance.

## With

`g:with` appears to be a function to initialize a scope which references a particular object. No example can be found and it is usually a best practice to avoid `with` operators.

## Item Link

`g:item_link` is partially documented. It must be contained in another function and has a configurable option `show_link` to control whether or not a link is generated.

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
    <g:item_link show_link="true" />
    ${jvar_show_link}
```

```
</j:jelly>
```

# Debugging

Just like any other language Jelly can have bugs, but unlike other languages there is no good native way to debug Jelly. In this section I will document some of the more common bugs in Jelly and how to resolve them.

## Page Fails to Render Completely

Pages can sometimes fail to render completely because of an exception in Jelly. These errors can be seen in the Log Table and usually are a result of a missing attribute.

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
    Break 1
    <j:set />
    Break 2
</j:jelly>
```

### Output

```
Break 1
```

### Log Error

```
<j:set> Either a 'var' or a 'target' attribute must be defined for this tag: org.apache.commons.jelly.JellyTagException
```

### Resolution

The exception on the line must be fixed. In this case a missing attribute needs to be added.

### Log Error

```
java.lang.NullPointerException:
```

### Resolution

An attribute specified on a Jelly tag is undefined. If using an expression make sure the expression evaluates to something by outputting it before the tag.

## Evaluator Errors

Evaluators errors indicate an issue with a JEXL Expression. Errors in the log with source "Evaluator" can be traced back to the offending line in a Jelly file.

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
    <j:set var="jvar_test" value="${myVar}" />
    ${jvar_test}
```

```
</j:jelly>
```

**Log Error** org.mozilla.javascript.EcmaError: "myVar" is not defined. Caused by error in at line 1

```
==> 1: myVar
```

The error does not tell us which file or which line in the file the error occurs on so it will be necessary to match the variable name to the expression in the file and work from there.

Evaluators are used in more places than just Jelly Pages, they are used throughout the system.

## Processing Errors

### Mismatched Ending Tag

```
The element type "START" must be terminated by the matching end-tag "END".
```

This error indicates that a tag was opened but never closed. This can be either a Jelly tag or an HTML tag.

#### Causes

- Closing tag omitted

```
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
    <h1>
</j:jelly>
```

- Self closing tag trailing slash

```
<br>
<h1>Hello World</h1>
```

- Mis-matched Phase

```
<g:evaluate>
    // some script
</g2:evaluate>
```

#### Resolution

The resolution is to end the tag properly but finding the offending tag can often be confusing. One easy way is to "beautify" the code and indent the tags properly. This will make it easier to see if a tag is missing, and if tags are lined up, it is easier to see if the tags match.

ServiceNow will output the error as well indicating the line the error occurs on. If the closing tag is simply the wrong phase, it will be easy to identify in this error. If the error indicates a completely different tag it is likely that the closing tag was omitted.

A last way is to use the W3C Validator which will give a more friendly error message and slightly more details.

## Attribute Already Specified

```
Attribute "NAME" was already specified for element "TAG".
```

This error indicates that an attribute was specified twice on a tag which is considered invalid XML. XML cares about case so "STYLE" is not the same as "style".

### Causes

- Attribute specified twice

```
<h1 style="" style="">Hello World</h1>
```

### Resolution

Look carefully, sometimes if the attributes are not next to each other or the line is wrapped, it can be difficult to see them.

## Attribute Missing Value

```
Attribute name "NAME" associated with an element type "TAG" must be followed by the ' = ' character.
```

This error indicates that an attribute was specified but no value was entered. This is considered invalid XML, although it is valid HTML in most cases. Attributes will sometimes be specified on HTML elements without values and while the browser handles these correctly, the XML Parser in ServiceNow will not.

### Causes

- Attribute specified without value

```
<input type="checkbox" value="HelloWorld" checked>
```

### Resolution

Specify the value or if the attribute does not use a value, specify true.

```
<input type="checkbox" value="HelloWorld" checked="true">
```

## Invalid Namespace

```
Attribute "j" bound to namespace "http://www.w3.org/2000/xmlns/" was already specified for element "j:jelly".
```

This error occurs when the `j:jelly` tag does not have its namespaces specified correctly.

### Causes

- Mis-typed `j:jelly` tag

```
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j="null" xmlns:g="null">
```

### Resolution

Fix or remove the second namespaces. `j` and `g` should become `j2` and `g2` on the 3rd and 4th specified namespaces.

```
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
```

## Processing Instruction Error

The processing instruction target matching "[xX][mM][lL]" is not allowed.

This error indicates that the XML Parser was expecting the first line of the file to be the Doctype Declaration but found another tag.

### Causes

- Tag occurs before the Doctype Declaration

```
<h1>Hello World</h1>
<?xml version="1.0" encoding="utf-8" ?>
```

### Resolution

Remove all tags and whitespace occurring before the Doctype Declaration.

## Content in Prolog

Content is not allowed in prolog.

This error indicates that some text is occurring before the Doctype Declaration.

### Causes

- Content exists before the Doctype Declaration

```
Hello World
<?xml version="1.0" encoding="utf-8" ?>
```

### Resolution

Remove all text and whitespace before Doctype Declaration.

## Unspecified Entities

The entity name must immediately follow the '&' in the entity reference.

The XML parser expects that & is the start of an entity. When no entity is found it causes an error.

### Causes

- Ampersand used in HTML or JavaScript

```
<h1>This & That</h1>
```

### Resolution

Use the escaped entities \${[AMP]} and \${[AND]} to correctly output the ampersand character.

## XML is not well formed

The content of elements must consist of well-formed character data or markup.

This error indicates that the XML is corrupt somehow.

### Causes

- A less than or greater than symbol is used in text or JavaScript

**Hello World <--- <="" h1="">**

---

### Resolution

Replace the < with \${[LT]} and > with \${[GT]} .

## Parser Errors

---

### Unspecified Entities

The entity name must immediately follow the '&' in the entity reference.

This error is the same as the Processing Script error but it occurs specifically when valid XML in the first phase generates invalid XML in the second phase.

### Causes

- Unintentionally passing invalid entities to the second phase

```
<h1>This ${AMP} that</h1>
```

---

### Resolution

Replace the \${AMP} with the second phase \${[AMP]} .

# Debugging Tags

This section covers tags that simplify debugging Jelly Pages. There is also a section in this chapter on other debugging techniques.

## Setting Breakpoints

`g:breakpoint` allows us to output variables and messages to a log. It can take a few attributes which are all optional. `id` which is included in the output making it easier to identify. `message` which will output a message to the log. `var` which will output a specific variable.

If you omit `var` then every variable in the context will be output.

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
    <j:set var="jvar_test_var" value="FooBar" />
    <g:breakpoint var="jvar_test_var" id="My Breakpoint" />
</j:jelly>
```

### Log Output

```
<<< BREAKPOINT TAG START <<< My Breakpoint
JVAR: jvar_test_var(java.lang.String) FooBar
>>> BREAKPOINT TAG END >>> My Breakpoint
```

To view log output in real-time use the Log Tail Page `channel.do?sysparm_channel=logtail`

This example outputs all variables in context.

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
    <j:set var="jvar_test_var" value="FooBar" />
    <g:breakpoint message="Outputting all Variables" id="My Breakpoint" />
</j:jelly>
```

### Log Output

```
<<< BREAKPOINT TAG START <<< My Breakpoint
JVAR: AMP=&
JVAR: AND=&&
JVAR: GT=&gt;;
JVAR: LT=&lt;;
JVAR: RP=FFFFFFFFFFFF250350nullnullnullnulljelly_set.donullFFFnullnullnullF
JVAR: SP=&nbsp;
JVAR: context=com.glide.ui.jelly.GlideJellyContext@111f3a0
JVAR: gc=com.glide.script.GlideController@191abac
JVAR: jvar_base_uri=https://server.service-now.com/
JVAR: jvar_cdata_end=]]>
JVAR: jvar_cdata_start=<![CDATA[
JVAR: jvar_form_sys_id=e0e98d472b1071009728f70a89da1549
JVAR: jvar_last_transaction_time=0
...
Outputting all Variables
>>> BREAKPOINT TAG END >>> My Breakpoint
```

This tag is intended for debugging purposes. Remember to remove it as soon as possible to prevent accidentally pushing it to production.

## Advanced Usage

---

This section covers some advanced and experimental techniques using Jelly in ServiceNow. It is important to note that as ServiceNow changes some of these techniques may become deprecated.

## Glide and Jelly JavaScript Objects

---

This section covers using the Jelly specific Glide API JavaScript objects.

# Executing Jelly

All the examples we have seen up until this point have been either in UI Pages or UI Macros but it is possible to execute Jelly programmatically in a Server Side JavaScript context such as Business Rules, Script Includes or Processors.

This is done using the `GlideJellyRunner` class. In this section I will document the methods and usage of this class and show some examples of how you can use this on your own.

The examples in this section will be set up to use Background Scripts for simplicity. Remember to be careful and only run experimental scripts in non-production environments.

## Basic Usage

To initialize the `GlideJellyRunner` create a new instance of it.

```
var runner = new GlideJellyRunner();
```

## Running Macros

Jelly can be run in a few ways, the first we will look at uses a UI Macro as the source. `runMacro` takes the name of the macro as a parameter and returns the executed Jelly as a result.

```
runner.runMacro('macro_name');
```

It depends on a **UI Macro**:

**Name:** runner\_test

### XML

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
    Hello World.
</j:jelly>
```

### Background Script

```
var runner = new GlideJellyRunner();
var result = runner.runMacro('runner_test');
gs.print(result);
```

### Output

```
*** Script:      Hello World.
```

## Running Jelly From Scripts

`run` and `runFromScript` both do the same thing and that is run a Jelly script passed as a string. The Jelly script will still

need to be valid XML including the Prolog and `jelly` tags.

Here is the same example again using this method, contained completely in JavaScript

### Background Script

```
var jellyScript = '<?xml version="1.0" encoding="utf-8" ?>';
jellyScript += '<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">';
jellyScript += 'Hello World.';
jellyScript += '</j:jelly>';

var runner = new GlideJellyRunner();
var result = runner.run(jellyScript);
gs.print(result);
```

### Output

```
*** Script: Hello World.
```

## Running Jelly From Templates

`runFromTemplate` will run Jelly contained in an XML template on the file system.

This example outputs the "glide\_user.xml" template.

```
var runner = new GlideJellyRunner();
var result = runner.runFromTemplate('glide_user.xml');
gs.print(result);
```

### Output

```
*** Script: <script>
    addTopRenderEvent( function() {
        window.g_user = new GlideUser('admin',
            'System',
            'Administrator',
            'maint,admin',
            '6816f79cc0a8016401c5a33be04be441');

    });</script>
```

### Available Templates

This list is subject to change between versions and does not document the expected variables in each case.

Templates		
actions2	action_body	activity_toggle
annotation_line_separator	annotation_section_separator	annotation_user_text
approval_summarizer_master	app_menu	app_menu_title
app_module	app_module_children	app_module_context
app_module_default	app_module_filter	app_module_html
app_module_label	app_module_link	app_module_menu_list

app_module_script	app_module_separator	app_module_show_list
attachment_list	attachment_list_body	attribute_roCollapse
calendar_day	calendar_dayofmonth	calendar_dayofweek
calendar_icons	calendar_month	calendar_simple_dayofmonth
calendar_simple_month	calendar_week	calendar_year
client_messages	client_transaction	clonetitle
collapsing_image	delete_rows_includes	diff_html_viewer
diff_version_html_viewer	dir.txt	dir_checker
dynamic_filter_options	element	element_attributes
element_attribute_knowledge	element_attribute_pick_list	element_auto_number
element_boolean	element_choice	element_choice_radio_active
element_color	element_color_display	element_conditions
element_date	element_date_time	element_debug
element_debug0	element_default	element_document_id
element_due_date	element_email	element_email_script
element_external_names	element_field_list	element_field_name
element.glide_action_list	element.glide_duration	element.glide_encrypted
element.glide_list	element.glide_time	element_hex
element_image	element_integer_date	element_internal_type
element_journal	element_journal_input	element_journal_list
element_multi	element_multi_small	element_multi_two_lines
element_number	element_password	element_password2
element_ph_number	element_pick_list	element_pick_list_ext
element_radio	element_reference	element_reference_choice_functions
element_reference_lookup	element_reference_name	element_reference_read_only
element_reference_read_write	element_reminder_field_name	element_row_multi
element_row_single	element_script	element_script_plain
element_slushbucket	element_slushbucket_choices	element_slushbucket_macro
element_string_boolean	element_sysevent_name	element_table_name
element_template_value	element_timer	element_url
element_user_image	element_user_image_h	element_user_roles
element_variables	element_variable_conditions	element_video
element_wide_text	email_client_attachments	email_client_template_simple
email_client_toolbar	email_detail	expandCollapse
field_type_options	filter_description_loader	filter_options
form	form_label	form_multiple_update
form_owned_by	form_search	get_can_write_function
get_group_detail_list	get_label_function	get_target_form_function

glide_user	hidden_parm	html_page
html_page	html_page_direct	html_page_fontsizer
html_page_header	html_page_meta	html_page_onready
html_page_plugin_contributions	html_page_properties	html_page_response_time
html_page_script_bottom	html_page_script_core	html_page_script_db
html_page_script_globals	html_page_script_includer	html_page_stack_mgmt
html_page_title	html_READONLY	html_READONLY_DATE
id_to_icon_path	ie_checker	image_browse_cell
incident_audit_relation	int_end_section	int_form
int_list	int_section	int_vspliter
item_body_cell	item_body_cell_calc_style	item_body_cell_default
item_body_cell_pick_list	item_body_cell_ref_list	item_color_display
item_counter	item_document_id	item_field_value
item.glide_action_list	item.glide_date_time	item_html
item_link	item_link_default	item_link_reference
item_percent_complete	item_table_name	item_url
item_user_image	item_video	item_workflow
javascript_includes	lightweight_reference	list_aggregates
list_body_cell_globals	list_cell_functions	list_ci_tree_picker
list_context	list_edit_action_buttons	list_edit_js_variables
list_filter_save	list_general_tree_picker	list_group_tree_picker
list_text_search	list_tree_picker	list_white_space
load_baseline_dates	load_choice_lists	load_filter_description
load_metadata	load_parsed_query	multiline_collapse
navigation_link	navigation_links	navigation_link_body
navpage	navpage_bottom	navpage_debug_tools
navpage_preview	navpage_stripe	navpage_top
navpage_top_banner	navpage_user	output_messages
output_messages2	page_timings	record_nav_buttons
reference_autocomplete_setup	reference_detail_decoration	ref_contributions
script_editor_toggle	script_go_to_line	script_tree_toggle
section	section_context	section_context_actions
section_head_right	session_debug	set_attachment_jvars
set_m2m_jvars	set_ref_can_write	set_ref_in_rhino_context
set_ref_meta_info	set_ref_multi_rows	set_sort_jvars
set_view_title	slushbucket_add_field	slushbucket_annotations
slushtitle	system_property	sys_gaugecount_template

sys_gaugelist_template	sys_gaugeurl_template	sys_gauge_template
sys_gauge_template_header	sys_m2m_body	sys_popup
sys_print_button	table_name_read_only_display	tabs2
tabs2_toggle	textarea_sizer	ui_policy
update_set	use_js_editor	vsplit
vsplitter	welcome_content	wizard
wizard_section	workflow_scheduling	

## Setting Variables

Variables can be passed directly into the Jelly Context using the `setVariable` method. It takes the name and value as parameters.

```
runner.setVariable('jvar_variable','value');
```

This example modifies the previously created `runner_test` macro.

### UI Macro: runner\_test

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
  <j:if test="${jvar_show}">
    Hello World.
  </j:if>
</j:jelly>
```

### Background Script

```
var runner = new GlideJellyRunner();
runner.setVariable('jvar_show','true');
var result = runner.runMacro('runner_test');
gs.print(result);
```

### Output

```
*** Script:
Hello World.
```

## Escaping Values

By default the runner will escape values so characters like `>` will be converted to `&gt;`. This can be configured using the `setEscaping` method.

```
runner.setEscaping(false);
```

### Background Script

```

var jellyScript = '<?xml version="1.0" encoding="utf-8" ?>';
jellyScript += '<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">';
jellyScript += 'This is with escaping turned off >';
jellyScript += '</j:jelly>';

var runner = new GlideJellyRunner();
runner.setEscaping(false);
runner.setVariable('jvar_show', 'true');
var result = runner.run(jellyScript);
gs.print(result);

```

## Output

```
*** Script: This is with escaping turned off >
```

You can see the > was not escaped.

## Disabling Phase Two

It is possible to completely disable the second phase. This might be useful if you only need to run a single phase or if you are interested in seeing the output of particular Jelly script after the first phase.

`setTwoPhase(Boolean)` will turn the second phase on or off.

This example modifies the `runner_test` example to show the output after Phase One.

### UI Macro - runner\_test

```

<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
<j:if test="${jvar_show}">
    Hello World.
</j:if>
<j2:if test="${jvar_show}">
    Hello Again.
</j2:if>
</j:jelly>

```

## Background Script

```

var runner = new GlideJellyRunner();
runner.setTwoPhase(false);
runner.setVariable('jvar_show', 'true');
var result = runner.run(jellyScript);
gs.print(result);

```

## Output

```

*** Script:

Hello World.

<j2:if test="${jvar_show}">
    Hello Again.
</j2:if>

```

You can see that after the first phase the second phase Jelly tags are still in tact.

When forms and lists are cached the script that is actually cached is the output from Phase 1.

# Render Properties

Render Properties is an object that can be passed to Jelly that pages may refer to as a standard source for variables. Typically abbreviated `RP`, Render Properties are available in UI Pages as well as Display Business rules.

The class for Render Properties is exposed via a Package Call Replacement object `GlideRenderProperties`.

## Setting Render Properties

When running Jelly programmatically you can manually create and set some Render Properties with the `GlideRenderProperties` object.

### UI Macro - runner\_test

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
  <j:if test="${jvar_show}">
    Hello World.
  </j:if>
  <j2:if test="${jvar_show}">
    Hello Again.
  </j2:if>
</j:jelly>
```

### Background Script

```
var runner = new GlideJellyRunner();
var rp = new GlideRenderProperties();
runner.setRenderProperties(rp)
var result = runner.runMacro('runner_test');
gs.print(result);
```

## GlideRenderProperties

The Render Properties object can be used both before rendering and after rendering within `g:evaluate` blocks. Here is a list of the available functions that can be called in both circumstances.

### Get Methods

These methods get the string value of a property.

- `getParameterValue(String)` - Gets a GET or POST parameter passed to a page.

```
<g:evaluate>
var sys_id = RP.getParameterValue('sys_id')
</g:evaluate>
```

- `getEncodedQuery()`
- `getGaugeHeight()`
- `getGaugeId()`
- `getGaugeName()`
- `getGaugeType()`
- `getGaugeWidth()`

- getListControl()
- getMedia()
- getParameters
- getReferringURL()
- getViewID()
- getViewManager()
- getWindowID()
- getWindowProperties()

### **Additional Getter Methods**

These methods return a boolean dependent on specific properties being set.

- isChartDetailOnReport()
- isDialog()
- isHomePage()
- isInteractive() - Returns true if mode is not print and not preview
- isMaintainOrder
- isManyToMany()
- isMultipleUpdate()
- isOneToMany()
- isPopup()
- isPortal()
- isPreview()
- isPrint()
- isReadOnly()
- isRelatedList()
- isSearch()
- isSmallCaption()
- genEmptyForm() - Returns true if mode is search or multiple update

### **Set Methods**

These methods set values on the Render Properties object.

- setDialog(Boolean)
- setListControl(SysListControl)
- setMedia(String) - Can be `print`, `preview`, `portal` OR `read_only`
- setPrint(Boolean)
- setReadOnly(Boolean)
- setSmallCaption(Boolean)
- setView(String)
- setView(String, String)

### **Helper Functions**

Additional functions.

- `toString()` - Returns all properties as a String
- `useRelatedRecordManager(String type)`
- `useSlushbucket(String type)`

# Jelly Contexts

The Jelly context is an instance (object) of the `GlideJellyContext` class that contains all the methods and properties which are available when executing Jelly. The object includes:

- Escaped HTML entities as Constants
- A GlideController which contains all global variables
- Caches of templates
- GlideRecord References
- Functions for evaluating expressions within the context

The `GlideJellyContext` object is actually dependent on `GlideController` which is itself another context which includes the entire Rhino environment and global variables.

`GlideController` has a few available functions around setting and getting global variables.

## Get and Set Globals

`putGlobal(String, Object)` creates or updates a global variable in a context and `getGlobal(String)` gets the value for a given variable name.

### Background Script

```
var gc = new GlideController();

gc.putGlobal("foo", "bar"); gs.print(gc.getGlobal("foo"));
```

### Output

```
*** Script: bar
```

## Deleting Globals and Checking for Existence

`removeGlobal(String)` will remove a global variable and `exists(String)` will check for existence of a variable.

### Background Script

```
var gc = new GlideController();

gc.putGlobal("foo", "bar");
gs.print(gc.getGlobal("foo"));
gc.removeGlobal("foo");
gs.print(gc.exists("foo"));
```

### Output

```
*** Script: bar
*** Script: false
```

## Evaluating Scripts

The last function on a `GlideController` is one to evaluate scripts. `evaluateAsObject(String)` expects a JavaScript script to be passed to it.

This example creates a global then increments it and outputs it. Notice the script passed does the work then sets the result of the script by placing `foo;` at the end of the script. This essentially returns the result of the script and puts it in `fooResult`.

Remember you cannot just do `gc.evaluateAsObject(foo++)` because the context that the Background Script runs in and the `gc` contexts are intended to be different.

## Background Script

```
var gc = new GlideController();

gc.putGlobal("foo", 0);
var fooResult = gc.evaluateAsObject("foo++; foo;");
gs.print(fooResult);
```

## Output

```
*** Script: 1.0
```

Even though the execution seems to be continuous each object (`GlideJellyContext`, `GlideController`) is its own entity. The purpose of creating contexts is to bundle all information relative to the context so it can easily be passed to other objects. `GlideJellyContext` does not necessarily want to assume `GlideController` exists so it must be explicitly passed. Keeping each object specific and focused, and passing the contexts re-enforces two principles: cohesion and coupling. The idea that objects should only contain methods and properties relevant to itself and decreasing interdependency between objects.

## Creating a Jelly Context

Now that we have seen how to create and configure a `GlideController` to create a `GlideJellyContext` you would create a new instance of the Object and pass the `GlideController` as a parameter

```
var gc = new GlideController();
var gjc = new GlideJellyContext(gc);
```

# Evaluating Expressions

The same class that processes JEXL expressions within Jelly Scripts can be accessed programmatically and run. It is called `GlideEvaluator` and it requires recreating the same context it would have access to in a Jelly page, the Jelly Context. The Jelly Context itself depends on a `GlideController`.

This example recreates the Controller, JellyContext and Evaluator and evaluates a simple JEXL Expression.

## Background Script

```
var gc = new GlideController();
var gjc = new GlideJellyContext(gc);
var ge = new GlideEvaluator();
gjc.setVariable("jvar_test","test")

var myJexlExpression = new Packages.com.glide.ui.jelly.GlideExpressionJexl("jvar_test.equals('test')");
gs.print(myJexlExpression.evaluateAsBoolean(gjc));
```

*\*Output*

```
*** Script: true
```

## Walkthrough

Create a new `GlideController`.

```
var gc = new GlideController();
```

Create a new `GlideJellyContext` based on the new `GlideController` `gc`.

```
var gjc = new GlideJellyContext(gc);
```

Create a new `GlideEvaluator` to evaluate the JEXL expression.

```
var ge = new GlideEvaluator();
```

Set a variable in the context

```
gjc.setVariable("jvar_test","test")
```

Create a new evaluator with the expression `jvar_test.equals('test')`.

```
var myJexlExpression = new Packages.com.glide.ui.jelly.GlideExpressionJexl("jvar_test.equals('test')");
```

GlideExpressionJexl expects a JEXL expression as the parameter to the constructor, after that there is only one function that can be called, `evaluateAsBoolean`, which will evaluate and return the result.

Actually evaluate the expression and output the result

```
gs.print(myJexlExpression.evaluateAsBoolean(gjc));
```

This technique could be useful for debugging a particular expression but keep in mind, since it is dependent on a Package call it is not recommended to use this in production code as it is not known how long this would be available and if there will be a Package Call replacement

# Creating UI Extensions

The terminology is a little tricky here. I will use UI Macro to mean the actual UI Macro record, and `macro` or `g:macro` to refer to re-usable component created.

`g:macro` functions very similar to `g:function` in definition but functions more like `g:inline`. It should be used at the top of a UI Macro which you intend to re-use in other Pages and Macros. Attributes, like with `function`, listed will be like parameters for the macro. Any values specified will be used as default values, if the parameter is left blank when calling the macro. (This works unlike with `function`)

Even though a `macro` could act like a function, it is best to use it how it's intended which is to create re-usable UI elements.

One other distinction is that when using `macro` the name of the UI Macro itself must start with `ui_`. When using the macro in a UI Page or other UI Macro you can use the name as if it were a Jelly Tag.

## UI Macro - ui\_my\_function

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
  <g:macro message="Hello World" name="" />
  Message: ${jvar_message}<br/>
  Name: ${jvar_name}
</j:jelly>
```

## UI Page

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
  <g:ui_my_function message="Hi World" name="Cal Sosta" />
</j:jelly>
```

Notice the new tag `g:ui_my_function` this maps directly back to the name of the UI Macro.

Although it's not fundamentally different from using other methods a `macro` does work to re-inforce the idea of idiomatic and self-documenting code. That is we are using the structure of the language and its conventions to create clear, self-explanatory code. This same idea is re-used heavily in Frameworks like Angular JS and does result in clean code which is easier to maintain.

## Decorators

Decorators or Field Decorations are UI Macros that are rendered as part of a specific field on a form. These differ from Formatters because they are rendered inline with the element and are configured as an attribute for the field within the dictionary. Also, unlike Formatters which might be their own stand-alone functionality on a form, Decorators are meant to augment the field they are on.

To create and use your own Decorators you would start by creating a UI Macro within your system. There is no particular format for a Decorator but it is probably best to keep them small as to not disrupt the layout of the form.

Once you have created the Macro you may include the Decorator on your field by going to the Dictionary Entry for the particular field you want. If the field you are using already has decorations you may append the name of your macro to that list and if not you may add an attribute called `field_decorations` and set it to your macro.

In Fuji+ field attributes are specified in a Related List. The value should be exactly the same.



Within the context of the Decoration Macro you will have access to a Reference (Covered in Section 3.2) which refers to the current field.

Here is an example that will simply output the current referred field.

### UI Macro - my\_decorator

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
    <strong>${ref}</strong>
</j:jelly>
```

### Output

task.short\_description



You may also use the Phases of Jelly to execute methods on the reference.

In this example the first phase renders the code for the second phase.

### UI Macro - my\_decorator

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
    <strong>${[${ref}.getDisplayValue()]}</strong>
</j:jelly>
```

### Output

Short description My Short Description  

# Overriding Default Templates

The order in which Macros are evaluated will first check the UI Macro table the the file system for a file with the given name. It is possible to recreate templates as UI Macros and completely override the default behavior, however this would severely impact things like upgrades. There is a potential solution for this which is to create a UI Macro which can pass through to a template.

## Creating a Custom Data Type

This is an example overriding "item\_body\_cell\_default.xml", a system template that is used to render each cell in a list table. The intent is to create a new data type that will render differently in the list.

### Create A New Type

Please note changing or altering System Tables can have an adverse affect on the system. Only make changes in sub-production environments and test them thoroughly.

Navigate to the Field class table ( `sys_glide_object` ). Create a new Field class with the settings.

**Name:** testType

**Label:** Test Type

**Visible:** True

**Length:** 100

**Extends:** String

Add a field of the new Test Type to a table and add the field to the list view of that table. Be sure to enter some test data in a record.

### Create a UI Macro

In the UI Macros table create a new UI Macro called `item_body_cell_default`. When the system looks up the macro template it will first look here and find this macro and stop looking for the template.

#### UI Macro

```
<?xml version="1.0" encoding="utf-8"?>
<j:jelly trim="true" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
    <g:set jv="__ref__" expression="${ref}" />
    <j:set var="jvar_handle_type" value="${__ref__.getED().getInternalType()}" />

    <j:choose>
        <j:when test="${jvar_handle_type=='testType'}">
            <g:call function="set_ref_in_rhino_context.xml" />

            <j2:set var="jvar_target_text" value="[$__ref__.getHTMLValue()]" />
            <g2:evaluate var="jvar_target_text" jelly="true">
                var txt = jelly.jvar_target_text.toUpperCase()
                txt;
            </g2:evaluate>
            <td>${jvar_target_text}</td>
        </j:when>
    </j:choose>
</j:jelly>
```

```

<j:otherwise>
    <g:inline template="../ui.jttemplates/item_body_cell_default.xml"/>
</j:otherwise>
</j:choose>
</j:jelly>

```

## Output

Incident Number	Caller	Short description	Category	Priority	State	Assignment group	Assigned to
01		<j2:set var="jvar_insecure" value="\${'Uh oh!'} />	Network	1 - Critical	Active		Charlie Whitherspoon THIS IS A TEST

## Walkthrough

Declare the document type

```
<?xml version="1.0" encoding="utf-8"?>
```

Start Jelly, this Macro gets called once per each

```
<j:jelly trim="true" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
```

First we need to create a local variable that evaluates to the reference expression. If you just output  `${ref}`  it would be like: "incident.number", but when you evaluate that as an expression you actually get the reference to Incident Number.

`${ref.getED()}`  wouldn't work here because  `ref`  is just a String.

```
<g:setjs var="__ref__" expression="${ref}" />
```

Create a variable that is the type of the referenced field.

I am not just calling it  `jvar_type`  because there are lots of globals floating around here and I am trying not to accidentally use them.

```
<j:set var="jvar_handle_type" value="${__ref__.getED().getInternalType()}" />
```

Start a conditional block based on the value of  `jvar_handle_type`  .

```
<j:choose>
```

First condition type is our custom field type.

```
<j:when test="${jvar_handle_type=='testType'}">
```

This function will properly set the reference in all phases.

```
<g:call function="set_ref_in_rhino_context.xml" />
```

Using the second phase, set the variable `jvar_target_text` to the HTML display value of the field.

```
<j2:set var="jvar_target_text" value="${__ref__.getHTMLValue()}" />
```

Here is our customization for the field, always display it in upper case. Remembering to set the value.

```
<g2:evaluate var="jvar_target_text" jelly="true">
    var txt = jelly.jvar_target_text.toUpperCase()
    txt;
</g2:evaluate>
```

Actually output the value. We can also had custom CSS here to fix layout issues in the list.

```
<td>${jvar_target_text}</td>
```

End the condition.

```
</j:when>
```

Add the Default condition.

```
<j:otherwise>
```

Inline the original template. The path is necessary because otherwise the system would find the same UI Macro you are in. The relative path is necessary because it will search `ui.jtemplates` by default. So this goes up and back down to the same location. :)

This will need to match whichever template you are trying to override.

```
<g:inline template="../ui.jtemplates/item_body_cell_default.xml"/>
```

End the default condition.

```
</j:otherwise>
```

End the choose condition.

```
</j:choose>
```

End Jelly.

```
</j:jelly>
```



# System Properties

---

Some collected properties that may not be documented by default.

Property	Description	Type	Default
glide.jelly.expression_trace	Run the trace on certain Jelly Tags	Boolean	False
glide.ui.escape_text	Escape HTML in text as a Security setting	Boolean	True
glide.jelly.rhino_support	Support Rhino Script with JEXL	2:4	3:4
glide.ui.jelly.cacheable	Override to cache files not normally cached	String	textsearch.xml
glide.jelly.cache_compiled_ui_jtemplates	Setting to cache compiled (Phase 1) Templates	Boolean	true
glide.jelly.log_compile_times	Log the compile times for Jelly Pages	Boolean	false
glide.cache.size.syscache_jelly_ui_jtemplates	Max number of items to cache	Number	200
glide.debug.trace_threshold	Number of MS to wait before Warning		250

## Jelly Examples

---

This chapter will present several examples of full Jelly Pages with walkthroughs explaining the code.

## Core Concepts

---

This section looks at some examples trying to cover some of the core concepts essential to Jelly and show some working examples of simple algorithms using Jelly.

# Multiple Phases

Incorrect use of phases in Jelly can make things a nightmare, but correct usage can simplify many complex tasks. These examples will illustrate and hopefully clarify the concept of phases.

## Execution Order

First it is important to understand the order in which things are executed. Here is a simple example to show this concept.

### UI Page

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
<g:evaluate>
    var counter = 0.0;
</g:evaluate>

${++counter}
${++counter}
${++counter}
${++counter}
${++counter}
${++counter}

<br/>

    Complete
</j:jelly>
```

### Output After Phase 1

```
1.0
${++counter}
2.0
${++counter}
3.0
${++counter}

<br/>

    Complete
```

### Output After Phase 2

**1.0 4.0 2.0 5.0 3.0 6.0**

Complete

Without knowledge of phases you might think this output is incorrect. The numbers jump from **1.0** to **4.0** but within the Jelly Page we can see that the second **\${++counter}** is actually Phase 2.

### Walkthrough

Declare the document and start Jelly <?xml version="1.0" encoding="utf-8" ?>

*Phase 1 - Set a counter variable to **0.0**.*

```
<g:evaluate>
    var counter = 0.0;
</g:evaluate>
```

*Phase 1 - Increment and output counter. This is the first output.*

```
 ${++counter}
```

*Phase 2 - Increment and output counter. This is the *fourth* output. In the first phase the Jelly parse simply treats this as text and passes it through because it is not looking for JEXL expressions in `$[]`, it only evaluates `${}>` in the first phase.*

```
 ${[++counter]}
```

*Phase 1 - Second output.*

```
 ${++counter}
```

*Phase 2 - Fifth output.*

```
 ${[+counter]}
```

*Phase 1 - Third output.*

```
 ${++counter}
```

*Phase 2 - Sixth output.*

```
 ${[+counter]}
```

*Phase 1 - Output static content*

```
<br/>
Complete
</j:jelly>
```

The key takeaway here is that Jelly will execute and output everything in phase 1 (including static content) and then execute phase 2. Phase 2 elements such as tags in the `g2` or `j2` namespace or JEXL Expressions wrapped in `$[]` are simply ignored.

## Controlling Phase 2 From Phase 1

It is sometimes useful to control what phase 2 will actually execute from phase 1. Here is a simple example showing that.

### UI Page

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
<j:set var="jvar_toggle" value="true"/>

<j:if test="${jvar_toggle=='true'}">
    <j2:set var="jvar_message" value="Toggle was true and this is phase 2.">
        ${jvar_message}
    </j:if>

</j:jelly>
```

## Output After Phase 1

```
<j2:set var="jvar_message" value="Toggle was true and this is phase 2.">
${jvar_message}
```

## Output After Phase 2

Toggle was true and this is phase 2.

### Walkthrough

Declare the document and start Jelly

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
```

*Phase 1 - Set a variable `jvar_toggle` to true.*

```
<j:set var="jvar_toggle" value="true"/>
```

*Phase 1 - Check to see if `jvar_toggle` is true.*

```
<j:if test="${jvar_toggle=='true'}">
```

*Phase 2 - Set a variable `jvar_message`.*

```
<j2:set var="jvar_message" value="Toggle was true and this is phase 2.">
```

*Phase 2 - Output the message `${jvar_message}`*

*Phase 1 - End the `if`*

```
</j:if>
```

End Jelly

```
</j:jelly>
```

Even though the lines within the `if` are valid, Jelly because they are not phase 1 (`j` and `g` namespaces & `$[]`), they are simply passed through the first phase. When the Jelly Parser is run for the second time, it specifically executes `j2` and `g2` namespaces and `$[]` expressions. As far as the Jelly parser is concerned if a tag or expression is not in its phase, it is text.

## Combining Phase 1 and Phase 2

Here is an example using output from Phase 1 combined with Phase 2 to create a list of fields and their value for a particular record. Phase 1 will execute and that output will combine with Phase 2 to create JEXL expressions which will render the result we want.

### UI Page

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">

    <g:evaluate>
        var incidentRec = new GlideRecord('incident');
        var fields = [];
        incidentRec.setLimit(1);
        incidentRec.query();
        incidentRec.next();

        for(field in incidentRec){
            fields.push(field);
        }

    </g:evaluate>

    <j:forEach items="${fields}" var="jvar_field" varStatus="jvar_status">
        <ul>
            <li><strong>${jvar_field}</strong> - ${incidentRec.${jvar_field}.getDisplayValue()}</li>
        </ul>
    </j:forEach>
</j:jelly>
```

### Output After Phase 1

```
<ul>
    <li><strong>closed_by</strong>-${incidentRec.closed_by.getDisplayValue()}</li>
    <li><strong>assigned_to</strong>-${incidentRec.assigned_to.getDisplayValue()}</li>
    .
    .
    <li><strong>work_notes_list</strong>-${incidentRec.work_notes_list.getDisplayValue()}</li>
</ul>
```

### Output After Phase 2

```
<ul>
    <li><strong>closed_by</strong>-John User</li>
    <li><strong>assigned_to</strong>-Jane User</li>
    .
    .
    <li><strong>work_notes_list</strong>-</li>
</ul>
```

### Walkthrough

Declare the document and start Jelly

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
```

Establish a GlideRecord to use

```
<g:evaluate>
    var incidentRec = new GlideRecord('incident');
    var fields = [];
    incidentRec.setLimit(1);
    incidentRec.query();
    incidentRec.next();
```

Capture fields for the record in a separate array so we can loop through them

```
for(field in incidentRec){
    fields.push(field);
}

</g:evaluate>
```

Loop through each item in the list assigning it to `jvar_field`

```
<j:forEach items="${fields}" var="jvar_field" varStatus="jvar_status">
```

Start the list

```
<ul>
```

Create a list item

```
<li>
```

Create a bold label using the field name

```
<strong>${jvar_field}</strong> -
```

*Phase 1 - Output the field name wrapped in a Phase 2 JEXL expression*

```
 ${[incidentRec.${jvar_field}.getDisplayValue()]}</li>
```

*Phase 2 - Output the display value of the field*

```
 ${[incidentRec.FIELD.getDisplayValue()]}
```

End the list item, list and loop

```

</li>
</ul>
</j:forEach>
```

End Jelly

```
</j:jelly>
```

## Using Phases to Group Results

Here is an example that combines multiple phases to group a record set easily.

### UI Page

```

<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">

    <g:evaluate>
        var currentCategory = "";
        var incidentRec = new GlideRecord('incident');
        incidentRec.orderBy('category');
        incidentRec.query();
    </g:evaluate>

    <j:while test="${incidentRec.next()}">
        <j:if test="${incidentRec.category!=currentCategory}">
            <g:evaluate>
                currentCategory = incidentRec.category.toString();
            </g:evaluate>
            <h3>${incidentRec.category.getDisplayValue()}</h3>
            <ul>
                <g2:evaluate>
                    incidentRec.setLocation(-1);
                </g2:evaluate>
                <j2:while test="${incidentRec.next()}">
                    <j2:if test="${incidentRec.category.toString()=='${incidentRec.category.toString()}'}">
                        <li>${incidentRec.number} - ${incidentRec.short_description}</li>
                    </j2:if>
                </j2:while>
            </ul>
        </j:if>
    </j:while>

</j:jelly>
```

### Walkthrough

Since this example is very close to the last I won't go through line by line but the idea is the same. The first phase generates the second phase. The same variable `incidentRec` is kept throughout and just "rewound" each time it needs to be used, rather than making an additional query each time.

Note the complex `test` attribute in the `j2:if` remember the second phase will be comparing a variable to the output of Phase 1, which is a string, wrapping the value in quotes will be necessary. Since double quotes are used for the attribute single quotes should be used inside the string.

An alternative would be to use a single phase to output everything, this can be done by creating a second loop in the first phase within the `ul`. An additional check can be made to see if the category changes then exiting the inner loop

and rolling back 1 record. This way is conceptually much more difficult than using both phases to our advantage.

# Fizz Buzz

Fizz Buzz is a popular interview type question which has a very simple algorithm. Iterate over a given range of numbers and output Fizz if the number is divisible by 3, Buzz if the number is divisible by 5 and FizzBuzz if the number is divisible by both 3 and 5. There are many variations, for this example we will follow the core algorithm and output the number itself if it is not divisible by 3 or 5.

## UI Page

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">

    <j:forEach begin="1" end="100" var="jvar_number">

        <j:choose>
            <j:when test="${jvar_number % 3==0 and jvar_number % 5==0}">
                FizzBuzz<br/>
            </j:when>
            <j:when test="${jvar_number % 3==0}">
                Fizz<br/>
            </j:when>
            <j:when test="${jvar_number % 5==0}">
                Buzz<br/>
            </j:when>
            <j:otherwise>
                ${jvar_number}<br/>
            </j:otherwise>
        </j:choose>

    </j:forEach>

</j:jelly>
```

## Walkthrough

Start the loop from 1 to 100 storing the current variable in `jvar_number`

```
<j:forEach begin="1" end="100" var="jvar_number">
```

Normally you could just use `if` but in Jelly there are no `else` conditions so we must use a `choose` OR `switch`

```
<j:choose>
```

The first condition checks that `jvar_number` is evenly divisible by 3 and 5. Notice the use of `and`. `&&` won't work as it will invalidate the XML and  `${AND}` won't help either since we are not outputting this and it would end up being invalid anyways (for instance trying to use it in Phase 2).

```
<j:when test="${jvar_number % 3==0 and jvar_number % 5==0}">
```

Output our first condition.

```
    FizzBuzz<br/>
</j:when>
```

Repeat the same check for 3 and 5 individually and output the correct term.

```
<j:when test="${jvar_number % 3==0}">
    Fizz<br/>
</j:when>
<j:when test="${jvar_number % 5==0}">
    Buzz<br/>
</j:when>
```

If no other condition is met just output the number.

```
<j:otherwise>
    ${jvar_number}<br/>
</j:otherwise>
```

End the `choose` and the loop.

```
</j:choose>

</j:forEach>
```

# Fibonacci Numbers

The Fibonacci Numbers are a famous sequence of numbers where the next number in the sequence is the sum of the last two number. It is a great algorithm to illustrate looping in programming languages.

This is an example of that algorithm written using Jelly and JavaScript.

## UI Page

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
<j:set var="jvar_max_fib" value="20" />
<g:evaluate object="true" var="jvar_number">
    [1,1]
</g:evaluate>

1<br/>
1<br/>
<j:while test="${jvar_number.length!=jvar_max_fib}">
    <g:evaluate jelly="true" var="jvar_next_fib">
        var next_fib = jelly.jvar_number[jelly.jvar_number.length - 1 ] + jelly.jvar_number[jelly.jvar_number.length - 2];
        jelly.jvar_number.push(next_fib)
        next_fib;
    </g:evaluate>

    ${jvar_next_fib}<br/>
</j:while>

    Complete
</j:jelly>
```

## Output

```

1
1
2
3
5
8
13
21
34
55
89
144
233
377
610
987
1597
2584
4181
6765
Complete

```

## Walkthrough

Declare the document and start Jelly

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
```

Set the maximum of numbers to add to the sequence (at least 2).

```
<j:set var="jvar_max_fib" value="20" />
```

Initialize the array to hold the numbers in the sequence setting the first two numbers storing them in the variable `jvar_number` and it is an object, so the `object` attribute must be set to true.

```
<g:evaluate object="true" var="jvar_number">
[1,1]
</g:evaluate>
```

In JavaScript terms a variable is either a base type (string, bool, number) or it is an object, which could be a true object or an array.

Output the first two numbers.

```
1<br/>
1<br/>
```

Check to see if we have reached the maximum number.

This is actually a pretty unsafe way to check. A better way would be `>=` but it causes an error since you may not have `<` or `>` in the `test` attribute. If error checking was absolutely a concern you could use an `evaluate` tag to do the check and store the result.

```
<j:while test="${jvar_number.length!=jvar_max_fib}">
```

Generate the next number in the sequence and add it to the array.

```
<g:evaluate jelly="true" var="jvar_next_fib">
    var next_fib = jelly.jvar_number[jelly.jvar_number.length - 1] + jelly.jvar_number[jelly.jvar_number.length - 2];
    jelly.jvar_number.push(next_fib)
    next_fib;
</g:evaluate>
```

Output the last number in the sequence.

```
${jvar_next_fib}<br/>
```

End the loop.

```
</j:while>
```

Outputting complete at the end lets us know the page compiled without any errors, so if the page displays "Complete" we at least know there are no syntax errors in the code.

```
Complete
```

End Jelly and the page

```
</j:jelly>
```

## Suggested Exercises

1. Fix the page to properly check the length against the max number in the while loop.
2. Modify the code to stop when the Fibonacci number exceeds a certain amount instead of having a maximum number of the sequence.

# Bubble Sort

The Bubble Sort is one of the simplest (and slowest) sorting algorithms. It continually iterates over an array comparing two elements and switching them if they are out of order. It is another great algorithm to show looping capabilities.

This is an example written purely in Jelly (and without the use `evaluate` tags).

## UI Page

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
<j:useList var="jvar_values" items="${[10,4,299,87,18,72]}" />
Before: ${jvar_values}<br/>
<j:set var="jvar_swapped" value="true" />
<j:while test="${jvar_swapped=='true'}">
    <j:set var="jvar_swapped" value="false" />

    <j:forEach begin="0" end="${jvar_values.size()-1}" step="1" var="jvar_index">
        <j:if test="${jvar_values[jvar_index] gt jvar_values[jvar_index+1]}">
            <j:set var="jvar_swapped" value="true" />
            <j:set var="jvar_temp" value="${jvar_values[jvar_index]}" />

            <j:invoke method="set" on="${jvar_values}">
                <j:arg value="${jvar_index}" />
                <j:arg value="${jvar_values[jvar_index+1]}" />
            </j:invoke>

            <j:set var="jvar_index_next" value="${jvar_index+1}" />
            <j:invoke method="set" on="${jvar_values}">
                <j:arg value="${jvar_index_next}" />
                <j:arg value="${jvar_temp}" />
            </j:invoke>
        </j:if>
    </j:forEach>
</j:while>
After: ${jvar_values}<br/>

Complete
</j:jelly>
```

## Output

Before: [10, 4, 299, 87, 18, 72]

After: [4, 10, 18, 72, 87, 299]

Complete

## Walkthrough

Declare the document and start Jelly <?xml version="1.0" encoding="utf-8" ?>

Create a list with some numeric values.

```
<j:useList var="jvar_values" items="${[10,4,299,87,18,72]}" />
```

Output the "before" list.

```
Before: ${jvar_values}<br/>
```

Set our flag to `true` so we can start the first iteration.

```
<j:set var="jvar_swapped" value="true" />
```

Continue the loop while the `jvar_swapped` flag is `true`.

```
<j:while test="${jvar_swapped=='true'}">
```

Once in the loop set the flag to false, unless a swap happens we will exit the loop.

```
<j:set var="jvar_swapped" value="false" />
```

Loop from 0 to the last index of the string (the length - 1) and store that iterator in `jvar_index`.

```
<j:forEach begin="0" end="${jvar_values.size()-1}" step="1" var="jvar_index">
```

Compare the current element to the next element in the list. Notice the use of `gt` as the operator.

```
<j:if test="${jvar_values[jvar_index] gt jvar_values[jvar_index+1]}>
```

Set the swapped flag to true.

```
<j:set var="jvar_swapped" value="true" />
```

Set a temporary variable to the current element.

```
<j:set var="jvar_temp" value="${jvar_values[jvar_index]}" />
```

Set the current element to the next element. Remember `invoke` allows us to execute a method, in this case we are using `set` which allows us to set a variable.

Alternatively it is possible to do this whole example using `evaluate` tags and then we could use the JavaScript bracket notation to do the same type of operation.

```
<j:invoke method="set" on="${jvar_values}">
```

Set the arguments for the method.

```
<j:arg value="${jvar_index}" />
<j:arg value="${jvar_values[jvar_index+1]}" />
```

```
</j:invoke>
```

This is a workaround because `j:arg` would not allow `value="${jvar_index+1}"`.

```
<j:set var="jvar_index_next" value="${jvar_index+1}" />
```

Set the next element to the temporary value.

```
<j:invoke method="set" on="${jvar_values}">
  <j:arg value="${jvar_index_next}" />
  <j:arg value="${jvar_temp}" />
</j:invoke>
</j:if>
```

End the for loop.

```
</j:forEach>
```

End the while loop.

```
</j:while>
```

Output the new list.

```
After: ${jvar_values}<br/>
```

Output complete and end Jelly.

```
Complete
</j:jelly>
```

# Calling Functions

Here is an example of creating and using a UI Macro as a function. Let's start with a simple function to sum two numbers.

## UI Macro - sum\_numbers

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">

    <g:function first="REQUIRED" second="REQUIRED" />
    Summing: ${jvar_first} + ${jvar_second}<br/>
    <j:set var="jvar_result" value="${jvar_first + jvar_second}" />

</j:jelly>
```

## UI Page

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">

    <g:call function="sum_numbers" first="10" second="5" />
    Result: ${jvar_result}

    <hr/>

    <g:inline template="sum_numbers" first="33" second="11" />
    Result: ${jvar_result}

    <hr/>

</j:jelly>
```

## Walkthrough

Declare the document and start Jelly

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
```

*Call* the function passing in parameters

```
<g:call function="sum_numbers" first="10" second="5" />
```

## Within function

Tell Jelly to expect two parameters

```
<g:function first="REQUIRED" second="REQUIRED" />
```

Output the variables back to the user to confirm

```
Summing: ${jvar_first} + ${jvar_second}<br/>
```

Sum the two variables and store the result in `jvar_result`

```
<j:set var="jvar_result" value="${jvar_first + jvar_second}" />
```

### Outside function

Once we leave the `call` the scope is no longer available. Therefore this will be blank.

```
Result: ${jvar_result}
<hr/>
```

Inline the template this time instead of just calling it.

```
<g:inline template="sum_numbers" first="33" second="11" />
```

### Within the included function

Tell Jelly to expect two parameters.

```
<g:function first="REQUIRED" second="REQUIRED" />
```

Output the variables back to the user to confirm...again.

```
Summing: ${jvar_first} + ${jvar_second}<br/>
```

Sum the two variables and store the result in `jvar_result` ...again.

```
<j:set var="jvar_result" value="${jvar_first + jvar_second}" />
```

### Outside function

This time we do have the result, because the function wasn't just called but the entire contents of the file was inlined, `jvar_result` is now in the outer scope.

```
Result: ${jvar_result}
```

### End the page and Jelly

```
<hr/>
</j:jelly>
```

Depending on how you want to use your function you might find one of these cases useful. If you need access to the inner scope you will need to do an inline, otherwise a simple call might be all you need.

# Towers of Hanoi

Towers of Hanoi is another classic puzzle which is often used to demonstrate recursion in programming. The puzzle consists of 3 rods which have a number of different sized discs. Initially all discs are stacked on the first rod in order by size with the largest on the bottom.

The goal of the game is to move all discs from the first to the third rod but you may not place a larger disc on top of a smaller disc.

Here is a diagram showing the first few steps of the solution.



## UI Macro - hanoi

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">

    <g:function disc="REQUIRED" from="REQUIRED" by="REQUIRED" to="REQUIRED" />
    <j:if test="${jvar_disc > 0}">
        <g:call function="hanoi" disc="${jvar_disc - 1}" from="${jvar_from}" by="${jvar_to}" to="${jvar_by}" />
        Moving ${jvar_disc} from ${jvar_from} to ${jvar_to}<br/>
        <g:call function="hanoi" disc="${jvar_disc - 1}" from="${jvar_by}" by="${jvar_from}" to="${jvar_to}" />
    </j:if>
</j:jelly>
```

## UI Page

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">

    <g:call function="hanoi" disc="3" from="A" by="B" to="C" />
</j:jelly>
```

## Walkthrough

There is no simple way to walk through this code, so at a high level we start by calling the Hanoi function.

Within the function we test that `disc` is not 0, meaning we are at the bottom of a stack. If it is, we just end the recursion and return, if it is not we call Hanoi again, decrementing the disc and inverting the `to` and `by` rods.

Once we get to the bottom of the stack we start outputting the moves, and moving back up the chain of recursion, after outputting each move, we descend again this time we move from the `by` to the `to` by the `from`.

This is a great example of the power of Jelly and how it can truly be used as a programming language.

## Static Page

---

This section covers Static Page examples, meaning the underlying structure of the page will not change. Field values or list content may change but the layout and execution of the page will always be the same.

# Hello World

The simplest possible page, the "Hello World".

## UI Page

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
    <h1>Hello World!</h1>
</j:jelly>
```

## Output

# Hello World!

## Walkthrough

Declare the Document Type

```
<?xml version="1.0" encoding="utf-8" ?>
```

Start Jelly

```
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
```

Output a Header

```
    <h1>Hello World!</h1>
```

End Jelly

```
  </j:jelly>
```

It is important to note that even though this UI Page only outputs "Hello World" if you examine the source all the standard components to a ServiceNow page get added. For instance the Output Messages panel, Page Timer and the standard Glide API Objects.

## Suggested Exercises

1. Modify the code to also output the current users name.
2. Add a conditional block to show an additional greeting if the current user is an Administrator.

# Simple Calculator

This example walks through a simple 4 function calculator. A simple form takes in two numbers and an operator. Using an `evaluate` tag the result is calculated and returned to the user.

## UI Page

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">

    <g:evaluate>
        var operator = RP.getParameterValue('operator') || '';
        var result = '';

        if(operator !== ''){
            var firstNumber = parseFloat(RP.getParameterValue('first_number')) || 0;
            var secondNumber = parseFloat(RP.getParameterValue('second_number')) || 0;

            if( operator == 'add'){
                result = firstNumber + secondNumber;
            } else if( operator == 'sub' ) {
                result = firstNumber - secondNumber;
            } else if( operator == 'mult' ) {
                result = firstNumber * secondNumber;
            } else if( operator == 'div' ) {
                result = firstNumber / secondNumber;
            }
        }
    </g:evaluate>

    <h2>Jelly Calculator</h2>
    <hr/>

    <form>
        <label>First Number</label>
        <input name="first_number" />

        <label>Operator</label>
        <select name="operator">
            <option value="add">+</option>
            <option value="sub">-</option>
            <option value="mult">*</option>
            <option value="div">/</option>
        </select>

        <label>Second Number</label>
        <input name="second_number" />
        <input type="submit" />

    </form>
    <hr/>

    <j:if test="${result!=''}" >
        Result ${result}
    </j:if>
</j:jelly>
```

## Output

## Jelly Calculator

First Number  Operator  Second Number

Result 3.0

### Walkthrough

Declare the Document Type

```
<?xml version="1.0" encoding="utf-8" ?>
```

Start Jelly

```
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
```

Begin the processing to check for parameters indicating values were submitted then do the calculation

```
<g:evaluate>
```

Check for a parameter named `operator` (part of the form) if it is present then the form was submitted as opposed to the first load.

```
var operator = RP.getParameterValue('operator') || '';
```

Initialize the response variable setting it to `".`. This is important because later in the page we will check for this default value to determine if the result is shown.

```
var result = '';
```

Check for `operator` indicating that we should do a calculation.

```
if(operator !== ''){
```

Check for the number parameters.

```
var firstNumber = parseFloat(RP.getParameterValue('first_number')) || 0;
var secondNumber = parseFloat(RP.getParameterValue('second_number')) || 0;
```

Do the calculation depending on the operator.

```
if( operator == 'add'){
    result = firstNumber + secondNumber;
} else if( operator == 'sub' ) {
```

```

        result = firstNumber - secondNumber;
    } else if( operator == 'mult') {
        result = firstNumber * secondNumber;
    } else if( operator == 'div') {
        result = firstNumber / secondNumber;
    }
}

```

We could just do an `eval` here but this is a poor way to do it. For one it is a security issue, code could be injected. Secondly, we would need to convert the operators between a string value and +,-,\*,/ since passing those values as parameters will render their URL encoded equivalents.

End the evaluate

```
</g:evaluate>
```

Start the main page output with a header and a separator

```

<h2>Jelly Calculator</h2>
<hr/>

```

Begin the form to enclose the inputs we want to submit. Leaving off the attributes will result in the default settings: submitting to the same page URL and as GET (passing parameters in the URL)

```
<form>
```

First value

```

<label>First Number</label>
<input name="first_number" />

```

Operator choice list with option values set to strings

```

<label>Operator</label>
<select name="operator">
    <option value="add">+</option>
    <option value="sub">-</option>
    <option value="mult">*</option>
    <option value="div">/</option>
</select>

```

Second value

```

<label>Second Number</label>
<input name="second_number" />

```

Submit button

```
<input type="submit" />
```

End form and separator

```
</form>
<hr/>
```

Check if `result` is the default value, if not output the result.

```
<j:if test="${result!=''}" >
    Result ${result}
</j:if>
```

End Jelly

```
</j:jelly>
```

## Suggested Exercises

1. Add an error if `first_number` or `second_number` is blank.
2. Add error handling to catch a divide by 0 error.
3. Add additional `third_number` and second `operator` values.

# Output A List

This example will output a list for a given table.

## UI Page

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
<g:evaluate>
    var table = RP.getParameterValue('table') || "incident";
    var query = RP.getParameterValue('query') || "";

    var tableRec = new GlideRecord(table)
    if(query != ''){
        tableRec.addQuery(query);
    }

    tableRec.query();
</g:evaluate>

<ul>
    <j:while test="${tableRec.next()}">
        <li>${tableRec.getDisplayValue()} - Created On ${tableRec.sys_created_on.getDisplayValue()}</li>
    </j:while>
</ul>
</j:jelly>
```

## Output

- INC0000001 - Created On 2012-12-07 10:24:13
- INC0000002 - Created On 2012-12-07 14:30:06
- INC0000003 - Created On 2012-12-22 06:41:46
- INC0000004 - Created On 2012-12-07 14:34:12
- INC0000005 - Created On 2012-12-22 07:14:01
- INC0000006 - Created On 2012-12-04 12:42:01
- INC0000007 - Created On 2012-12-04 12:43:10
- INC0000008 - Created On 2012-12-04 12:43:58
- INC0000009 - Created On 2014-05-21 15:51:33
- INC0000010 - Created On 2014-05-21 15:54:41
- INC0000011 - Created On 2014-05-21 16:02:15
- INC0000012 - Created On 2014-05-21 16:11:21
- INC0000013 - Created On 2014-05-21 16:18:07
- INC0000014 - Created On 2014-05-21 16:38:40
- INC0000015 - Created On 2014-05-21 16:40:15
- INC0000016 - Created On 2014-05-21 16:41:15
- INC0000017 - Created On 2014-05-21 16:41:54
- INC0000018 - Created On 2014-05-21 16:42:56
- INC0000019 - Created On 2014-05-21 16:45:51
- INC0000020 - Created On 2014-05-21 16:51:54
- INC0000021 - Created On 2014-05-21 16:52:18
- INC0000024 - Created On 2014-05-21 16:53:13
- INC0000025 - Created On 2014-05-21 16:55:07
- INC0000026 - Created On 2014-05-21 16:56:03
- INC0000027 - Created On 2014-05-21 16:59:52
- INC0000028 - Created On 2014-05-21 16:58:44
- INC0000029 - Created On 2014-05-21 17:01:31
- INC0000030 - Created On 2014-05-21 17:02:01

## Walkthrough

Declare the Document Type

```
<?xml version="1.0" encoding="utf-8" ?>
```

## Start Jelly

```
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
```

Do some initial processing getting request parameters and initializing a Glide Record for the given table. This `evaluate` could be replaced with a Processing Script but to emphasize the usage in Jelly I am doing it here.

```
<g:evaluate>
```

Get the `table` and `query` parameters.

```
var table = RP.getParameterValue('table') || "incident";
var query = RP.getParameterValue('query') || "";
```

Initialize the GlideRecord for `table`.

```
var tableRec = new GlideRecord(table)
```

Only add a query if it is defined

```
if(query != ''){
    tableRec.addQuery(query);
}
```

Execute the Glide Record

```
tableRec.query();
```

End the evaluate

```
</g:evaluate>
```

Begin an unordered list in the main body of the page

```
<ul>
```

Loop through the records in `tableRec`

```
<j:while test="${tableRec.next()}">
```

Output the display field for the record and it's created date

```
<li>${tableRec.getDisplayValue()} - Created On ${tableRec.sys_created_on.getDisplayValue()}</li>
```

End the loop

```
</j:while>
```

End the list

```
</ul>
```

End Jelly

```
</j:jelly>
```

## Suggested Exercises

1. Add Short Description and Priority to the list of fields.
2. Show every other field as bold
3. Add a form with an input to set a query. *Hint: Part of the code is there already*
4. Add a field to the form to specify the `table`. *Hint: There are specific functions in the Glide Tag Library to help with this.*

# Simple Form

This example will output a form with any submitted values being carried over into the form fields. The `evaluate` makes use of a helper function to simplify the task of getting the values and specifying them on an object. This will result in cleaner code with less global variables floating around.

## UI Page

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">

<g:evaluate>
    var formObj = {};
    getFormValue('name', '');
    getFormValue('age', 18);
    getFormValue('email', '');
    getFormValue('phone', '');

    function getFormValue(name, defaultValue){
        formObj[name] = RP.getParameterValue(name) || defaultValue;
    }
</g:evaluate>

<form>
    <input name="name" value="${formObj.name}" placeholder="Name" />
    <input name="age" value="${formObj.age}" placeholder="Age" />
    <input name="email" value="${formObj.email}" placeholder="Email" />
    <input name="phone" value="${formObj.phone}" placeholder="Phone" />
    <input type="submit" />
</form>

</j:jelly>
```

## Output

Johnny Test	21	johnny.test@jellyiscool.com	123-123-1234	Submit
-------------	----	-----------------------------	--------------	--------

## Walkthrough

Declare the Document Type

```
<?xml version="1.0" encoding="utf-8" ?>
```

Start Jelly

```
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
```

Begin processing the page parameters, since no var is specified the intention is to use the variables within the script on the page.

```
<g:evaluate>
```

Create a generic form object to hold all the values instead of multiple global variables.

```
var formObj = {};
```

Use the helper function to get and set the page parameters to script variables.

```
getFormValue('name', '');
getFormValue('age', 18);
getFormValue('email', '');
getFormValue('phone', ''');
```

Helper function to get parameters and set them to the form object property.

```
function getFormValue(name, defaultValue){
```

Using bracket notation, check for the parameter and set the property to that OR the default value if the property is not present.

```
    formObj[name] = RP.getParameterValue(name) || defaultValue;
}
```

End the evaluate.

```
</g:evaluate>
```

Begin the form to enclose the inputs we want to submit. Leaving off the attributes will result in the default settings: submitting to the same page URL and as GET (passing parameters in the URL)

```
<form>
```

Specify the inputs, the value is an expression that will evaluate to a property on `formObj`. Add a place holder to indicate the field name.

```
<input name="name" value="${formObj.name}" placeholder="Name" />
<input name="age" value="${formObj.age}" placeholder="Age" />
<input name="email" value="${formObj.email}" placeholder="Email" />
<input name="phone" value="${formObj.phone}" placeholder="Phone" />
```

Submit button for the form.

```
<input type="submit" />
```

End the form.

```
</form>
```

End Jelly

```
</j:jelly>
```

## Suggested Exercises

1. Replace one of the fields with a Reference Field
2. Modify the page to add simple error checking

# Form Using Macros

This example modifies the Simple Form example to use a UI Macro to generalize the Form a bit. I will go through two versions of this example to show how to refactor the code to make use of as much Jelly as possible.

## UI Macro - jelly\_form\_field

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
    <input name="${jvar_name}" value="${jvar_value}" placeholder="${jvar_placeholder}" />
</j:jelly>
```

## Walkthrough

Declare the Document Type

```
<?xml version="1.0" encoding="utf-8" ?>
```

Start Jelly

```
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
```

Add an `input` tag using JEXL expressions for the values of each attribute.

```
<input name="${jvar_name}" value="${jvar_value}" placeholder="${jvar_placeholder}" />
```

End Jelly

```
</j:jelly>
```

## UI Page

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">

    <g:evaluate>
        var formObj = {};
        getFormValue('name', '');
        getFormValue('age', 18);
        getFormValue('email', '');
        getFormValue('phone', '');

        function getFormValue(name, defaultValue){
            formObj[name] = RP.getParameterValue(name) || defaultValue;
        }
    </g:evaluate>

    <form>
        <g:call function="jelly_form_field" name="name" value="${formObj.name}" placeholder="Name" />
        <g:call function="jelly_form_field" name="age" value="${formObj.age}" placeholder="Age" />
        <g:call function="jelly_form_field" name="email" value="${formObj.email}" placeholder="Email" />
        <g:call function="jelly_form_field" name="phone" value="${formObj.phone}" placeholder="Phone" />
    </form>
</j:jelly>
```

```

<input type="submit" />
</form>
</j:jelly>

```

## Output

Name	18	Email	phone	Submit
------	----	-------	-------	--------

## Walkthrough

Declare the Document Type

```
<?xml version="1.0" encoding="utf-8" ?>
```

Start Jelly

```
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
```

Begin processing the page parameters, since no var is specified the intention is to use the variables within the script on the page.

```
<g:evaluate>
```

Create a generic form object to hold all the values instead of multiple global variables.

```
var formObj = {};
```

Use the helper function to get and set the page parameters to script variables.

```

getFormValue('name', '');
getFormValue('age', 18);
getFormValue('email', '');
getFormValue('phone', '');

```

Helper function to get parameters and set them to the form object property.

```
function getFormValue(name, defaultValue){
```

Using bracket notation, check for the parameter and set the property to that OR the default value if the property is not present.

```

    formObj[name] = RP.getParameterValue(name) || defaultValue;
}
```

End the evaluate.

```
</g:evaluate>
```

Begin the form to enclose the inputs we want to submit. Leaving off the attributes will result in the default settings: submitting to the same page URL and as GET (passing parameters in the URL)

```
<form>
```

Call the UI Macro passing in the values for each attribute. The advantage here is that if the input ever needs to change, instead of updating each line, you now only need to update the macro once.

```
<g:call function="jelly_form_field" name="name" value="${formObj.name}" placeholder="Name" />
<g:call function="jelly_form_field" name="age"    value="${formObj.age}" placeholder="Age" />
<g:call function="jelly_form_field" name="email"  value="${formObj.email}" placeholder="Email" />
<g:call function="jelly_form_field" name="phone"  value="${formObj.phone}" placeholder="phone" />
```

Submit button for the form.

```
<input type="submit" />
```

End the form.

```
</form>
```

End Jelly

```
</j:jelly>
```

## Suggested Exercises

1. Modify the UI Macro to add a Label or Help field

## Dynamic Page

---

Dynamic pages are those that have multiple purposes or layouts built in. The configuration of the page may change depending on its parameters. In this section we will revisit the examples from the last section and expand the functionality of each.

# Dynamic Calculator

This example will expand the previous example to have different "screens". The page remains the same but the output will be different depending on the action.

For this example we will have three "actions" or layouts for the page. The basic layout is no "action" and this will be the standard form as before. The second layout will be triggered when submitting the form, the action will be "calculate" and this layout will show the result and a back button.

The final layout is an error layout, to catch any potential accidental navigations away from the basic actions.

```

<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
    <g:evaluate>
        var action = RP.getParameterValue('action') || '';
    </g:evaluate>

    <h1>Jelly Calculator</h1>

    <j:choose>
        <j:when test="${action==''}">
            <h3>Welcome to Math!</h3>
            <form>
                <label>First Number</label>
                <input name="first_number" />

                <label>Operator</label>
                <select name="operator">
                    <option value="add">+</option>
                    <option value="sub">-</option>
                    <option value="mult">*</option>
                    <option value="div">/</option>
                </select>

                <label>Second Number</label>
                <input name="second_number" />
                <input type="hidden" value="calculate" name="action" />
                <input type="submit" />

            </form>
        </j:when>

        <j:when test="${action=='calculate'}">
            <g:evaluate>
                var operator = RP.getParameterValue('operator') || '';
                var result = '';

                if(operator !== ''){
                    var firstNumber = parseFloat(RP.getParameterValue('first_number')) || 0;
                    var secondNumber = parseFloat(RP.getParameterValue('second_number')) || 0;

                    if( operator == 'add'){
                        result = firstNumber + secondNumber;
                    } else if( operator == 'sub' ) {
                        result = firstNumber - secondNumber;
                    } else if( operator == 'mult' ) {
                        result = firstNumber * secondNumber;
                    } else if( operator == 'div' ) {
                        result = firstNumber / secondNumber;
                    }
                }
            </g:evaluate>

            <j:if test="${result!=''}" >
                Result ${result}
                <hr/>
                <a href="dynamic_calculator.do">Do more math!</a>
            </j:if>
        </j:when>
    </j:choose>
</j:jelly>

```

```
</j:when>  
  
<j:otherwise>  
    An Error Has Occured  
</j:otherwise>  
</j:choose>  
</j:jelly>
```

Output - Basic Layout

## Jelly Calculator

Welcome to Math!

First Number  Operator  Second Number

Output - Result Layout

## Jelly Calculator

Result 3.0

---

Do more math!

Output - Error Layout

## Jelly Calculator

An Error Has Occured

Walkthrough

Declare the Document Type <?xml version="1.0" encoding="utf-8" ?>

Start Jelly

Begin the processing to check for the action parameter which will determine the layout of the page

```
<g:evaluate>  
    var action = RP.getParameterValue('action') || '';  
</g:evaluate>
```

Output the page header

```
<h1>Jelly Calculator</h1>
```

Start a Jelly `choose` block to create conditions for each possible page action

```
<j:choose>
```

The first condition will check for a blank action which is the default.

```
<j:when test="${action==' '}>
```

Output the calculator form as we did before

```
<h3>Welcome to Math!</h3>
<form>
    <label>First Number</label>
    <input name="first_number" />

    <label>Operator</label>
    <select name="operator">
        <option value="add">+</option>
        <option value="sub">-</option>
        <option value="mult">*</option>
        <option value="div">/</option>
    </select>

    <label>Second Number</label>
    <input name="second_number" />
```

Add in a "hidden" input type, this will be passed as a parameter and will trigger the result layout when the form is submitted.

```
<input type="hidden" value="calculate" name="action" />

<input type="submit" />

</form>
```

End the condition.

```
</j:when>
```

The second `choose` condition will check for "action" is "calculate".

```
<j:when test="${action=='calculate'}">
```

Do some additional processing to get the calculation parameters, we could have done it in the top `evaluate` block using JavaScript.

```
<g:evaluate>
    var operator = RP.getParameterValue('operator') || '';
    var result = '';

    if(operator != ''){
        var firstNumber = parseFloat(RP.getParameterValue('first_number')) || 0;
        var secondNumber = parseFloat(RP.getParameterValue('second_number')) || 0;
```

```

if( operator == 'add'){
    result = firstNumber + secondNumber;
} else if( operator == 'sub' ) {
    result = firstNumber - secondNumber;
} else if( operator == 'mult' ) {
    result = firstNumber * secondNumber;
} else if( operator == 'div' ) {
    result = firstNumber / secondNumber;
}
}
</g:evaluate>
```

Output the result as before

```

<j:if test="${result!=''}" >
    Result ${result}
    <hr/>
    <a href="dynamic_calculator.do">Do more math!</a>
</j:if>
</j:when>
```

Create a default `otherwise` condition to execute if action is neither blank nor "calculate", meaning someone accidentally (or intentionally) changed the `action` parameter in the URL.

```

<j:otherwise>
    An Error Has Occured
</j:otherwise>
```

End the `choose` condition.

```
</j:choose>
```

End Jelly

```
</j:jelly>
```

## Advanced Form

This example replicates the Simple Form example as multi-step form (like a Wizard). The key concepts here were to change the form to present the input in smaller steps and to preserve the data for the entire process in each step.

In addition to the basic form fields we also need to add navigation forwards/backwards and the ability to reset the form entirely. This is accomplished by adding a `step` value which the page uses to determine which step to show, and passing all of the variables each time.

### UI Page

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">

<g:evaluate>
    var step = parseInt( RP.getParameterValue("step"),10 ) || 1;
    var formObj = {};

    getFormValue('name', "");
    getFormValue('age', 18);
    getFormValue('email', "");
    getFormValue('phone', "");

    function getFormValue(name, defaultValue){
        formObj[name] = RP.getParameterValue(name) || defaultValue;
    }
</g:evaluate>

<form>
    <j:choose>

        <j:when test="${step==1}">
            <h4>Welcome To Advanced Wizard</h4>
            <p>Enter your name to continue</p>
            <input name="age" value="${formObj.age}" type="hidden" />
            <input name="email" value="${formObj.email}" type="hidden" />
            <input name="phone" value="${formObj.phone}" type="hidden" />
            <input name="name" value="${formObj.name}" />
            <input name="step" value="${step+1}" type="hidden" />
        </j:when>

        <j:when test="${step==2}">
            <h4>Hi ${formObj.name}</h4>
            <p>What is your age?</p>
            <input name="age" value="${formObj.age}" />
            <input name="email" value="${formObj.email}" type="hidden" />
            <input name="phone" value="${formObj.phone}" type="hidden" />
            <input name="name" value="${formObj.name}" type="hidden"/>
            <input name="step" value="${step+1}" type="hidden" />
        </j:when>

        <j:when test="${step==3}">
            <h4>Great!</h4>
            <p>So you are ${formObj.age} years old? Interesting.</p>
            <p>How can we contact you?</p>
            <input name="age" value="${formObj.age}" type="hidden" />
            <label>Email$[SP]
                <input name="email" value="${formObj.email}" />
            </label>
            $[SP]
            <label>Phone$[SP]
                <input name="phone" value="${formObj.phone}" />
            </label>
            <input name="name" value="${formObj.name}" type="hidden"/>
            <input name="step" value="${step+1}" type="hidden" />
        </j:when>

        <j:when test="${step==4}">
            <h4>Thanks!</h4>
    </j:when>

```

```
<p>${formObj.name} we will contact you at ${formObj.email} or ${formObj.phone}</p>
</j:when>

</j:choose>
<br />

<j:if test="${step!=4}">
    <input type="submit" />
</j:if>
<br/>
<j:if test="${step!=1}">
    <a href="dynamic_form.do?age=${formObj.age}${[AMP]email=${formObj.email}${[AMP]phone=${formObj.phone}${[AMP]name=${formObj.name}}${[AMP]}>${formObj.name}</a>
</j:if>
<br/>
<a href="dynamic_form.do">Start Over</a>

</form>

</j:jelly>
```

### Output - Step 1

#### Welcome To Advanced Wizard

Enter your name to continue

[Start Over](#)

### Output - Step 2

Hi Sal

What is your age?

[Go Back](#)

[Start Over](#)

### Output - Step 3

**Great!**

So you are 18 years old? Interesting.

How can we contact you?

Email  Phone

[Go Back](#)

[Start Over](#)

#### Output - Step 4

**Thanks!**

Sal we will contact you at [tedest@test.com](mailto:tedest@test.com) or 123-123-1234

[Go Back](#)

[Start Over](#)

---

#### Walkthrough

Declare the Document Type

```
<?xml version="1.0" encoding="utf-8" ?>
```

Start Jelly

```
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
```

Begin processing the page parameters, since no var is specified the intention is to use the variables within the script on the page.

Get the `step` variable if it exists or set to the default.

```
var step = parseInt( RP.getParameterValue("step"),10) || 1;
```

Create a generic form object to hold all the values instead of multiple global variables.

```
var formObj = {};
```

Use the helper function to get and set the page parameters to script variables.

```
getFormValue('name', '');
getFormValue('age', 18);
getFormValue('email', '');
getFormValue('phone', ''');
```

Helper function to get parameters and set them to the form object property.

```
function getFormValue(name, defaultValue){
```

Using bracket notation, check for the parameter and set the property to that OR the default value if the property is not present.

```
    formObj[name] = RP.getParameterValue(name) || defaultValue;
}
```

End the evaluate.

```
</g:evaluate>
```

Begin the form to enclose the inputs we want to submit. Leaving off the attributes will result in the default settings: submitting to the same page URL and as GET (passing parameters in the URL)

```
<form>
```

Begin a condition block to choose between the different form sections based on the `step`

```
<j:choose>
```

Condition block for the first step. Even though we are only displaying `name` there are hidden variables for all the other variables. This is so the values are preserved always when going back and forth.

```
<j:when test="${step==1}">
    <h4>Welcome To Advanced Wizard</h4>
    <p>Enter your name to continue</p>
    <input name="age" value="${formObj.age}" type="hidden" />
    <input name="email" value="${formObj.email}" type="hidden" />
    <input name="phone" value="${formObj.phone}" type="hidden" />
    <input name="name" value="${formObj.name}" />
```

This variable is special because it will pass the value of the step we want to go to. It takes the current `step` and adds one to it.

```
<input name="step" value="${step+1}" type="hidden" />
```

```
</j:when>
```

Start of the second condition. It is basically the same as the first but the `age` variable is shown and the message is slightly different.

```
<j:when test="${step==2}">
    <h4>Hi ${formObj.name}</h4>
    <p>What is your age?</p>
    <input name="age" value="${formObj.age}" />
    <input name="email" value="${formObj.email}" type="hidden" />
    <input name="phone" value="${formObj.phone}" type="hidden" />
    <input name="name" value="${formObj.name}" type="hidden"/>
    <input name="step" value="${step+1}" type="hidden" />
</j:when>
```

Third step, again mostly the same only displaying email and phone.

```
<j:when test="${step==3}">
    <h4>Great!</h4>
    <p>So you are ${formObj.age} years old? Interesting.</p>
    <p>How can we contact you?</p>
    <input name="age" value="${formObj.age}" type="hidden" />
    <label>Email${[SP]}
        <input name="email" value="${formObj.email}" />
    </label>
    ${[SP]}
    <label>Phone${[SP]}
        <input name="phone" value="${formObj.phone}" />
    </label>
    <input name="name" value="${formObj.name}" type="hidden"/>
    <input name="step" value="${step+1}" type="hidden" />
</j:when>
```

Final step, outputting all the information.

```
<j:when test="${step==4}">
    <h4>Thanks!</h4>
    <p>${formObj.name} we will contact you at ${formObj.email} or ${formObj.phone}</p>
</j:when>
```

End the condition block.

```
</j:choose>
<br />
```

If the step is not 4 (the final step) show the Submit button.

```
<j:if test="${step!=4}">
    <input type="submit" />
</j:if>
<br/>
```

If the step is not 1 (the first step) show the back button.

```
<j:if test="${step!=1}">
```

Build the URL to return back. Remember going back and forth the state (all our form values) is not preserved. Sometimes these values are stored in Sessions (Server Side) or Cookies (Client Side). In this case we are trying to keep it simple so we keep track of our state in the URL.

```
<a href="dynamic_form.do?age=${formObj.age}${[AMP]}email=${formObj.email}${[AMP]}phone=${formObj.phone}${[AMP]}name=${  
</j:if>  
<br/>
```



Because the state is in the URL, if we go to the base page with no parameters we clear all the values.

```
<a href="dynamic_form.do">Start Over</a>
```

End the Form

```
</form>
```

End Jelly

```
</j:jelly>
```

## Suggested Exercises

1. Add an additional step before confirmation to gather the users location
2. Add an initial Menu form before all forms to allow user to check off which forms to see

## Multi-Page Application

---

This section will show how to link several pages together to build an application.

In multi-page applications it is important to create re-usable components so each page does no repeat any content.

# Simple Portal

This example will cover creating a simple portal, linking several pages together using common components.

There are four core components we want to make to re-use in our portal. The header and footer, a menu, and a layout to tie all the components together. We will also create a Style Sheet and JavaScript application file.

## Project Layout

```
UI Pages
|- portal_welcome
UI Macros
|- portal_header
|- portal_footer
|- portal_layout
|- portal_menu
Style Sheets
|- portal
```

### UI Page - portal\_welcome

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
    <g:inline template="portal_layout.xml">
        <h2>Welcome to the Jelly Portal</h2>
        <br/>
        <hr />

        <p>This is an example of creating a Portal using Jelly and ServiceNow</p>
        <br/>
        <p>
            Features include:
            <br/>
            <ul>
                <li>Header</li>
                <li>Footer</li>
                <li>Menus</li>
                <li>This List</li>
            </ul>
        </p>
    </g:inline>
</j:jelly>
```

## Output

# Jelly Portal

Welcome Home

## Welcome to the Jelly Portal

This is an example of creating a Portal using Jelly and ServiceNow

Features include:

Header  
Footer  
Menus  
This List

© 2015 My Company

## Walkthrough

Declare the document type

```
<?xml version="1.0" encoding="utf-8" ?>
```

Start Jelly

```
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
```

Immediately include the "portal\_layout" UI Macro.

```
<g:inline template="portal_layout.xml">
```

Create the content of the page. The UI Macro will insert this body into the proper place in the body of the layout.

```
<h2>Welcome to the Jelly Portal</h2>
<br/>
<hr />

<p>This is an example of creating a Portal using Jelly and ServiceNow</p>
<br/>
<p>
    Features include:
    <br/>
    <ul>
        <li>Header</li>
        <li>Footer</li>
        <li>Menus</li>
        <li>This List</li>
    </ul>
</p>
```

```
</p>
```

End the inline and the content

```
</g:inline>
```

End Jelly

```
</j:jelly>
```

### **UI Macro - portal\_header**

Creates a simple header

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">

    <div class="header">
        <h1>Jelly Portal</h1>
    </div>
</j:jelly>
```

### **UI Macro - portal\_footer**

Creates a simple footer, notice the use of the \$[AMP] constant to output the Copyright symbol.

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
    <p>
        $[AMP]copy; 2015 My Company
    </p>
</j:jelly>
```

### **UI Macro - portal\_menu**

Create a simple menu with the links for main pages included in the portal.

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
    <div class="menu">
        <ul>
            <li>
                <a href="portal_welcome.do">Welcome</a>
            </li>
            <li>
                <a href="portal_home.do">Home</a>
            </li>
        </ul>
    </div>
</j:jelly>
```

### **UI Macro - portal\_layout**

This is the most important piece of the portal. All pages must use this layout page, and it is even more than just layout, it includes all the scripts and styles.

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">

<link href="669071f42b2df1009728f70a89da156c.cssdbx" type="text/css" rel="stylesheet"></link>

<div class="portal">
    <div class="top">
        <g:inline template="portal_header.xml" />
        <g:inline template="portal_menu.xml" />
    </div>
    <div class="main">
        <g:insert />
    </div>

    <div class="bottom">
        <g:inline template="portal_footer.xml" />
    </div>
</div>

</j:jelly>
```

## Walkthrough

Set up the Jelly Macro

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
```

Include our Style Sheet. This will change depending on the sys\_id of the Style Sheet.

```
<link href="669071f42b2df1009728f70a89da156c.cssdbx" type="text/css" rel="stylesheet"></link>
```

Start creating the layout of the page using `divs` and `classes`.

```
<div class="portal">
    <div class="top">
```

Include the UI Macros for the Header and Menu

```
<g:inline template="portal_header.xml" />
<g:inline template="portal_menu.xml" />
</div>
```

Begin the main content section

```
<div class="main">
```

This `insert` will include the body from the tag that called this macro (in the UI Page). This is what makes this page wrap around the content of a particular UI Page.

```
<g:insert />
</div>
```

Start the bottom footer section.

```
<div class="bottom">
```

Include the footer macro.

```
<g:inline template="portal_footer.xml" />
```

End the page

```
    </div>
</div>

</j:jelly>
```

## Suggested Exercises

1. Add an additional "About" page.
2. Add a page which has functionality to use a GET parameter.

## AJAX Application

---

AJAX or Asynchronous JavaScript and XML is a term that has come to mean websites or applications which load the core application once and then make subsequent requests, without reloading the page, to retrieve data. The goal is to make better performing and more native feeling applications, but it does require you to think a bit differently on how to design the application.

In a traditional page you return all the application (layout HTML, Style Sheets, JavaScript) and data all at once and so all those components can be included on your Jelly Page. In an AJAX style page you would return the application but to get the data you will make additional AJAX requests to a service (in our case we can use GlideRecord and GlideAjax) then change the page based on the response.

Practically speaking AJAX style applications could not be further away from Jelly. Jelly is executed exclusively on the server and AJAX applications on the client, however I am including this example to show how Jelly can create an AJAX page/application.

It is still necessary - and always will be - to have a server side component to any application. Many frameworks now build templating frameworks that do exactly what Jelly does. This allows for code reuse and simplified design of the core forms and UI.

# AJAX Form

This example will walk through setting up a simple form and a Script Include to act as a "service" that will provide data. The page will have a few different views to give you a sense of how to change the page based on the content. Home, Incident and Knowledge sections will be created. The user can navigate between them using a simple menu. On each of the Knowledge and Incident panels the user can search for a record based on its number.

The search uses AJAX to call a Script Include passing the search term and expecting back an Object which it will iterate over, adding the results to the content section of the correct panel.

## Script Include - JellyAJAX

```
gs.include("JSON");

var JellyAJAX = Class.create();

JellyAJAX.prototype = Object.extendsObject(AbstractAjaxProcessor, {

    getIncident : function(){
        var incidentNumber = this.getParameter('sysparm_number').toString();
        var incidentRec = new GlideRecord('incident');
        incidentRec.addQuery("number",incidentNumber);
        incidentRec.query();

        if( incidentRec.next() ){
            return new JSON().encode({
                "Number" : incidentRec.number.toString(),
                "Short Description" : incidentRec.short_description.toString(),
                "Description" : incidentRec.description.toString()
            });
        }
    },

    getKnowledge : function(){
        var knowledgeNumber = this.getParameter('sysparm_number').toString();
        var knowledgeRec = new GlideRecord('kb_knowledge');
        knowledgeRec.addQuery("number",knowledgeNumber);
        knowledgeRec.query();

        if( knowledgeRec.next() ){
            return new JSON().encode({
                "Number" : knowledgeRec.number.toString(),
                "Short Description" : knowledgeRec.short_description.toString(),
                "Description" : knowledgeRec.description.toString()
            });
        }
    }
});
```

## Walkthrough

The Script must have Client Callable enabled to work as an AJAX endpoint.

Almost every Script Include intending to be an endpoint should use JSON as the mechanism to send data back and forth. This makes it much easier to use the data on the client side.

```
gs.include("JSON");
```

Create the class JellyAJAX

```
var JellyAJAX = Class.create();
```

Extend the prototype (definition) from AbstractAjaxProcessor.

Make sure to not include an `initialize` function as this will clobber the `initialize` in the `AbstractAjaxProcessor` prototype, which will break AJAX functionality.

```
JellyAJAX.prototype = Object.extend(AbstractAjaxProcessor, {
```

`getIncident` will be called when the end user searches for an incident.

```
getIncident : function(){
```

The function expects 'sysparm\_number' to be set. If it is not the function will fail and will not return any response.

```
    var incidentNumber = this.getParameter('sysparm_number').toString();
```

Create and execute our GlideRecord query.

```
    var incidentRec = new GlideRecord('incident');
    incidentRec.addQuery("number",incidentNumber);
    incidentRec.query();

    if( incidentRec.next() ){
```

Create the response object by passing an object to the `encode` method in the `JSON()` object.

```
        return new JSON().encode({
            "Number" : incidentRec.number.toString(),
            "Short Description" : incidentRec.short_description.toString(),
            "Description" : incidentRec.description.toString()
        });
    }

},
```

Repeat the same function but for Knowledge.

```
getKnowledge : function(){
    var knowledgeNumber = this.getParameter('sysparm_number').toString();
    var knowledgeRec = new GlideRecord('kb_knowledge');
    knowledgeRec.addQuery("number",knowledgeNumber);
    knowledgeRec.query();

    if( knowledgeRec.next() ){
        return new JSON().encode({
            "Number" : knowledgeRec.number.toString(),
            "Short Description" : knowledgeRec.short_description.toString(),
            "Description" : knowledgeRec.description.toString()
        });
    }

});
```

This is just an example but in practice you would want to create private and public functions. For example:

This is the public function and is what you would "call" from the client side. It should retrieve the parameters needed, do any error checking, and then call the private function. I am encoding the result here as JSON instead of the private function. Logically this makes more sense as the private function should be usable without any other dependencies (having to decode the result when using it).

```
getKnowledge : function(){
    var knowledgeNumber = this.getParameter('sysparm_number').toString();
    return new JSON().encode( this._getKnowledge(knowledgeNumber) );
},
```

The private function now accepts `knowledgeNumber` as a parameter and returns only a plain object (not encoded).

```
_getKnowledge : function(knowledgeNumber){
    var knowledgeRec = new GlideRecord('kb_knowledge');
    knowledgeRec.addQuery("number",knowledgeNumber);
    knowledgeRec.query();

    if( knowledgeRec.next() ){
        return {
            "Number" : knowledgeRec.number.toString(),
            "Short Description" : knowledgeRec.short_description.toString(),
            "Description" : knowledgeRec.description.toString()
        }
    }
}
```

## UI Page - jelly\_ajax

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">

<style>
    .panel-link{
        cursor: pointer;
        margin-right: 10px;
        color: blue;
    }
</style>

<div>
    <span class="panel-link" data-panel="home">Home</span>
    <span class="panel-link" data-panel="incident">Incident</span>
    <span class="panel-link" data-panel="knowledge">KB</span>
</div>
<hr />
<div class="content">

    <div id="home" class="panel">
        <h2>Welcome to the AJAX Example</h2>
        <p>Clicking the links above will navigate to the individual forms</p>
        <p>The page will not reload, only data is being transferred.</p>
    </div>

    <div id="incident" class="panel">
        <h2>Incident</h2>

        <input id="incident-number" />
        <button id='get-incident' >Get Incident</button>
        <br/>
        <div id="incident-content">
        </div>
    </div>

    <div id="knowledge" class="panel">
```

```

<h2>Knowledge</h2>
<input id="knowledge-number" />
<button id='get-knowledge'>Get Knowledge</button>
<br/>
<div id="knowledge-content">
</div>
</div>
</div>

<script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/2.1.3/jquery.min.js" />

<script>
    jQuery.noConflict();
</script>

<script>

jQuery(document).ready(function(){
    jQuery(".panel").hide();
    jQuery("#home").show();

    jQuery(".panel-link").click(function(){
        showPanel( jQuery(this).attr("data-panel") );
    });

    jQuery("#get-incident").click(function(){
        var incidentNumber = jQuery("#incident-number").val();
        if( incidentNumber == "" ) return;

        getIncident(incidentNumber);
    });

    jQuery("#get-knowledge").click(function(){
        var knowledgeNumber = jQuery("#knowledge-number").val();
        if( knowledgeNumber == "" ) return;

        getKnowledge(knowledgeNumber);
    });
});

function showPanel(panel){
    jQuery(".panel").hide();
    jQuery("#" + panel).show();
}

function getIncident(number){
    var ga = new GlideAjax("JellyAJAX");
    ga.addParam('sysparm_name', 'getIncident');
    ga.addParam('sysparm_number', number);

    ga.getXML(function(response){
        var data = JSON.parse(response.responseText.getAttribute("answer"));
        jQuery("#incident-content").html("");

        if( data == null ) jQuery("<span></span>").html("Record not found.").appendTo(jQuery("#incident-content"));

        jQuery.map(data, function(item,key){
            jQuery("<span></span>").html(key + " : ").appendTo(jQuery("#incident-content"));
            jQuery("<span></span>").html(item).appendTo(jQuery("#incident-content"));
            jQuery("<br />").html(key).appendTo(jQuery("#incident-content"));
        });
    });
}

function getKnowledge(number){
    var ga = new GlideAjax("JellyAJAX");
    ga.addParam('sysparm_name', 'getKnowledge');
    ga.addParam('sysparm_number', number);

    ga.getXML(function(response){
        var data = JSON.parse(response.responseText.getAttribute("answer"));
        jQuery("#knowledge-content").html("");

        if( data == null ) jQuery("<span></span>").html("Record not found.").appendTo(jQuery("#knowledge-content"));

        jQuery.map(data, function(item,key){
    
```

```

        jQuery("<span></span>").html(key + " : ").appendTo(jQuery("#knowledge-content"));
        jQuery("<span></span>").html(item).appendTo(jQuery("#knowledge-content"));
        jQuery("<br />").html(key).appendTo(jQuery("#knowledge-content"));
    });

});

</script>
</j:jelly>

```

## Output

### Home

[Home](#) [Incident](#) [KB](#)

---

## Welcome to the AJAX Example

Clicking the links above will navigate to the individual forms

The page will not reload, only data is being transferred.

### Incident Error

[Home](#) [Incident](#) [KB](#)

---

## Incident

INC000001	<b>Get Incident</b>
-----------	---------------------

Record not found.

### Knowledge

[Home](#) [Incident](#) [KB](#)

---

## Knowledge

KB0000024	<b>Get Knowledge</b>
-----------	----------------------

Description :

Number : KB0000024

Short Description : Can't access SAP

### Walkthrough

Declare the document type

```
<?xml version="1.0" encoding="utf-8" ?>
```

## Start Jelly

```
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
```

Add a simple style for the Menu Links.

```
<style>
    .panel-link{
        cursor: pointer;
        margin-right: 10px;
        color: blue;
    }
</style>
```

Menu links, the `data-panel` attribute is used to determine which panel to switch to. This allows us to create a generic function for `panel-link` instead of one per each item.

Avoid using `onClick` attributes in cases like this, we should always be binding events using JavaScript to cleanly separate our concerns.

```
<div>
    <span class="panel-link" data-panel="home">Home</span>
    <span class="panel-link" data-panel="incident">Incident</span>
    <span class="panel-link" data-panel="knowledge">KB</span>
</div>
```

A line.

```
<hr />
```

Begin the "content" section.

```
<div class="content">
```

Each `div` under content is a "panel", only one panel will be shown at a time depending on the users selection.

```
<div id="home" class="panel">
    <h2>Welcome to the AJAX Example</h2>
    <p>Clicking the links above will navigate to the individual forms</p>
    <p>The page will not reload, only data is being transferred.</p>
</div>
```

The incident and knowledge panels are very similar. If we were going to repeat this many more times this would be a potential candidate to refactor into it's own UI Macro.

```
<div id="incident" class="panel">
    <h2>Incident</h2>
    <input id="incident-number" />
```

```

<button id='get-incident' >Get Incident</button>
<br/>
<div id="incident-content">
</div>

<div id="knowledge" class="panel">
<h2>Knowledge</h2>
<input id="knowledge-number" />
<button id='get-knowledge'>Get Knowledge</button>
<br/>
<div id="knowledge-content">
</div>
</div>
</div>

```

To simplify Binding Events and doing the DOM manipulation necessary to show the results, I am including jQuery here from a CDN. For the most part there is no issue with including jQuery this way. The only catch is you must declare `noConflict` mode immediately after including it.

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/2.1.3/jquery.min.js" />
```

No conflict will release the `$` variable back, this is necessary in ServiceNow because the Prototype Framework uses it.

```

<script>
  jQuery.noConflict();
</script>

```

Start our main application.

```
<script>
```

Wait for the `document` to be ready.

```
jQuery(document).ready(function(){
```

Start by hiding all the panels, this could also be done using CSS.

```
jQuery(".panel").hide();
```

Show the home panel.

```
jQuery("#home").show();
```

Bind an event to the click of the "panel-link" elements.

```
jQuery(".panel-link").click(function(){
```

Use the `data-panel` attribute as the panel name to show.

```
    showPanel( jQuery(this).attr("data-panel") );
});
```

Bind the click events on the "get-incident" and "get-knowledge" buttons.

```
jQuery("#get-incident").click(function(){
```

Get the search term in the input box for this panel.

```
    var incidentNumber = jQuery("#incident-number").val();
    if( incidentNumber == "" ) return;
```

Call the getIncident function passing in the number.

```
        getIncident(incidentNumber);
});
```

This functions exactly the same as "get-incident" except referencing the knowledge elements.

```
jQuery("#get-knowledge").click(function(){
    var knowledgeNumber = jQuery("#knowledge-number").val();
    if( knowledgeNumber == "" ) return;

    getKnowledge(knowledgeNumber);
});
```

`showPanel` will accept a panel name in, it will immediately hide all panels, then show the correct one.

```
function showPanel(panel){
    jQuery(".panel").hide();
    jQuery("#" + panel).show();
}
```

`getIncident` makes the AJAX call it expects number to be passed in.

```
function getIncident(number){
```

Create a new GlideAjax object that points to our Script Include

```
var ga = new GlideAjax("JellyAJAX");
```

Set the function we want to call.

```
ga.addParam('sysparm_name', 'getIncident');
```

Set the `sysparm_number` parameter to the number passed in.

```
ga.addParam('sysparm_number', number);
```

Make the actual AJAX call. `getXML` is asynchronous, meaning it makes the call and sets the function that should be called when the result is ready (the callback) so the User Interface is responsive during this activity. If we were to use the synchronous equivalent the user would not be able to do anything and the application would seem unresponsive.

The `function(response){}` is the start of the callback function. We are defining it directly here as an anonymous function but it is just as correct to define the function elsewhere in your script and pass the reference instead.

```
ga.getXML(function(response){
```

At this point we are in the callback function when the response is returned. The first thing is to get the "answer" portion of the XML Response then parse it using the built-in JSON object within the browser.

```
var data = JSON.parse(response.responseXML.documentElement.getAttribute("answer"));
```

Getting ready for the response, we clear the content section.

```
jQuery("#incident-content").html("");
```

If we got a `null` response then inform the user and stop immediately.

```
if( data == null ) jQuery("<span></span>").html("Record not found.").appendTo(jQuery("#incident-content"));
```

If actual results were returned, using the `map` function to iterate over the items in the object, output each to the content section using jQuery.

```
jQuery.map(data, function(item,key){
    jQuery("<span></span>").html(key + " : ").appendTo(jQuery("#incident-content"));
    jQuery("<span></span>").html(item).appendTo(jQuery("#incident-content"));
    jQuery("<br />").html(key).appendTo(jQuery("#incident-content"));
});
```

`getKnowledge` functions exactly like `getIncident` except it references knowledge elements and calls the `getKnowledge` function in the Script Include.

Again because these are so similar in practice it might be beneficial to refactor these into a more generic version.

```
function getKnowledge(number){
    var ga = new GlideAjax("JellyAJAX");
    ga.addParam('sysparm_name', 'getKnowledge');
    ga.addParam('sysparm_number', number);

    ga.getXML(function(response){
        var data = JSON.parse(response.responseXML.documentElement.getAttribute("answer"));
```

```
jQuery("#knowledge-content").html("");

if( data == null ) jQuery("<span></span>").html("Record not found.").appendTo(jQuery("#knowledge-content"));

jQuery.map(data, function(item,key){
    jQuery("<span></span>").html(key + " : ").appendTo(jQuery("#knowledge-content"));
    jQuery("<span></span>").html(item).appendTo(jQuery("#knowledge-content"));
    jQuery("<br />").html(key).appendTo(jQuery("#knowledge-content"));
});

});
```

});

End the main script

```
</script>
```

End Jelly

```
</j:jelly>
```

## Suggested Exercises

1. Add an additional section for Change

# Form Widgets

I am using the term "Widget" to mean any element added to a form that does not mean to take the place of an element input. Here we will look at two such examples the first uses a value from the form to draw a custom gauge.

## Simple Gauge

This first example will take the value from a field (Percent Complete) and draw a custom gauge using justGage. This same technique can be used to load virtually any JavaScript plugin.

### Setup

1. Download and install [justGage](#)
2. Create a UI Script for raphael.2.1.0.min and justGage.
3. If ServiceNow throws an error when trying to save raphael
  - i. Add Script to the list view and use the list edit to input the script
  - ii. Link to raphael on a CDN <https://cdnjs.cloudflare.com/ajax/libs/raphael/2.1.2/raphael-min.js>
4. Create the UI Macro and a Formatter
5. Create a Field Named Percent Complete (u\_percent\_complete) as an integer.

### UI Macro

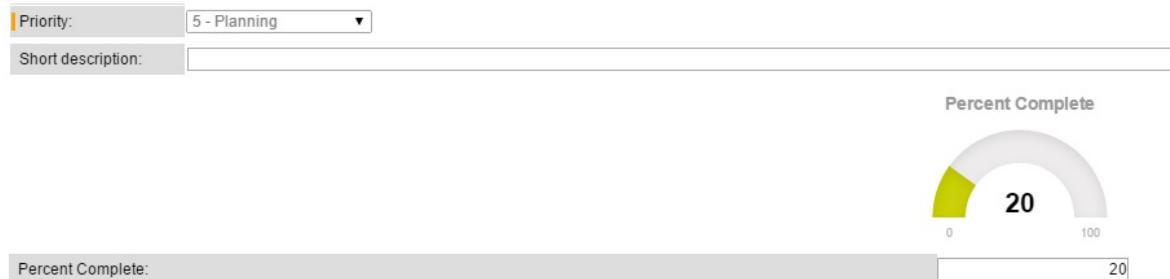
```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
  <table class="wide" cellspacing="2">
    <tbody>
      <tr>
        <td>
          <div id="gauge" class="200x160px"></div>
        </td>
      </tr>
    </tbody>
  </table>

  <script src="raphael.2.1.0.jsdbx" />
  <script src="justgage.jsdbx" />

  <g2:set_if var="jvar_percent_complete" test="[$[current.u_percent_complete]!='']" true="[$[current.u_percent_complete]]" false="0" />

  <script>
    var g = new JustGage({
      id: "gauge",
      value: $[jvar_percent_complete],
      min: 0,
      max: 100,
      title: "Percent Complete"
    });
  </script>
</j:jelly>
```

### Output



## Walkthrough

Declare the document type

```
<?xml version="1.0" encoding="utf-8" ?>
```

Start Jelly

```
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
```

Create a table container so the widget displays on the form correctly

```
<table class="wide" cellspacing="2">
  <tbody>
    <tr>
      <td>
```

Add the container for the custom gauge

```
<div id="gauge" class="200x160px"></div>
```

Close the container

```
</td>
</tr>
</tbody>
</table>
```

Add the scripts for raphael and justGauge.

```
<script src="raphael.2.1.0.jsdbx" />
<script src="justgauge.jsdbx" />
```

Conditionally set a variable `jvar_percent_complete` to either the value of Percent Complete (if its not blank) or 0.

```
<g2:set_if var="jvar_percent_complete" test="#[current.u_percent_complete!='']" true="#[current.u_percent_complet
```

Add the script to initialize the gauge.

```
<script>
    var g = new JustGage({
        id: "gauge",
        value: $[jvar_percent_complete],
        min: 0,
        max: 100,
        title: "Percent Complete"
    });
</script>
```

End Jelly.

```
</j:jelly>
```

## Inline List

This example creates a list inline on a form showing the last 5 Incidents created.

### UI Macro

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">

    <g2:evaluate>
        var incidentRec = new GlideRecord('incident');
        incidentRec.orderByDesc('sys_created_on');
        incidentRec.setLimit(5);
        incidentRec.query();

        var row_class = "list_odd";
    </g2:evaluate>

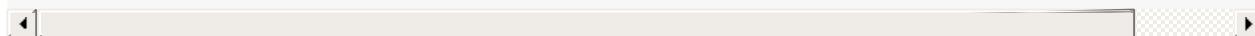
    <table cellspacing="2" width="100%">
        <tbody>
            <tr>
                <td>
                    <h3>Last 5 Incidents</h3>
                    <table class="wide">
                        <thead>
                            <tr class="header">
                                <td class="column_head">Number</td>
                                <td class="column_head">Short Description</td>
                                <td class="column_head">Opened</td>
                            </tr>
                        </thead>

                        <j2:while test="${incidentRec.next()}">
                            <tr class="list_row ${row_class}">
                                <td><a href="incident.do?sys_id=${incidentRec.sys_id}">${incidentRec.number.toString()}</a>
                                <td>${incidentRec.short_description.toString()}</td>
                                <td>${incidentRec.sys_created_on.getDisplayValue()}</td>
                            </tr>
                        </j2:while>

                        <g2:evaluate>
                            if( row_class == "list_odd" ){
                                row_class = "list_even";
                            } else {
                                row_class = "list_odd";
                            }
                        </g2:evaluate>
                    </table>
                </td>
            </tr>
        </tbody>
    </table>
</j:jelly>
```

```

        </tr>
    </tbody>
</table>
</j:jelly>
```



## Output

Number	Short Description	Opened
INC0010078	Urgent: Exchange Server Down	2015-01-10 21:17:11
INC0010055	Reset the password for System Administrator on	2014-12-04 07:10:02
INC0010017	Executive Laptop Broken	2014-10-26 18:09:45
INC0010002	Getting 'Site Blocked' Warning	2014-08-21 09:34:33
INC0000055	SAP Sales app is not accessible	2014-07-29 21:49:39

## Walkthrough

Declare the document type

```
<?xml version="1.0" encoding="utf-8" ?>
```

Start Jelly

```
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
```

In the second phase, create a GlideRecord to Incident and get the last 5 Incidents created.

```

<g2:evaluate>
    var incidentRec = new GlideRecord('incident');
    incidentRec.orderByDesc('sys_created_on');
    incidentRec.setLimit(5);
    incidentRec.query();
```

To get the colored row banding create a `row_class` variable.

```

    var row_class = "list_odd";
</g2:evaluate>
```

Create the container table.

```

<table cellspacing="2" width="100%">
    <tbody>
        <tr>
            <td>
                <h3>Last 5 Incidents</h3>
```

## Create the content Table

```
<table class="wide">
```

Add some table headers.

```
<thead>
  <tr class="header">
    <td class="column_head">Number</td>
    <td class="column_head">Short Description</td>
    <td class="column_head">Opened</td>
  </tr>
</thead>
```

Iterating over `incidentRec` output the record rows.

```
<j2:while test="${[incidentRec.next()]}>
```

Set the `row_class` so we can get the row banding.

```
<tr class="list_row ${[row_class]}>
```

The first column should have a link to the record since users will expect to be able to click it.

```
<td><a href="incident.do?sys_id=${[incidentRec.sys_id]}>${[incidentRec.number.toString()]}</a></td>
```

Output the remaining values. Remember for fields that need to be localized, like dates, to use `getDisplayValue` to avoid getting items as their database values.

```
<td>${[incidentRec.short_description.toString()]}</td>
<td>${[incidentRec.sys_created_on.getDisplayValue()]}</td>
</tr>
```

Set the row banding variable.

```
<g2:evaluate>
  if( row_class == "list_odd" ){
    row_class = "list_even";
  } else {
    row_class = "list_odd";
  }
</g2:evaluate>
```

End the iteration

```
</j2:while>
```

End the content table

```
</table>
```

End the container table

```
    </td>
  </tr>
</tbody>
</table>
```

End Jelly

```
</j:jelly>
```

# AJAX Form Widget

Like normal form Widgets, AJAX Widgets may appear on a form but the data or functionality uses AJAX as opposed to being returned as part of the Jelly Page.

Let's rewrite the Latest Incidents example as an AJAX Form Widget

## Setup

### UI Macro

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">



|                                                                                                                                                                                                                                                                  |                   |                   |        |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------|-------------------|--------|
| <h3>Last 5 Incidents</h3> <table class="wide"> <thead> <tr class="header"> <td class="column_head">Number</td> <td class="column_head">Short Description</td> <td class="column_head">Opened</td> </tr> </thead> <tbody id="latest-incidents"> </tbody> </table> | Number            | Short Description | Opened |
| Number                                                                                                                                                                                                                                                           | Short Description | Opened            |        |



<script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/2.1.3/jquery.min.js" />
<script>
    jQuery.noConflict();
</script>

<script>
    var sys_id = "[current.sys_id]"
    var row_class = "list_odd";

    var incidentRec = new GlideRecord('incident');
    incidentRec.orderByDesc('sys_created_on');
    incidentRec.setLimit(5);
    incidentRec.query(function(incidentRec){
        jQuery("#latest-incidents").html("");

        while(incidentRec.next()){
            var tableRow = jQuery("<tr></tr>").addClass("list_row").addClass(row_class).appendTo(jQuery("#latest-incidents"));
            var tableCell = jQuery("<td></td>").appendTo(tableRow);
            var cellLink = jQuery("<a></a>").attr("href", "incident.do?sys_id=" + incidentRec.sys_id).html(incidentRec.short_description);
            jQuery("<td></td>").html(incidentRec.short_description).appendTo(tableRow);
            jQuery("<td></td>").html(incidentRec.sys_created_on).appendTo(tableRow);

            if( row_class == "list_odd" ){
                row_class = "list_even";
            } else {
                row_class = "list_odd";
            }
        }
    });
</script>
```

```
</j:jelly>
```

## Output

Number	Short Description	Opened
INC0010078	Urgent Exchange Server Down	2015-01-10 21:17:11
INC0010055	Reset the password for System Administrator on	2014-12-04 07:10:02
INC0010017	Executive Laptop Broken	2014-10-26 18:09:45
INC0010002	Getting 'Site Blocked' Warning	2014-08-21 09:34:33
INC0000055	SAP Sales app is not accessible	2014-07-29 21:49:39

## Walkthrough

Declare the document type

```
<?xml version="1.0" encoding="utf-8" ?>
```

Start Jelly

```
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
```

Create the container table

```
<table cellspacing="2" width="100%">
  <tbody>
    <tr>
      <td>
        <h3>Last 5 Incidents</h3>
        <table class="wide">
```

Create the Table Headers

```
<thead>
  <tr class="header">
    <td class="column_head">Number</td>
    <td class="column_head">Short Description</td>
    <td class="column_head">Opened</td>
  </tr>
</thead>
```

Create the table body with an ID, this makes it easier for us to place the dynamic elements on the page.

```
<tbody id="latest-incidents">
</tbody>
</table>
```

End the container table

```
</td>
</tr>
</tbody>
</table>
```

Include jQuery from CDN. If jQuery can be depended on either globally or somewhere else on the page it's not necessary to load it in every Macro.

In cases where I have had common scripts like this, I create just a "dependencies" UI Macro which I include as a formatter at the top of the form. That way I can be sure all dependencies are loaded in time, and I don't need to check in each Macro.

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/2.1.3/jquery.min.js" />
<script>
    jQuery.noConflict();
</script>
```

Main Script

```
<script>
```

`sys_id` is not used in this script but I included it in case you want to use `sys_id` in your script.

```
var sys_id = "${current.sys_id}"
```

Create the row banding variable.

```
var row_class = "list_odd";
```

Set up the GlideRecord query.

```
var incidentRec = new GlideRecord('incident');
incidentRec.orderByDesc('sys_created_on');
incidentRec.setLimit(5);
```

Query using asynchronous methods to prevent the UI from locking.

```
incidentRec.query(function(incidentRec){
```

Clear the container.

```
jQuery("#latest-incidents").html("");
```

Iterate over the results creating a table row for each.

```
while(incidentRec.next()){
```

Create the row adding the row banding class. (Stored in variable so it's easier to append child elements to it)

```
var tableRow = jQuery("<tr></tr>").addClass("list_row").addClass(row_class).appendTo(jQuery("#latest-incidents"));
```

Create and append a Cell to the Row

```
var tableCell = jQuery("<td></td>").appendTo(tableRow);
```

Create a link (users will expect to be able to click the column) and add it to the cell.

```
var cellLink = jQuery("<a></a>").attr("href", "incident.do?sys_id=" + incidentRec.sys_id).html(incidentRec.number).append
```

Add the remaining data cells.

```
jQuery("<td></td>").html(incidentRec.short_description).appendTo(tableRow);
jQuery("<td></td>").html(incidentRec.sys_created_on).appendTo(tableRow);
```

Toggle between the two row banding classes

```
        if( row_class == "list_odd" ){
            row_class = "list_even";
        } else {
            row_class = "list_odd";
        }
    });
}
```

End the Main script

```
</script>
```

End Jelly

```
</j:jelly>
```

## Custom Formatters

---

Formatters are a great way to add custom functionality without creating an entirely new UI Page. Since they can be added to standard ServiceNow forms you can easily expand what a user can do on a form while keeping a very simple user experience.

In this chapter we will look at a few different examples of formatters that can serve as the basis for almost any type of functionality.

# Custom Element

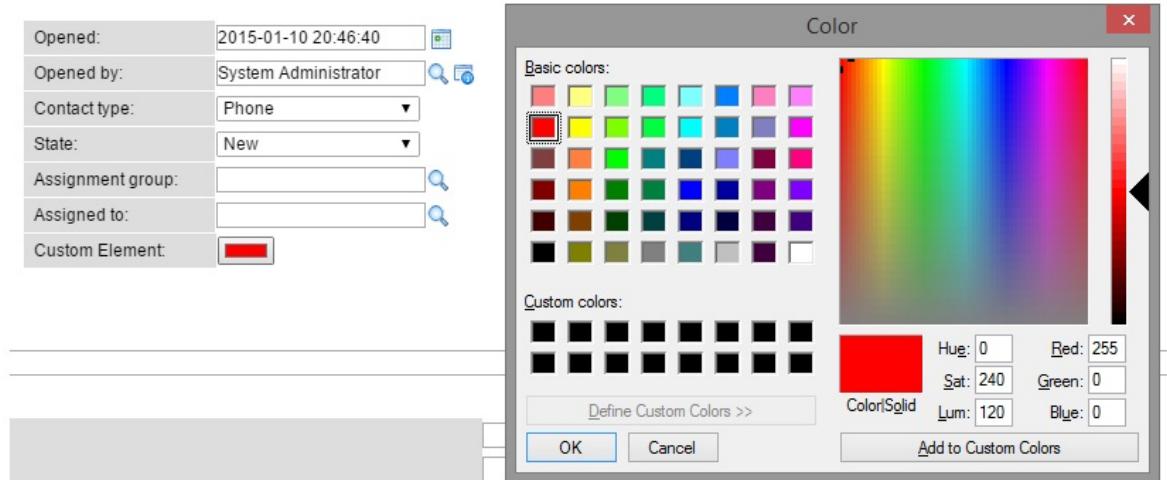
A custom element is a way to create a new type of input for a ServiceNow field. ServiceNow does a good job of covering most of the basic input types but there are cases where users may want a different type of input.

To create a custom element on a form I usually start by creating a String field with a size I know can accommodate the data that will be in there. Other fields types can be used but you will need to make sure to format the data to fit in the field and it's usually not necessary.

## UI Macro - my\_element

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
    <table cellspacing="2" width="100%">
        <tbody>
            <tr>
                <td id="label.incident.u_custom" title="" data-type="label" choice="0" nowrap="true" class="label label_label">
                    <span id="status.incident.u_custom" class="label_description" title="" mandatory="false" oclass="">
                        <label for="u_custom" onclick="return labelClicked(this);">Custom Element:</label>
                    </span>
                </td>
                <td>
                    <input id="incident.u_custom" type="color" onchange="onChange('incident.u_custom');" name="incident.u_custom" value="#ff0000" style="width: 100%; height: 10px;" />
                </td>
            </tr>
        </tbody>
    </table>
</j:jelly>
```

## Output



## Walkthrough

Declare the document type

```
<?xml version="1.0" encoding="utf-8" ?>
```

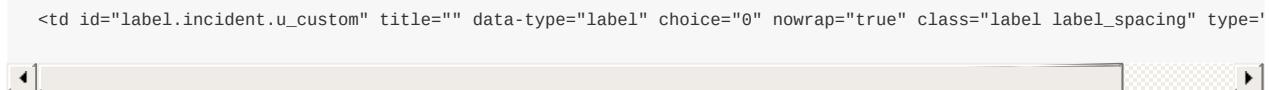
Start Jelly

```
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
```

Since standard ServiceNow forms use tables to layout elements we must also use tables. This replicates the structure that standard elements use.

```
<table cellspacing="2" width="100%">
<tbody>
    <tr>
```

Adding a form label makes your element look more like ServiceNow standard elements.



```
<td id="label.incident.u_custom" title="" data-type="label" choice="0" nowrap="true" class="label label_spacing" type='button'>[<| |>]</td>
```

Although standard policies will probably not work with this field, since it is custom, this is not really functional but serves to line up the field correctly.

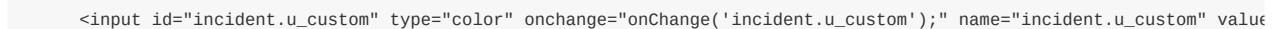
```
<span id="status.incident.u_custom" class="label_description" title="" mandatory="false" oclass="">$[SP]</span>
```

The actual label, functionally it also has the ability to be right-clicked like normal fields.

```
<label for="u_custom" onclick="return labelClicked(this);>Custom Element:</label>
</td>
```

Start the actual new input, in this case the HTML5 Color Picker input.

Notice the field name `u_custom` in your custom element replace this with your field name. The `$[current.u_custom]` expression puts the correct value on load into the input.



```
<input id="incident.u_custom" type="color" onchange="onChange('incident.u_custom');" name="incident.u_custom" value=$[current.u_custom]>[<| |>]
```

End the element and container table.

```
</td>
</tr>
</tbody>
</table>
```

End Jelly

```
</j:jelly>
```

## Mobile Page

---

Mobile is increasingly becoming the way people access websites. ServiceNow is of course taking advantage of that by creating mobile views of their standard forms and interface but there may be times when we want to create mobile specific versions of UI Pages.

Outside of building an App specific to a platform there are two ways you can create a mobile page. The first is to create a page that is mobile-only. This means detecting if the user is accessing the page through mobile or desktop and generating a page specifically for that view. The second way is to create a Responsive view. Which means the page can dynamically change its content based on the size of the screen viewing it.

There are other considerations, such as events, which need to be tweaked for mobile. For instance, creating "swipes" and replacing mouse clicks with "touches".

It is also worth noting that even though phones are more powerful and have faster networks they are not nearly as powerful as computers and so the amount of data sent back and forth should be taken into account.

# Mobile Only Page

The Mobile Only page is set up to render only on a mobile device. Usually this means you create the mobile version after the desktop version. This example is very simple, creating a mobile page, setting the viewport to match the device and creating some mobile events.

The intent is to create a page that does not require a user to zoom to view content, and to take advantage of natural events like swiping. These will make the page much more user friendly.

Please note this is meant more to show how to distinguish between mobile and desktop. You can use this code as a base but there are more detailed examples in the Full Examples section.

## UI Page

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
    <g:evaluate var="jvar_mobile_device_type" expression="session.getProperty('mobile_device_type')"/>

    <j:if test="${jvar_mobile_device_type!='null'}">

        <meta name="viewport" content="width=device-width, initial-scale=1" />
        <link rel="stylesheet" href="https://code.jquery.com/mobile/1.0/jquery.mobile-1.0.min.css" />

        <style type="text/css">
            .swipe-example{
                padding:40px;
                background-color:red;
            }
        </style>

        <h2>This is a mobile specific page</h2>
        <p>The device type is ${jvar_mobile_device_type}</p>
        <div class="swipe-example">
            Tap or Swipe Here.
        </div>

        <script src="//cdnjs.cloudflare.com/ajax/libs/jquery-mobile/1.4.1/jquery.mobile.min.js" />
        <script>
            jQuery.noConflict();
        </script>

        <script type="text/javascript">

            jQuery(document).on('pagecreate',function(){
                jQuery("#page_timing_div").hide();

                jQuery(".swipe-example").on("swipeleft",function(){
                    jQuery(this).html("You swiped left!");
                });

                jQuery(".swipe-example").on("swiperight",function(){
                    jQuery(this).html("You swiped right!");
                });

                jQuery(".swipe-example").on("tap",function(){
                    jQuery(this).html("You tapped!");
                });

                jQuery(".swipe-example").on("taphold",function(){
                    jQuery(this).html("You are long tapping!");
                });

            });
        </script>
    </j:if>

    <j:if test="${jvar_mobile_device_type=='null'}">
        <h2>This page is intended for Mobile Devices only.</h2>
    </j:if>

```

```
</j:if>
</j:jelly>
```

## Output

### Walkthrough

Declare the document type

```
<?xml version="1.0" encoding="utf-8" ?>
```

Start Jelly

```
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
```

Check to see if the user is using a mobile device.

If so return the Mobile Page

```
<j:if test="${jvar_mobile_device_type != 'null'}">
```

Viewport must be set in order to match the pages width to the devices width. Without it, it will require the user to scroll, which we specifically want to avoid.

```
<meta name="viewport" content="width=device-width, initial-scale=1" />
```

Include the jQuery Mobile CSS from a CDN.

```
<link rel="stylesheet" href="https://code.jquery.com/mobile/1.0/jquery.mobile-1.0.min.css" />
```

Add some styles for our page.

I am doing this here for the convenience of showing everything in a single page however this is a bad practice. Styles should always be included as a CSS file. This will save bandwidth as the CSS file is cacheable.

```
<style type="text/css">
    .swipe-example{
        padding:40px;
        background-color:red;
    }
</style>
```

Header that should only be displayed for mobile.

```
<h2>This is a mobile specific page</h2>
```

Output the device type. ServiceNow will recognize some common device types like iPhone and Android devices. Others are not supported and will most likely come back as "m". The standard ServiceNow mobile interface will only support the

most common devices, however that will not prevent us here. As long as the device is not known to be desktop it should work as mobile.

```
<p>The device type is ${jvar_mobile_device_type}</p>
```

Set up a `div` to do some events.

```
<div class="swipe-example">  
    Tap or Swipe Here.  
</div>
```

Include jQuery Mobile from a CDN. This is not required but it makes it much easier to bind events.

```
<script src="//cdnjs.cloudflare.com/ajax/libs/jquery-mobile/1.4.1/jquery.mobile.min.js" />
```

Of course we need to run noConflict mode so as to not interfere with Prototype.

```
<script>  
    jQuery.noConflict();  
</script>
```

Start the Main script section

```
<script type="text/javascript">
```

jQuery Mobile uses 'pagecreate' to signify the page is ready.

```
jQuery(document).on('pagecreate', function(){
```

Hide the Timing Div section in case its shown.

```
    jQuery("#page_timing_div").hide();
```

Bind some mobile events. Swipe left and right.

```
    jQuery(".swipe-example").on("swipeleft",function(){  
        jQuery(this).html("You swiped left!");  
    });  
  
    jQuery(".swipe-example").on("swiperight",function(){  
        jQuery(this).html("You swiped right!");  
    });
```

Tap is the equivalent of a click on the desktop.

```
    jQuery(".swipe-example").on("tap",function(){  
        jQuery(this).html("You tapped!");  
    });
```

Long-tapping can be used as the right-click.

```
jQuery(".swipe-example").on("taphold",function(){
    jQuery(this).html("You are long tapping!");
});

});
</script>
</j:if>
```

Create a specific section for desktop users to alert them the page is Mobile-only. You could also use this point to "include" the desktop version.

```
<j:if test="${jvar_mobile_device_type=='null'}">
    <h2>This page is intended for Mobile Devices only.</h2>
</j:if>
```

End Jelly

```
</j:jelly>
```

# Responsive Page

A Responsive Page, unlike the mobile only page, uses CSS Media Queries to dynamically change the site based on the width of the device accessing it. This has its advantages and disadvantages. There will be only one code base however it will be more complex. You also need to design and write the HTML to be use the responsive styles. It would be very difficult to go back and retro-fit a page to be responsive.

This example will create a simple responsive page with three modes: phone, tablet and desktop. The responsive classes will be `menu` and `section` each having a slightly different view in each mode. In practice there may be several responsive types of elements.

## UI Page

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
<meta name="viewport" content="width=device-width, initial-scale=1" />
<style>

*, body{
    padding: 0;
    margin: 0;
}

.body-vertical-spacer{
    display: none;
}

#page_timing_div{
    display: none;
}

#wrapper {
    width: 96%;
    max-width: 920px;
    margin: auto;
    padding: 2%;
}

.header{
    width: 100%;
    padding: 20px;
    background-color: lightblue;
}

.menu ul{
    list-style: none;
}

.menu li{
    font-weight: bold;
    display:inline-block;
    margin-right: 10px;
}

.content .section{
    width: 29%;
    margin: 2%;
    float: left;
}

.tablet{
    display: none;
}

.phone{
    display: none;
}

</style>
```

```

@media only screen
and (max-width : 767px) {
    .phone{
        display:initial;
    }

    .tablet{
        display: none;
    }

    .desktop{
        display: none;
    }

    .content .section{
        width: 96%;
        margin:2%;
    }

    .menu li{
        width: 96%;
        margin-right: 0px;
        padding:2%;
    }
}

@media only screen
and (min-width : 768px)
and (max-width : 1024px) {
    .phone{
        display: none;
    }

    .tablet{
        display:initial;
    }

    .desktop{
        display: none;
    }

    .content .section{
        width: 45%;
        margin:2%;
    }

}
}

</style>

<div id="wrapper">
    <div class="header">
        <h2>
            Responsive Example$[SP]
            <span class="phone">Phone</span>
            <span class="tablet">Tablet</span>
            <span class="desktop">Desktop</span>
        </h2>
    </div>
    <div class="menu">
        <ul>
            <li>Home</li>
            <li>Incident</li>
            <li>Problem</li>
            <li>Change</li>
            <li>Knowledge</li>
        </ul>
    </div>
    <div class="content">
        <p class="section">
            Lorem ipsum dolor sit amet, consectetur adipiscing elit. Cras auctor augue ligula, a volutpat sem t
        </p>
        <p class="section">
            Lorem ipsum dolor sit amet, consectetur adipiscing elit. Cras auctor augue ligula, a volutpat sem t
        </p>
    </div>
</div>

```

```

</p>
<p class="section">
    Lorem ipsum dolor sit amet, consectetur adipiscing elit. Cras auctor augue ligula, a volutpat sem t
</p>
</div>

</div>

</j:jelly>

```

**Output****Phone**

**Responsive Example Phone**

- Home
- Incident
- Problem
- Change
- Knowledge

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Cras auctor augue ligula, a volutpat sem tristique et. Nunc quis orci nec turpis lacinia rhoncus. Nullam id libero placerat, mollis tortor eget, vehicula erat. Etiam ac facilisis leo, sit amet rutrum ipsum. Morbi vitae placerat massa. Nam nisi augue, imperdiet eu aliquam a, accumsan ac nulla. Mauris ipsum mauris, rutrum ac sodales sed, sodales sed urna. Sed egestas et ipsum eu ullamcorper.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Cras auctor augue ligula, a volutpat sem tristique et. Nunc quis orci nec turpis lacinia rhoncus. Nullam id libero placerat, mollis tortor eget, vehicula erat. Etiam ac facilisis leo, sit amet rutrum ipsum. Morbi vitae placerat massa. Nam nisi augue, imperdiet eu aliquam a, accumsan ac nulla. Mauris ipsum mauris, rutrum ac sodales sed, sodales sed urna. Sed egestas et ipsum eu ullamcorper.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Cras auctor augue ligula, a volutpat sem tristique et. Nunc quis orci nec turpis lacinia rhoncus. Nullam id libero placerat, mollis tortor eget, vehicula erat. Etiam ac facilisis leo, sit amet rutrum ipsum. Morbi vitae placerat massa. Nam

**Phone**

## Responsive Example Tablet

Home Incident Problem Change Knowledge

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Cras auctor augue ligula, a volutpat sem tristique et. Nunc quis orci nec turpis lacinia rhoncus. Nullam id libero placerat, mollis tortor eget, vehicula erat. Etiam ac facilisis leo, sit amet rutrum ipsum. Morbi vitae placerat massa. Nam nisi augue, imperdiet eu aliquam a, accumsan ac nulla. Mauris ipsum mauris, rutrum ac sodales sed, sodales sed urna. Sed egestas et ipsum eu ullamcorper.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Cras auctor augue ligula, a volutpat sem tristique et. Nunc quis orci nec turpis lacinia rhoncus. Nullam id libero placerat, mollis tortor eget, vehicula erat. Etiam ac facilisis leo, sit amet rutrum ipsum. Morbi vitae placerat massa. Nam nisi augue, imperdiet eu aliquam a, accumsan ac nulla. Mauris ipsum mauris, rutrum ac sodales sed, sodales sed urna. Sed egestas et ipsum eu ullamcorper.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Cras auctor augue ligula, a volutpat sem tristique et. Nunc quis orci nec turpis lacinia rhoncus. Nullam id libero placerat, mollis tortor eget, vehicula erat. Etiam ac facilisis leo, sit amet rutrum ipsum. Morbi vitae placerat massa. Nam nisi augue, imperdiet eu aliquam a, accumsan ac nulla. Mauris ipsum mauris, rutrum ac sodales sed, sodales sed urna. Sed egestas et ipsum eu ullamcorper.

## Phone

### Responsive Example Desktop

Home Incident Problem Change Knowledge

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Cras auctor augue ligula, a volutpat sem tristique et. Nunc quis orci nec turpis lacinia rhoncus. Nullam id libero placerat, mollis tortor eget, vehicula erat. Etiam ac facilisis leo, sit amet rutrum ipsum. Morbi vitae placerat massa. Nam nisi augue, imperdiet eu aliquam a, accumsan ac nulla. Mauris ipsum mauris, rutrum ac sodales sed, sodales sed urna. Sed egestas et ipsum eu ullamcorper.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Cras auctor augue ligula, a volutpat sem tristique et. Nunc quis orci nec turpis lacinia rhoncus. Nullam id libero placerat, mollis tortor eget, vehicula erat. Etiam ac facilisis leo, sit amet rutrum ipsum. Morbi vitae placerat massa. Nam nisi augue, imperdiet eu aliquam a, accumsan ac nulla. Mauris ipsum mauris, rutrum ac sodales sed, sodales sed urna. Sed egestas et ipsum eu ullamcorper.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Cras auctor augue ligula, a volutpat sem tristique et. Nunc quis orci nec turpis lacinia rhoncus. Nullam id libero placerat, mollis tortor eget, vehicula erat. Etiam ac facilisis leo, sit amet rutrum ipsum. Morbi vitae placerat massa. Nam nisi augue, imperdiet eu aliquam a, accumsan ac nulla. Mauris ipsum mauris, rutrum ac sodales sed, sodales sed urna. Sed egestas et ipsum eu ullamcorper.

## Walkthrough

### Declare the document type

```
<?xml version="1.0" encoding="utf-8" ?>
```

### Start Jelly

```
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
```

### Add the viewport meta tag to match device width.

```
<meta name="viewport" content="width=device-width, initial-scale=1" />
```

### Create a style block for our responsive styles.

In practice this would be in a CSS file to allow for caching.

```
<style>
```

### Just some housekeeping styles to remove some page elements and padding we don't want.

```
*, body{
```

```

        padding: 0;
        margin: 0;
    }

    .body-vertical-spacer{
        display: none;
    }

    #page_timing_div{
        display: none;
    }

```

Create the wrapper in which all content will be displayed. This one has a max width of 920px, which is generally a good cut off for pages to display nicely on all desktops but it is not a rule, only a suggestion. This rule will also center the content.

```

#wrapper {
    width: 96%;
    max-width: 920px;
    margin: auto;
    padding: 2%;
}

```

Create a simple header style for our page.

```

.header{
    width: 100%;
    padding: 20px;
    background-color: lightblue;
}

```

The mock menu uses an unordered list for layout, style it so it appears without any bullets.

```

.menu ul{
    list-style: none;
}

```

Make each menu item "inline" and add some spacing to the right.

```

.menu li{
    font-weight: bold;
    display:inline-block;
    margin-right: 10px;
}

```

This is the key responsive element. In the standard (desktop) view each section should be 29% of the page width with 2% margins. This will come out to 33% for each section so we can easily fit 3 columns.

```

.content .section{
    width: 29%;
    margin: 2%;
    float: left;
}

```

These styles are included to show and hide elements specifically for certain devices. To start .tablet and .phone classes are hidden.

```
.tablet{
    display: none;
}

.phone{
    display: none;
}
```

Here starts the Media Queries. This first one states that for screens with a maximum width of 767 pixels these rules will be ineffect, overriding any default style rules. 767 is specific because 768 is the standard width of tablets, so this will effectively work for all phones. It is possible to get more granular, creating more breakpoints for each device, but this will generally take care of most scenarios.

```
@media only screen
and (max-width : 767px) {
```

Show the `phone` classed elements.

```
.phone{
    display: initial;
}
```

Hide the `tablet` and `desktop` elements.

```
.tablet{
    display: none;
}

.desktop{
    display: none;
}
```

In the phone view we want our content sections to be the full-width of the device so they are actually readable. We could tweak the font-size here too if need be.

```
.content .section{
    width: 96%;
    margin:2%;
}
```

Also we want our menu items to appear one per line now, otherwise the behavior would be unpredictable.

```
.menu li{
    width: 96%;
    margin-right: 0px;
    padding:2%;
}
```

The second media query covers all devices from tablet up to desktop.

```
@media only screen
and (min-width : 768px)
and (max-width : 1024px) {
    .phone{
```

```
        display: none;
    }

    .tablet{
        display:initial;
    }

    .desktop{
        display: none;
    }

    .content .section{
        width: 45%;
        margin:2%;
    }

}

</style>
```

Create the wrapper section, this is the outer most element on the page.

```
<div id="wrapper">
    <div class="header">
        <h2>
            Responsive Example$[SP]
```

Create some device specific elements for the header. This will show which view you are in.

```
        <span class="phone">Phone</span>
        <span class="tablet">Tablet</span>
        <span class="desktop">Desktop</span>
    </h2>
</div>
```

Create our mock menu.

```
<div class="menu">
    <ul>
        <li>Home</li>
        <li>Incident</li>
        <li>Problem</li>
        <li>Change</li>
        <li>Knowledge</li>
    </ul>
</div>
```

Create the content section

```
<div class="content">
```

Create sections, this is where the main content of the page would go. As long as everything is located in a section it will be responsive. You are by no means limited to just `section`, you can have many responsive element types, you just need to make sure that each one acts accordingly for each view.

```
<p class="section">
    Lorem ipsum dolor sit amet, consectetur adipiscing elit. Cras auctor augue ligula, a volutpat sem t
</p>
<p class="section">
```

```
    Lorem ipsum dolor sit amet, consectetur adipiscing elit. Cras auctor augue ligula, a volutpat sem t  
    </p>  
    <p class="section">  
        Lorem ipsum dolor sit amet, consectetur adipiscing elit. Cras auctor augue ligula, a volutpat sem t  
    </p>
```



End the content section

```
</div>
```

End the wrappper section

```
</div>
```

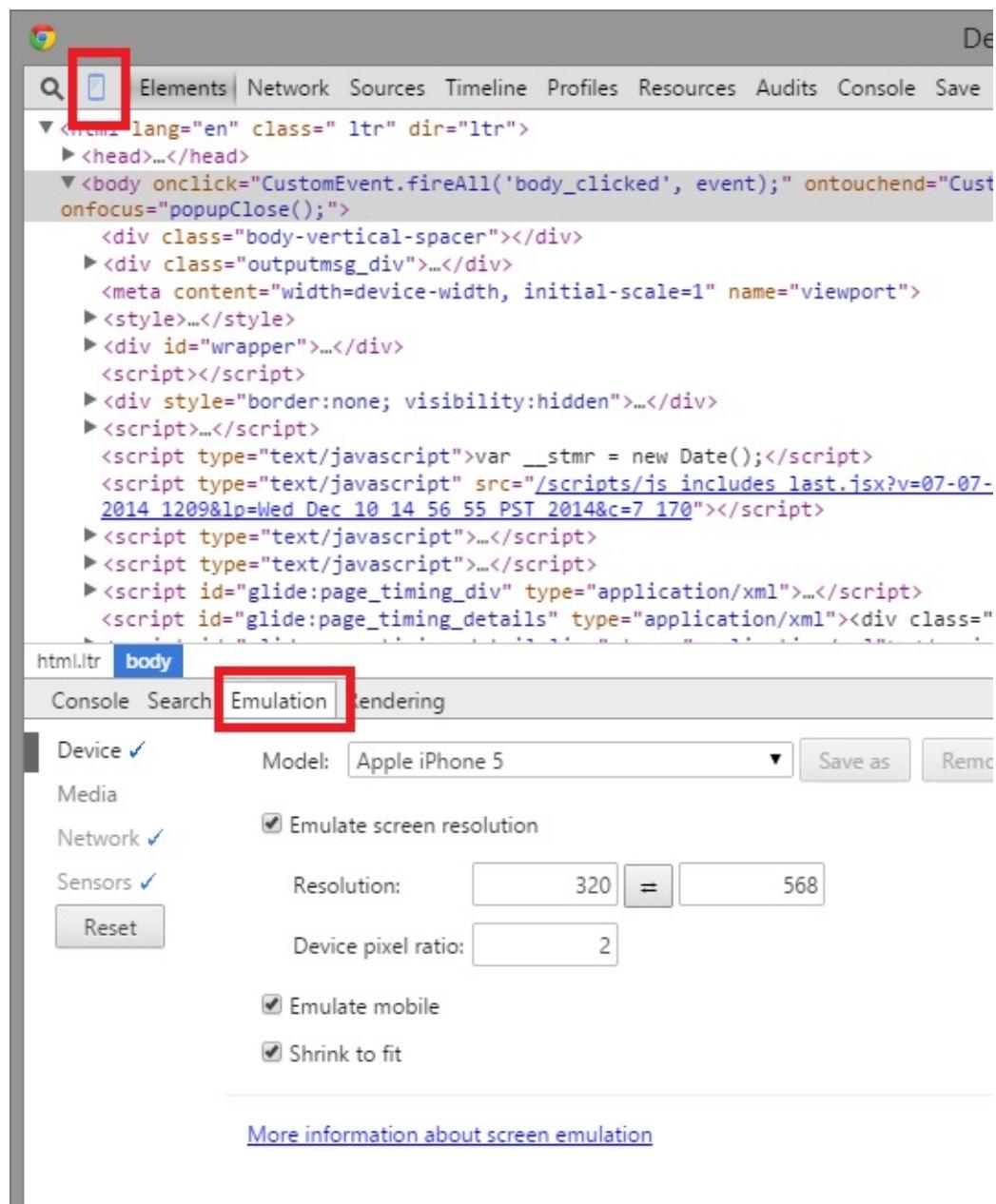
End Jelly

```
</j:jelly>
```

## Notes About Testing Mobile Pages

Using a phone or tablet is a great way to do final testing for a page but for development it makes more sense to emulate the view. Google Chrome has tools built in to do this.

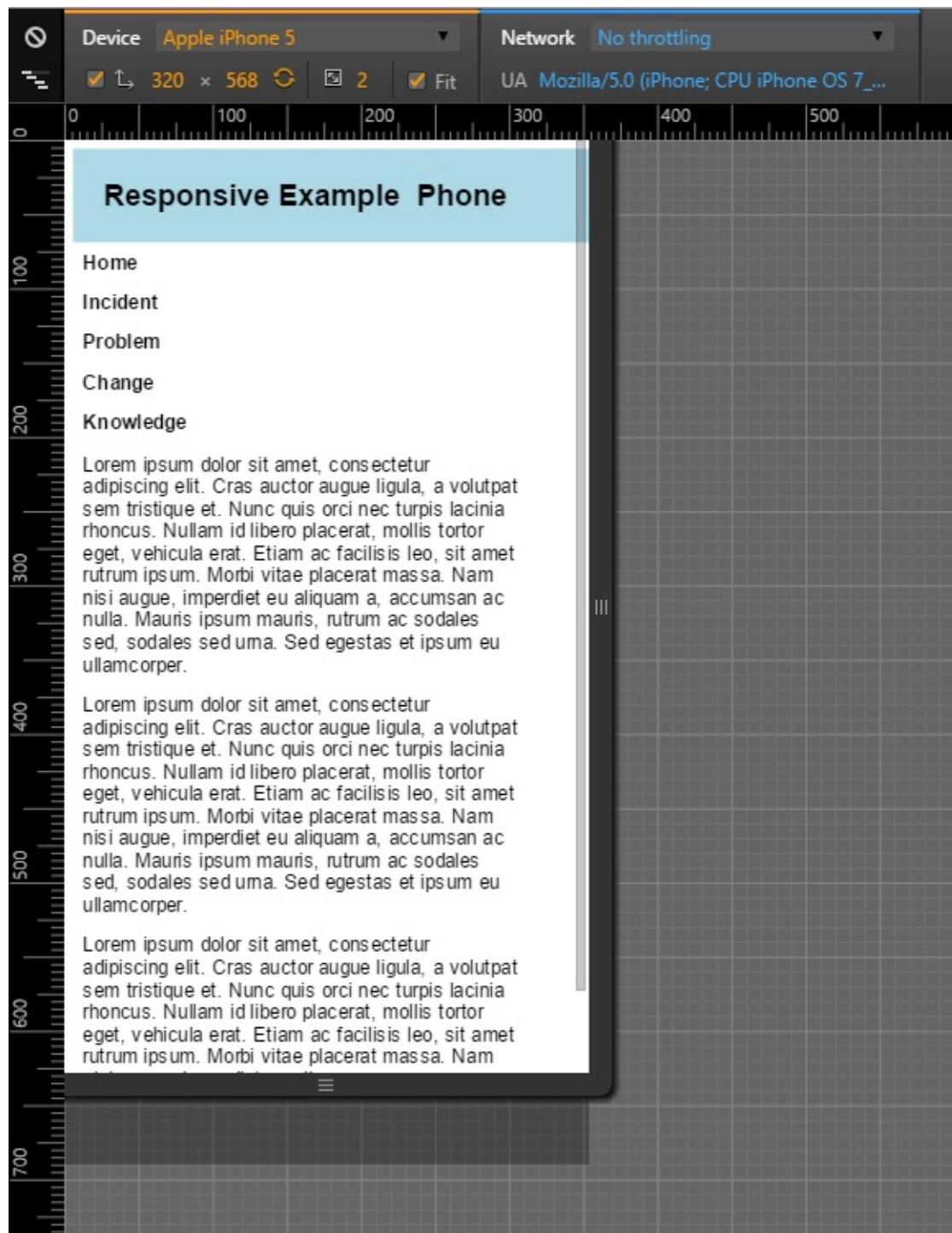
To enable this make sure you have the latest Chrome version. Go to the developer tools (F12) and in the upper left corner, click the Phone icon. This will enable emulation.



An emulation tab also appears on the bottom where you can configure some of the settings for the emulated device.

Within the browser you will see the emulated device and will also have some options to change devices, flip orientation and scale the device.

- | To perform a "pinch" hold shift and click and drag on the page.



In order to get ServiceNow to properly set the device I had to log in and out of the instance.

## Frameworks and Libraries

---

The previous sections give a good set of examples of different types of pages where you can use Jelly, but they are not full examples. In practice there are ways to bootstrap development through the use of UI Frameworks and libraries.

This section will walk through how to use some of those frameworks and libraries on your own and give you some idea on how to port other frameworks to ServiceNow.

# Datatables List

The first library we will look at is Datatables a jQuery plugin that creates functional tables.

In this example we will create a UI Macro, so it is re-usable, and a page that passes content into the Macro.

## UI Macro - datatables

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">

    <link rel="stylesheet" type="text/css" href="//cdn.datatables.net/1.10.4/css/jquery.dataTables.min.css" />

    <div class="table-wrapper">
        <g:insert />
    </div>

    <script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/2.1.3/jquery.min.js" />
    <script>
        jQuery.noConflict();
    </script>
    <script src="//cdn.datatables.net/1.10.4/js/jquery.dataTables.min.js" />

    <script>
        (function($){
            $(document).ready(function(){
                $('#${jvar_table_id}').DataTable();
            });
        })(jQuery)
    </script>
</j:jelly>
```

## Walkthrough

Declare the document type

```
<?xml version="1.0" encoding="utf-8" ?>
```

Start Jelly

```
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
```

Link to the Datatables stylesheet on a CDN

```
<link rel="stylesheet" type="text/css" href="//cdn.datatables.net/1.10.4/css/jquery.dataTables.min.css" />
```

Create a table wrapper div. This could be used to create some styling around the list.

```
<div class="table-wrapper">
```

Insert the body from the tag that called this macro, in this case from the accompanying UI Page.

```
<g:insert />  
</div>
```

Include jQuery from a CDN as it is a dependency of Datatables

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/2.1.3/jquery.min.js" />
```

Set jQuery to noConflict mode to avoid problems with Prototype.

```
<script>  
    jQuery.noConflict();  
</script>
```

Include Datatables from a CDN

```
<script src="//cdn.datatables.net/1.10.4/js/jquery.dataTables.min.js" />
```

Main script starts

```
<script>
```

Using a closure to run the main script. This allows us to set `$` to jQuery without conflicts with Prototype.

```
(function($){
```

When the document is ready execute this function.

```
$(document).ready(function(){
```

Create 'DataTable' plugin for the element with the id `jvar_table_id`. This will be set in the UI Page that calls the function. It allows us to use the plugin multiple times as long as `jvar_table_id` is different each time.

```
    $('#${jvar_table_id'}).DataTable();  
});
```

End the closure and pass jQuery in as a parameter.

```
})(jQuery)
```

End the main script

```
</script>
```

End Jelly

```
</j:jelly>
```

## UI Page

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">

    <j:set var="jvar_table_id" value="incident" />
    <g2:evaluate>
        var incidentRec = new GlideRecord('incident');
        incidentRec.query();
    </g2:evaluate>
    <g:inline template="datatables.xml">
        <table id="${jvar_table_id}" class="table table-striped table-bordered" cellspacing="0" width="100%">
            <thead>
                <tr>
                    <th>Number</th>
                    <th>Short Description</th>
                    <th>Priority</th>
                    <th>Created</th>
                </tr>
            </thead>
            <tbody>
                <j2:while test="${[incidentRec.next()]}>
                    <tr>
                        <td>${[incidentRec.number.getDisplayValue()]}</td>
                        <td>${[incidentRec.short_description.getDisplayValue()]}</td>
                        <td>${[incidentRec.priority.getDisplayValue()]}</td>
                        <td>${[incidentRec.sys_created_on.getDisplayValue()]}</td>
                    </tr>
                </j2:while>
            </tbody>
        </table>
    </g:inline>
</j:jelly>
```

## Output

Show <select>10</select> entries <input placeholder="Search:" type="text"/>			
Number	Short Description	Priority	Created
INC0000001	<j2:set var="jvar_insecure" value="[\$'Uh oh!']" />	1 - Critical	2012-12-07 10:24:13
INC0000002	Can't get to network file shares	1 - Critical	2012-12-07 14:30:06
INC0000003	Wireless access not available on floor 3	4 - Low	2012-12-22 06:41:46
INC0000004	Forgot email password	4 - Low	2012-12-07 14:34:12
INC0000005	CPU load high for over 10 minutes	2 - High	2012-12-22 07:14:01
INC0000006	Hang when trying to print VISIO document	4 - Low	2012-12-04 12:42:01
INC0000007	Need access to sales db for the west	4 - Low	2012-12-04 12:43:10
INC0000008	Printer in my office is out of toner	4 - Low	2012-12-04 12:43:58
INC0000009	Reset my password	1 - Critical	2014-05-21 15:51:33
INC0000010	Need Oracle 10GR2 installed	2 - High	2014-05-21 15:54:41

Showing 1 to 10 of 54 entries

Previous 1 2 3 4 5 6 Next  
(⌚ Response time(ms)): , network: 0, server: 75, browser: 1103

## Walkthrough

Declare the document type

```
<?xml version="1.0" encoding="utf-8" ?>
```

## Start Jelly

```
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
```

Set up a variable `jvar_table_id` to store the ID we want to use for our table. It will be used setting up the table and in the included script to enable the plugin (above).

```
<j:set var="jvar_table_id" value="incident" />
```

Set up a simple query to get some results for our table.

```
<g2:evaluate>
    var incidentRec = new GlideRecord('incident');
    incidentRec.query();
</g2:evaluate>
```

Inline the `datatables` UI Macro. The body of this tag will be inserted in the appropriate place in the UI Macro's output. This is the `g:insert` tag in the UI Macro.

```
<g:inline template="datatables.xml">
```

Create a table with the ID specified in `jvar_table_id` and add some styling to it.

```
<table id="${jvar_table_id}" class="table table-striped table-bordered" cellspacing="0" width="100%">
```

Datatables requires a well-formed table that includes a `thead` and `tbody`.

```
<thead>
```

Add our table headers for the fields we want.

```
<tr>
    <th>Number</th>
    <th>Short Description</th>
    <th>Priority</th>
    <th>Created</th>
</tr>
</thead>
```

Create a `tbody` for Datatables.

```
<tbody>
```

Iterate over the returned records creating a row for each record.

```
<j2:while test="${incidentRec.next()}">
    <tr>
        <td>${incidentRec.number.getDisplayValue()}</td>
```

```
<td>${incidentRec.short_description.getDisplayValue()}</td>
<td>${incidentRec.priority.getDisplayValue()}</td>
<td>${incidentRec.sys_created_on.getDisplayValue()}</td>
</tr>
```

End the iteration and wrap up the tbody and table tags.

```
</j2:while>
</tbody>
</table>
```

End the inline.

```
</g:inline>
```

End Jelly

```
</j:jelly>
```

## Notes on implementing Datatables

Datatables is a very robust plug-in with many configurable options and extensions. Visit the [Datatables Website](#) to get more information on configurable options and instructions on how to include plugins and styling options.

# Twitter Bootstrap

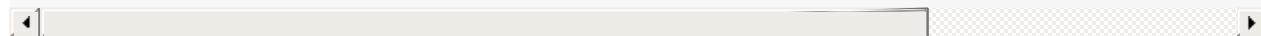
Twitter Bootstrap is an immensely popular UI Framework and it's big enough that it is worth covering it here. Generally speaking, HTML that has Bootstrap applied on it will look better by default, since many of the styles included apply directly to HTML tags. The remainder of the UI Enhancements are applied through CSS classes.

This example shows how to port over examples but does not cover each Bootstrap class. For that visit [getbootstrap.com](http://getbootstrap.com).

## UI Page

```
<?xml version="1.0" encoding="utf-8" ?>
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
<head>
    <meta charset="utf-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge,chrome=1" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />

    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/twitter-bootstrap/3.3.1/css/bootstrap.min.css" />
    <style>
        body {
            padding-top: 50px;
            padding-bottom: 20px;
        }
    </style>
    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/twitter-bootstrap/3.3.1/css/bootstrap-theme.min.css" />
    <script src="//cdnjs.cloudflare.com/ajax/libs/modernizr/2.8.3/modernizr.min.js"></script>
</head>
<body>
<nav class="navbar navbar-inverse navbar-fixed-top" role="navigation">
    <div class="container">
        <div class="navbar-header">
            <button type="button" class="navbar-toggle collapsed" data-toggle="collapse" data-target="#navbar" aria-expanded="false" aria-controls="navbar">
                <span class="sr-only">Toggle navigation</span>
                <span class="icon-bar"></span>
                <span class="icon-bar"></span>
                <span class="icon-bar"></span>
            </button>
            <a class="navbar-brand" href="#">Project name</a>
        </div>
        <div id="navbar" class="navbar-collapse collapse">
            <form class="navbar-form navbar-right" role="form">
                <div class="form-group">
```



Like `meta` and `link` Slashes must be added to `inputs`

```
        <input type="text" placeholder="Email" class="form-control" />
    </div>
    <div class="form-group">
        <input type="password" placeholder="Password" class="form-control" />
    </div>
    <button type="submit" class="btn btn-success">Sign in</button>
</form>
</div><!-- .navbar-collapse -->
</div>
</nav>

<!-- Main jumbotron for a primary marketing message or call to action -->
<div class="jumbotron">
    <div class="container">
        <h1>Hello, world!</h1>
        <p>This is a template for a simple marketing or informational website. It includes a large callout called a jumbotron for important messaging.</p>
        <p><a class="btn btn-primary btn-lg" href="#" role="button">Learn more &gt;</a></p>
    </div>
```

```

</div>

<div class="container">
    <!-- Example row of columns -->
    <div class="row">
        <div class="col-md-4">
            <h2>Heading</h2>
            <p>Donec id elit non mi porta gravida at eget metus. Fusce dapibus, tellus ac cursus commodo, tortor mauris cor
            <p><a class="btn btn-default" href="#" role="button">View details $[AMP]raquo;</a></p>
        </div>
        <div class="col-md-4">
            <h2>Heading</h2>
            <p>Donec id elit non mi porta gravida at eget metus. Fusce dapibus, tellus ac cursus commodo, tortor mauris cor
            <p><a class="btn btn-default" href="#" role="button">View details $[AMP]raquo;</a></p>
        </div>
        <div class="col-md-4">
            <h2>Heading</h2>
            <p>Donec sed odio dui. Cras justo odio, dapibus ac facilisis in, egestas eget quam. Vestibulum id ligula porta
            <p><a class="btn btn-default" href="#" role="button">View details $[AMP]raquo;</a></p>
        </div>
    </div>

    <hr />

    <footer>
        <p>$[AMP]copy; Company 2014</p>
    </footer>
</div> <!-- /container -->
<script src="//cdnjs.cloudflare.com/ajax/libs/jquery/2.1.3/jquery.min.js"></script>
<script>
    jQuery.noConflict();
</script>

<script src="https://cdnjs.cloudflare.com/ajax/libs/twitter-bootstrap/3.3.1/js/bootstrap.min.js"></script>

</body>

</j:jelly>

```

## Output

Project name

Email

Password

Sign in

# Hello, world!

This is a template for a simple marketing or informational website. It includes a large callout called a jumbotron and three supporting pieces of content. Use it as a starting point to create something more unique.

[Learn more »](#)

## Heading

Donec id elit non mi porta gravida at eget metus. Fusce dapibus, tellus ac cursus commodo, tortor mauris condimentum nibh, ut fermentum massa justo sit amet risus. Etiam porta sem malesuada magna mollis euismod. Donec sed odio dui.

[View details »](#)

## Heading

Donec id elit non mi porta gravida at eget metus. Fusce dapibus, tellus ac cursus commodo, tortor mauris condimentum nibh, ut fermentum massa justo sit amet risus. Etiam porta sem malesuada magna mollis euismod. Donec sed odio dui.

[View details »](#)

## Heading

Donec sed odio dui. Cras justo odio, dapibus ac facilisis in, egestas eget quam. Vestibulum id ligula porta felis euismod semper. Fusce dapibus, tellus ac cursus commodo, tortor mauris condimentum nibh, ut fermentum massa justo sit amet risus.

[View details »](#)

## Walkthrough

Declare the document type

```
<?xml version="1.0" encoding="utf-8" ?>
```

Start Jelly

```
<j:jelly trim="false" xmlns:j="jelly:core" xmlns:g="glide" xmlns:j2="null" xmlns:g2="null">
```

In the original example there was a `DOCTYPE` declaration which is not necessary in ServiceNow since it is done before the UI Page is returned. It has been removed here.

This `head` tag won't break anything but it doesn't serve any purpose aside from keeping the page organized. The returned document will already have a `head` tag defined and only one is permitted per page. The elements in here will be moved automatically by the browser.

```
<head>
```

Declare the character set in the response document.

Many times the trailing slash is left off of `meta` and `link` tags, you must add this back in for it to be valid XML.

```
<meta charset="utf-8" />
```

Supposedly this will force IE to use the latest rendering engine, thereby eliminating any confusion with compatibility. This is questionable here since this `meta` tag is actually returned in the body and may occur too late for the browser to even honor this tag.

```
<meta http-equiv="X-UA-Compatible" content="IE=edge,chrome=1" />
```

Add a viewport meta tag so mobile devices display properly.

```
<meta name="viewport" content="width=device-width, initial-scale=1" />
```

Include the base Twitter Bootstrap Style Sheet from CDN

```
<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/twitter-bootstrap/3.3.1/css/bootstrap.min.css" />
```

Add some page specific styling.

If styles go beyond this, they should be moved to their own Style Sheet record and included with a `link` to help caching and increase the readability of the page.

```
<style>
  body {
    padding-top: 50px;
    padding-bottom: 20px;
```

```

        }
    </style>

```

Include the Themed CSS style sheet for Bootstrap.

```

<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/twitter-bootstrap/3.3.1/css/bootstrap-theme.n

```

[Modernizr](#) adds feature detection to a page and allows you to use those results to control what features are enabled for users.

```

<script src="//cdnjs.cloudflare.com/ajax/libs/modernizr/2.8.3/modernizr.min.js"></script>
</head>

```

Start the main content.

```

<body>

```

The `nav` tag creates the header bar. Look back at the Portal example to see how this can be abstracted into its own Macro.

```

<nav class="navbar navbar-inverse navbar-fixed-top" role="navigation">
    <div class="container">
        <div class="navbar-header">
            <button type="button" class="navbar-toggle collapsed" data-toggle="collapse" data-target="#navbar" aria-expanded="false" aria-controls="navbar">
                <span class="sr-only">Toggle navigation</span>
                <span class="icon-bar"></span>
                <span class="icon-bar"></span>
                <span class="icon-bar"></span>
            </button>
            <a class="navbar-brand" href="#">Project name</a>
        </div>
        <div id="navbar" class="navbar-collapse collapse">
            <form class="navbar-form navbar-right" role="form">
                <div class="form-group">
                    <input type="text" placeholder="Email" class="form-control" />
                </div>
                <div class="form-group">
                    <input type="password" placeholder="Password" class="form-control" />
                </div>
                <button type="submit" class="btn btn-success">Sign in</button>
            </form>
        </div><!-- .navbar-collapse -->
    </div>
</nav>

```

Some more content in the form of a "Jumbotron" the huge main section.

```

<!-- Main jumbotron for a primary marketing message or call to action --&gt;
&lt;div class="jumbotron"&gt;
    &lt;div class="container"&gt;
        &lt;h1&gt;Hello, world!&lt;/h1&gt;
        &lt;p&gt;This is a template for a simple marketing or informational website. It includes a large callout called a jumbo
        &lt;p&gt;&lt;a class="btn btn-primary btn-lg" href="#" role="button"&gt;Learn more &amp;gt;&lt;/a&gt;&lt;/p&gt;
    &lt;/div&gt;
&lt;/div&gt;
</pre>

```

Start of the actual content section. Containers are the outer most element in bootstrap, they can contain rows, and each row contains columns.

Bootstrap uses a 12 column layout so you can add elements in each row and using `col-` classes you can control how they are laid out. The second part of the `col-` class is the view size. This is used in responsive layouts.

```
<div class="container">
  <!-- Example row of columns -->
  <div class="row">
```

In this case there is only one `col` style applied so this element will always be 4 columns wide but you may apply multiple `col` styles to change it's width depending on the device width. `class="col-sm-12 col-md-6 col-lg-4"` would create an element that is the full width on small devices, like phones, half the width on devices like tablets, and 1/3rd on desktop or larger screen devices.

```
<div class="col-md-4">
  <h2>Heading</h2>
  <p>Donec id elit non mi porta gravida at eget metus. Fusce dapibus, tellus ac cursus commodo, tortor mauris cor
  <p><a class="btn btn-default" href="#" role="button">View details $[AMP]raquo;</a></p>
</div>
<div class="col-md-4">
  <h2>Heading</h2>
  <p>Donec id elit non mi porta gravida at eget metus. Fusce dapibus, tellus ac cursus commodo, tortor mauris cor
  <p><a class="btn btn-default" href="#" role="button">View details $[AMP]raquo;</a></p>
</div>
<div class="col-md-4">
  <h2>Heading</h2>
  <p>Donec sed odio dui. Cras justo odio, dapibus ac facilisis in, egestas eget quam. Vestibulum id ligula porta
  <p><a class="btn btn-default" href="#" role="button">View details $[AMP]raquo;</a></p>
</div>
</div>
```

`hr` also needs a slash

```
<hr />
```

Footer section.

```
<footer>
```

Be sure when porting over examples to replace & with \$[AMP]

```
<p>$[AMP]copy; Company 2014</p>
</footer>
```

End the main container div.

```
</div> <!-- /container -->
```

Include jQuery from a CDN

```
<script src="//cdnjs.cloudflare.com/ajax/libs/jquery/2.1.3/jquery.min.js"></script>
```

Run noConflict mode, you should know why by now!!!

```
<script>
  jQuery.noConflict();
</script>
```

Include Bootstrap JS from CDN

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/twitter-bootstrap/3.3.1/js/bootstrap.min.js"></script>
```

End the `body` and Jelly

```
</body>
</j:jelly>
```

## Notes on Twitter Bootstrap

Twitter Bootstrap can be a great way to add a standard look and feel to your custom pages. The only issues I have run into is when trying to use Bootstrap on a standard ServiceNow form, then the styles conflict and the result can be messy and confusing to the end user. It's best to only use this on custom UI Pages.

## Suggested Exercises

1. Expand this example by using Jelly to create ServiceNow based content: a form or list view.
2. Combine Twitter Bootstrap with Datatables to create a Boostrap Themed Datatables list.
3. Using this example and the Portal example, create a re-usable Twitter Bootstrap layout that you can pass content to.

## Index

---

## action\_filter

*No Description*

**Library** Glide

### ⊕ Parameters

*Unknown*

### ⊖ Returns

*Unknown*

## Example

No documented example

## activity

Gets the activity for a record

**Library** Glide

### ⊕ Parameters

Attribute	Description	Type	Required	Default
sys_id	The sys_id of the record	String	Yes	
fields	List of fields to get activity for	String	Yes	
table	The source table	String	Yes	
type	Specific type of activity to get	String		
var	The variable to store the activity in	String	Yes	

### ⊖ Returns

Return	Description	Type
var	Array of activity records	Array

## Example

```
<g:activity sys_id="9c573169c611228700193229ffff72400" fields="" table="incident" type="changes" var="jvar_activity">
<j:forEach items="${jvar_activity}" var="jvar_history">
    Created On: ${jvar_history.getValue('sys_created_on')}
    <g:activity type="header" records="${jvar_history}" var="jvar_field_header"/>
        Header: ${jvar_field_header}
        <br/>
    <g:activity type="labels" records="${jvar_history}" var="jvar_field_labels"/>
    <strong>Fields</strong>
    <ul>
        <j:forEach items="${jvar_field_labels}" var="jvar_h">
```

```

<li>${jvar_h}</li>
</j:forEach>
</ul>

<strong>Updated Fields</strong>
<ul>

<j:forEach items="${jvar_history}" var="jvar_h">
<li>Source Table: ${jvar_h.getTableName()} - Field Name: ${jvar_h.getValue('fieldname')} - New Value: ${jvar_h.
</j:forEach>
</ul>

<hr/>

</j:forEach>
```



## annotation

Adds an annotation to a form

**Library** Glide

### ⊕ Parameters

Attribute	Description	Type	Required	Default
text	Text to output	String	Yes	<i>None</i>
annotation_type	Type of annotation to output	String		

### ⊕ Returns

*None*

## Example

```

<g2:no_escape>
  <g2:annotation text="${jvar_sc_category_infomsg}" annotation_type="Section Details"/>
</g2:no_escape>
```

## application\_selector

Outputs the Application Selector if Applications are enabled on the instance

**Library** Glide

### ⊕ Parameters

*None*

### ⊕ Returns

*None*

## Example

```
<g:application_selector />
```

## arg

Passes an argument to an invoke element

**Libary** Jelly

### ⊕ Parameters

Attribute	Description	Type	Required	Default
value	Value to pass to the outer <code>invoke</code>	String	Yes	

### ⊕ Returns

*None*

### Example

```
<j:arg value="6816f79cc0a8016401c5a33be04be441"/>
```

## attachment\_entry

Ouputs an attachment ServiceNow widget

**Libary** Glide

### ⊕ Parameters

Attribute	Description	Type	Required	Default
show_rename	Controls whether to show rename	Boolean		

### ⊕ Returns

*None*

### Example

```
<g2:attachment_entry show_rename="true" />
```

## attachment\_list

Gets a list of attachments for a given record

**Libary** Glide

### ⊕ Parameters

Attribute	Description	Type	Required	Default
table	Table of record attachments are for	String	Yes	

sys_id	Sys Id of record attachments are for	String	Yes	
var	Variable to store results in	String	No	sys_attachment
Body	Body of this tag has access to the attachments variable	Jelly/HTML/Text	No	

### ⊕ Returns

Return	Description	Type
var	Array of attachments	Array

### Example

```
<g2:attachment_list table="incident" sys_id="9c573169c611228700193229fff72400">
  <g2:for_each_record file_variable="sys_attachment" var="attachment">
    <g2:attachment_entry show_rename="true" />
  </g2:for_each_record>
</g2:attachment_list>
```

## breakpoint

Stops execution of a script and outputs a variable to the system log

### Library Glide

### ⊕ Parameters

Attribute	Description	Type	Required	Default
var	The name of the Jelly variable to output	String	Yes	
id	A unique identifier to output in the log	String	Yes	

### ⊕ Returns

None

### Example

```
<j:set var="jvar_test_var" value="FooBar" />
<g:breakpoint var="jvar_test_var" id="My Breakpoint" />
```

## calendar

Generates a list of events for a given context

### Library Glide

### ⊕ Parameters

Attribute	Description	Type	Required	Default
view	Calendar view to use	String	Yes	
year	Year to pull events from	String	Yes	

month	Month to pull events from	String	Yes	
day	Day to pull events from	String	Yes	

### ⊕ Returns

*None*

### Example

```
<j2:set var="sysparm_view" value="year" />
<j2:set var="sysparm_calview" value="year" />

<j2:set var="sysparm_year" value="2014" />
<j2:set var="sysparm_month" value="1" />
<j2:set var="sysparm_day" value="1" />
<j2:set var="sysparm_table" value="incident" />
<j2:set var="sysparm_query" value="active=true" />
<j2:set var="sysparm_cal_field" value="sys_updated_on" />

<g2:calendar view="[$[sysparm_calview]" year="[$[sysparm_year]" month="[$[sysparm_month]" day="[$[sysparm_day]"/>

<j2:if test="[$[sysparm_calview == 'month' || sysparm_calview == '' || sysparm_calview == null]">
    <j2:set var="jvar_month" value="[$[jvar_calendar.firstMonth]" />
    <g2:inline template="calendar_month.xml"/>
</j2:if>

<j2:if test="[$[sysparm_calview == 'week']">
    <g2:inline template="calendar_week.xml"/>
</j2:if>

<j2:if test="[$[sysparm_calview == 'day']">
    <g2:inline template="calendar_day.xml"/>
</j2:if>

<j2:if test="[$[sysparm_calview == 'year']">
    <g2:inline template="calendar_year.xml"/>
</j2:if>
```

## calendar\_event\_id

*No Description*

### Library Glide

### ⊕ Parameters

*Unknown*

### ⊕ Returns

*None*

### Example

No documented example

## call

Executes a given Jelly file as a function passing in parameters

**Library Glide****⊕ Parameters**

Attribute	Description	Type	Required	Default
function	Name of the function (UI Macro) to call	String	Yes	

**⊕ Returns***None***Example**

```
<j:set var="jvar_message" value="Hello World" />
Message Outside Before: ${jvar_message}<br />
<g:call function="my_function.xml" /><br />
Message Outside After: ${jvar_message}<br />
```

**case**

Nested tag within a `switch` block which conditionally runs nested code if it's value matches the given variable of the parent switch

**Library Jelly****⊕ Parameters**

Attribute	Description	Type	Required	Default
value	The value to match in order to execute the body	String	Yes	
Body	Contents to conditionally execute	Jelly/HTML/Text	Yes	

**⊕ Returns***None***Example**

```
<j:case value="Hello">
    Hello Again.
</j:case>
```

**catch**

Catches a thrown exception and it may work in ServiceNow, however it seems when an exception is thrown the page will assert that exception anyways, so this is not usable in UI Pages or Macros.

**Library Jelly****⊕ Parameters***Unknown*

## ⊕ Returns

*None*

## Example

```
*None*
```

## **child\_tables**

Generates a list of child tables for a given root table

Library Glide

## ⊕ Parameters

Attribute	Description	Type	Required	Default
root	The root table to list child tables from	String	Yes	
skipRoot	Controls whether or not to include the root in the list	Boolean	Yes	
includeDefault	Includes "Default" in the list	Boolean	Yes	
skipTables	List of tables to exclude	String	Yes	
choiceType	Option to add a "None" option if set to "1"	String	No	False

## ⊕ Returns

*None*

## Example

```
<select>
  <g:child_tables root="task" skipRoot="true" includeDefault="true" skipTables="incident" choiceType="1" />
</select>
```

## **choice\_list**

Gets values for a field on a table and stores them in a variable

Library Glide

## ⊕ Parameters

Attribute	Description	Type	Required	Default
table	Table to pull choice values from	String	Yes	
column	Field to pull choice values from	String	Yes	
value	Selected value	String	No	

## ⊕ Returns

Return	Description	Type
jvar_choices	List of choices	List

**Example**

```
<g:choice_list table="incident" column="priority" value="3" />
<select>
<j:forEach var="jvar_choice" items="${jvar_choices}">
<j:if test="${jvar_choice.selected=='true'}">
    <option value="${jvar_choice.value}" selected="true">${jvar_choice.label}</option>
</j:if>
<j:if test="${jvar_choice.selected=='false'}">
    <option value="${jvar_choice.value}">${jvar_choice.label}</option>
</j:if>
</j:forEach>
</select>
```

**choose**

Evaluates nested `when` blocks as a single set of conditions and runs blocks if their test condition passes

**Library Jelly**

⊕ **Parameters** | Attribute | Description | Type | Required | Default | -- | -- | -- | -- | -- | Body | Contains a set of conditional blocks | Jelly/HTML/Text | Yes | |

*None*

**⊕ Returns**

*None*

**Example**

```
<j:set var="jvar_showMessage" value="Hello" />

<j:choose>
<j:when test="${jvar_showMessage=='Hello'}">
    Hello
</j:when>
<j:when test="${jvar_showMessage=='Goodbye'}">
    Goodbye
</j:when>
<j:when test="${jvar_showMessage=='Hello'}">
    This will never be displayed.
</j:when>
<j:otherwise>
    Not Sure
</j:otherwise>
</j:choose>
```

**ci\_relationship\_manage**

Outputs the CI Relationship Manager form

**Library Glide****⊕ Parameters**

None

#### ⊕ Returns

None

#### Example

```
<g:ci_relationship_manage/>
```

## client\_script

Outputs different type of Client Side scripts

Library Glide

#### ⊕ Parameters

Attribute	Description	Type	Required	Default
type	Can be specified to output certain types of scripts (e.g. "user")	String	No	
tableName	Can be used with <code>type</code> or <code>file</code> to output table client scripts	Reference	No	
type	<i>Unknown</i>	String	No	
file	Outputs tables client scripts	Reference	No	

#### ⊕ Returns

None

#### Example

```
<g:evaluate var="incident" copyToRhino="true" object="true">
    var gr = new GlideRecord('incident');
    gr.query();
    gr.next();
    gr;
</g:evaluate>

<j:set var="ref" value="incident" />

<g:client_script type="user"/>
<g:client_script tableName="${ref}" type="phase1"/>
<g2:client_script tableName="${ref}" file="${[${ref}]}" />
```

## cms\_menu

Outputs a CMS menu

Library Glide

#### ⊕ Parameters

Attribute	Description	Type	Required	Default
-----------	-------------	------	----------	---------

menu_sysid	Sys Id of the Menu to output	String	Yes	
------------	------------------------------	--------	-----	--

### ⊕ Returns

*None*

### Example

```
<g:cms_menu menu_sysid='577043b4c0a8016b003cbacf363d38f3' />
```

## cms\_menu\_functions

Generates a function Server Side used to check if a user can read a menu

Library Glide

### ⊕ Parameters

*None*

### ⊕ Returns

*None*

### Example

```
*No documented example*
```

## cms\_menu\_set\_url\_and\_target

Generate the URL and target for a particular menu item

Library Glide

### ⊕ Parameters

Attribute	Description	Type	Required	Default
link_sysid	Sys Id of a CMS Menu Item	String	Yes	

### ⊕ Returns

Return	Description	Type
jvar_link_url	URL of the Menu Item	String
jvar_link_target	Target of the URL ("gsft_main","_blank")	String

### Example

```
*No documented example*
```

## columnoptions

Generates a list of searchable fields for a table

Library Glide

### ⊕ Parameters

Attribute	Description	Type	Required	Default
ref	The table to pull fields from	Reference	Yes	

### ⊕ Returns

*None*

### Example

```
<g:evaluate var="incident" copyToRhino="true" object="true">
    var gr = new GlideRecord('incident');
    gr.query();
    gr.next();
    gr;
</g:evaluate>
<select>
    <g:columnoptions ref="incident" />
</select>
```

## comment

Outputs an HTML Comment

Library Glide

### ⊕ Parameters

Attribute	Description	Type	Required	Default
Body	Text of the comment to output	String	Yes	

### ⊕ Returns

*None*

### Example

```
<g:comment>
    This comment will not be returned.
</g:comment>
<g2:comment>
    This will be returned to the user.
</g2:comment>
```

## content\_block

Renders a CMS Content Block

**Library Glide****⊕ Parameters**

Attribute	Description	Type	Required	Default
type	The type of CMS Block to output	String	Yes	
id	The Sys Id of the Content Block to output	String	Yes	

**⊕ Returns***None***Example**

```
<g:content_block type="content_block_menu" id="577043b4c0a8016b003cbacf363d38f3"/>
```

**content\_summarizer**

Used to generate a list item for a particular CMS Content Type

**Library Glide****⊕ Parameters**

Attribute	Description	Type	Required	Default
content	Reference to the content to output	Reference	Yes	

**⊕ Returns***None***Example**

```
<div style="font-size:larger;">
    <div class="content_list_title" style="width:98%;">${jvar_title}</div>
    <ul>
        <g:for_each_record file="${current}" max="${jvar_max_entries}">
            <li style="padding-top:4px;">
                <g:content_summarizer content="${current}" />
            </li>
        </g:for_each_record>
    </ul>
</div>
```

**copy\_to\_p2**

Copies a variable and sets it in Phase 2

**Library Glide****⊕ Parameters**

Attribute	Description	Type	Required	Default
-----------	-------------	------	----------	---------

var	Variable to set in Phase 2	String	Yes	
Body	Encoded value to set the variable to	String	Yes	

### ⊕ Returns

None

### Example

```
<g2:copy_to_p2 var="jvar_message">H4sIvJyCxWAKJEheT83IKi10Li1BSF4JKizLx0PQB3a91mH==</g2:copy_to_p2>
```

## currency\_format

Outputs a formatted value for a value in the users currency format

### Library Glide

### ⊕ Parameters

Attribute	Description	Type	Required	Default
value	Value to format	Decimal	Yes	

### ⊕ Returns

None

### Example

```
<g:currency_format value="100.99" />
```

## dashboard

Outputs dashboard elements for a given dashboard

### Library Glide

### ⊕ Parameters

Attribute	Description	Type	Required	Default
board	Name of the board to output	String	Yes	

### ⊕ Returns

None

### Example

```
<table>
  <g:dashboard board="Overview"/>
</table>
```

## decorations

Gets the list of UI Decorations

**Library** Glide

### ⊕ Parameters

Attribute	Description	Type	Required	Default
var	Variable to store the result in	String	Yes	

### ⊕ Returns

Return	Description	Type
var	List of decorations	List

### Example

```
<g:decorations var="jvar_decorations" />
```

## default

Nested tag within a `switch` block which runs nested code if no sibling cases are true

**Library** Jelly

### ⊕ Parameters

Attribute	Description	Type	Required	Default
Body	Conditional block to execute	Jelly/HTML/Text	Yes	

### ⊕ Returns

*None*

### Example

```
<j:default>
    Not Sure
</j:default>
```

## dialog\_button

Outputs a form Button

**Library** Glide

### ⊕ Parameters

Attribute	Description	Type	Required	Default
onclick	JavaScript function to execute	String	Yes	
name	Name of the element	String	Yes	
id	HTML ID of the element	String	Yes	
Body	The content of the button	Jelly/HTML/Text	Yes	

### ⊕ Returns

None

### Example

```
<g:dialog_button onclick="console.log('Dialog Button')" name="ok_button" id="map_button">${gs.getMessage('OK')}</g:dia]
```

## dialog\_button\_ok

Outputs a form Button

Library Glide

### ⊕ Parameters

Attribute	Description	Type	Required	Default
ok	JavaScript to execute when user clicks button	String	Yes	
ok_type	Sets the Type of button for the OK action ("button")	String	Yes	

### ⊕ Returns

None

### Example

```
<g:dialog_button_ok ok="console.log('Dialog Button OK')" ok_type="button" />
```

## dialog\_buttons\_ok\_cancel

Outputs a set of Form Buttons

Library Glide

### ⊕ Parameters

Attribute	Description	Type	Required	Default
ok	JavaScript to execute when user clicks OK	String	Yes	
cancel	JavaScript to execute when user clicks Cancel	String	Yes	
cancel_type	Sets the type of button for the Cancel action ("button")	String	Yes	

ok_id	HTML Element ID of the OK button	String	Yes	
-------	----------------------------------	--------	-----	--

### ⊕ Returns

*None*

### Example

```
<g:dialog_buttons_ok_cancel
ok="console.log('Dialog Button OK')"
cancel="console.log('Dialog Button Cancel')"
ok_id="ok_button"
cancel_type="button" />
```

## dialog\_buttons\_save\_cancel\_discard

Outputs a set of Form Buttons

### Library Glide

### ⊕ Parameters

Attribute	Description	Type	Required	Default
discard	JavaScript to execute when user clicks discard	String	Yes	
discard_type	Sets the type of button for the Discard action	String		
cancel	JavaScript to execute when user clicks cancel	String	Yes	
cancel_type	Sets the type of button for the Cancel action	String	Yes	
save	JavaScript to execute when user clicks save	String	Yes	
save_type	Sets the type of button for the Save action	String	Yes	

### ⊕ Returns

*None*

### Example

```
<g:dialog_buttons_save_cancel_discard
discard="console.log('Dialog Button Discard')"
discard_type="button"
cancel="console.log('Dialog Button Cancel')"
cancel_type="button"
save="console.log('Dialog Button Save')"
save_type="button"/>
```

## doctype

Sets a returned documents doctype

### Library Glide

### ⊕ Parameters

Attribute	Description	Type	Required	Default
name	Name of the doctype to output	String	Yes	

⊕ **Returns**

*None*

**Example**

```
<g2:doctype name="html" />
```

## domain\_reference\_picker

Outputs the Domain Reference Picker

**Library** Glide

⊕ **Parameters**

*None*

⊕ **Returns**

*None*

**Example**

```
<g:domain_reference_picker />
```

## domain\_select

Outputs the Domain Drop Down

**Library** Glide

⊕ **Parameters**

*None*

⊕ **Returns**

*None*

**Example**

```
<g:domain_select />
```

## element

Outputs a field

**Library Glide****⊕ Parameters**

Attribute	Description	Type	Required	Default
ref	Reference of the element to output	Reference	Yes	

**⊕ Returns***None***Example**

```
<g:evaluate var="incident" copyToRhino="true" object="true">
    var gr = new GlideRecord('incident');
    gr.query();
    gr.next();
    gr;
</g:evaluate>

<g:element ref="incident.short_description" />
<g:element ref="incident.priority" />
```

**element\_list**

May output a list of fields in a section

**Library Glide****⊕ Parameters**

Attribute	Description	Type	Required	Default
section_id	ID of the section to output	String	Yes	
table	Name of the table to output a section for	String	Yes	

**⊕ Returns***None***Example**

```
<g:evaluate>
    var incidentRec = new GlideRecord('incident');
    incidentRec.query();
    incidentRec.next();
    incidentRec.next();
    incidentRec;
</g:evaluate>

<j:set var="ref" value="incident" />

<g:element_list section_id="991f88d20a00064127420bc37824d385" table="incident">
</g:element_list>

$[jvar_p2flow_all_visible_fields]
```

## elevate\_privilege

Outputs the Elevate Privilege Button

**Library** Glide

### ⊕ Parameters

*None*

### ⊖ Returns

*None*

#### Example

```
<g:elevate_privilege />
```

## elevated\_role\_dialog

Outputs the Elevate Privilege Dialog

**Library** Glide

### ⊕ Parameters

*None*

### ⊖ Returns

*None*

#### Example

```
<g:elevated_role_dialog />
```

## emitParms

Outputs all page parameters

**Library** Glide

### ⊕ Parameters

*None*

### ⊖ Returns

*None*

#### Example

```
<g2:emitParms />
```

## encryption\_select

Outputs the Encryption Context selector if it is in the system

**Library** Glide

### ⊕ Parameters

*None*

### ⊕ Returns

*None*

### Example

```
<j:if test="${pm.isActive('com.glide.encryption')}">
  <g:encryption_select />
</j:if>
```

## evaluate

Evaluates a given expression or body as JavaScript and stores the result in a variable or globally

**Library** Glide

### ⊕ Parameters

Attribute	Description	Type	Required	Default
var	The name of the variable to store the output in	String	No	
object	Sets whether the output should be treated as an object	Boolean	No	False
expression	The expression to evaluate if setting a var and body is not specified	String	No	
jelly	Sets whether the Jelly variables should be accessible to the JavaScript within	Boolean	No	False
copyToPhase2	If in Phase 1, controls whether to copy variables to Phase 2	Boolean	No	True
Body	JavaScript code to execute	JavaScript	No	

### ⊕ Returns

Return	Description	Type
var	If specified, the result of the script	Varies
Varies	Variables initialized in evaluate block will be available outside of it	Varies

### Example

```
<g:evaluate var="jvar_incident_rec" object="true" expression="var incidentRec = new GlideRecord('incident'); incidentRe
<g:evaluate var="jvar_incident_id" jelly="true">
    var incidentRec = new GlideRecord('incident');
    incidentRec.query();
    incidentRec.next();
</g:evaluate>
${incidentRec.sys_id}
```



## file\_browser

Gets a list of images for a given directory

**Library** Glide

### ⊕ Parameters

Attribute	Description	Type	Required	Default
directory	Directory to output images from	String	Yes	
exts	Image extensions to include in output	String	Yes	

### ⊕ Returns

*None*

### Example

```
<g:file_browser directory="images/UPLOADS" exts="jpeg,jpg,png,gif" />
```

## filter\_groups

Generates a choice list of Groups who can create Global Filters

**Library** Glide

### ⊕ Parameters

*None*

### ⊕ Returns

*None*

### Example

```
<g:filter_groups />
```

## for\_each\_record

Iterates over a `current` or given GlideRecord executing it's body for each record

**Library Glide****⊕ Parameters**

Attribute	Description	Type	Required	Default
file	GlideRecord data set	Reference	Yes	
max	Max number of iterations	Number	No	
Body	Code to execute for each iteration	Jelly/HTML/Text	Yes	

**⊕ Returns***None***Example**

```

<g:evaluate>
    var incidentRec = new GlideRecord('incident');
    incidentRec.addQuery("active","true");
    incidentRec.query();
</g:evaluate>

<ul>
    <g:for_each_record file="${incidentRec}" max="10">
        <li>${incidentRec.number} is ${jvar_active_row} of ${jvar_list_rows} on table ${jvar_list_file} with filter ${}
    </g:for_each_record>
</ul>

Skipped ${jvar_skipped_rows} Rows

```

**forEach**

Loops executing the body of the tag once per each item in the given set of values

**Library Jelly****⊕ Parameters**

Attribute	Description	Type	Required	Default
items	List of items to iterate over	List	Yes	
var	Variable to store current item in	String	Yes	
Body	Code to execute for each iteration	Jelly/HTML/Text	Yes	

**⊕ Returns**

Return	Description	Type
var	Current item in the list	Varies

**Example**

```

<j:set var="jvar_values" value="[[1,2,3,4,5]]" />
<j:forEach items="${jvar_values}" var="jvar_value">
    Hello World: ${jvar_value} <br/>

```

&lt;/j:forEach&gt;

## form

Deprecated. Includes a given template for a form

**Library** Glide

### ⊕ Parameters

Attribute	Description	Type	Required	Default
template	Name of the form (UI Macro/template) to output	String	Yes	

### ⊕ Returns

*None*

### Example

&lt;g:form template="welcome\_content.xml" /&gt;

## form\_label

Outputs a form label

**Library** Glide

### ⊕ Parameters

Attribute	Description	Type	Required	Default
for	Name of the HTML Input label is for	String	Yes	
Body	Content of the Label	Jelly/HTML/Text	Yes	

### ⊕ Returns

*None*

### Example

&lt;g:form\_label for="my\_field"&gt; Field:&lt;/g:form\_label&gt;&lt;br/&gt;

## function

Sets attributes for a Macro acting as a function

**Library** Glide

### ⊕ Parameters

Attribute	Description	Type	Required	Default
Varies	Attribute Names and options	String	No	

### ⊕ Returns

*None*

### Example

```
<g:function message="REQUIRED" />
```

## gauge

Renders a given gauge

Library Glide

### ⊕ Parameters

Attribute	Description	Type	Required	Default
id	Id of the Gauge to output	String	Yes	

### ⊕ Returns

*None*

### Example

```
<g:requires name="scripts/GlideV2ChartingIncludes.js" includes="true" />
<g:gauge id="18d48b3fc6112282015f087653a9c24c" />
```

## get\_action\_list

*No Description*

Library Glide

### ⊕ Parameters

*Unknown*

### ⊕ Returns

*Unknown*

### Example

No documented example

## get\_action\_list

*No Description*

**Libary Glide**

**⊕ Parameters**

*Unknown*

**⊕ Returns**

*Unknown*

**Example**

```
No documented example
```

## get\_canned\_messages

Gets a list of Canned Email messages

**Libary Glide**

**⊕ Parameters**

*None*

**⊕ Returns**

*None*

**Example**

```
None
```

## get\_collection\_data

*No Description*

**Libary Glide**

**⊕ Parameters**

*Unknown*

**⊕ Returns**

*Unknown*

**Example**

```
<g:get_collection_data ref="incident" id="9c573169c611228700193229fff72400" key="number" related_field="parent" related
```



## get\_connectors

No Description

Library Glide

### ⊕ Parameters

Unknown

### ⊕ Returns

Unknown

#### Example

```
No documented example
```

## get\_connectors

No Description

Library Glide

### ⊕ Parameters

Unknown

### ⊕ Returns

Unknown

#### Example

```
No documented example
```

## get\_distribution\_list

No Description

Library Glide

### ⊕ Parameters

Unknown

### ⊕ Returns

Unknown

#### Example

\*No documented example\*

## get\_distribution\_list

Used in the generation of Glide Lists

Library Glide

### ⊕ Parameters

Attribute	Description	Type	Required	Default
ref	Reference to the table to get values from	String	Yes	
record	Reference to the record to get values from	String	Yes	
type	<i>Unknown</i>			
readonly	Controls whether the list is editable			
ensure	<i>Unknown</i>			

### ⊕ Returns

*Unknown*

### Example

No documented example

## get\_filters

Gets the list of filters for a table and sets them to jvar\_filter\_list

Library Glide

### ⊕ Parameters

Attribute	Description	Type	Required	Default
tablename	Table name to pull filters from	String	Yes	

### ⊕ Returns

*None*

### Example

```
<g:get_filters tablename="incident" />
```

## get\_groupby\_list

Gets a list of groupable fields for a table.

**Library Glide****⊕ Parameters**

Attribute	Description	Type	Required	Default
table	Table to get groupable fields from	String	Yes	
related	<i>Unknown</i>			

**⊕ Returns***None***Example**

```
<g:evaluate var="incident" copyToRhino="true" object="true">
    sysparm_view="test";
    var gr = new GlideRecord('incident');
    gr.query();
    gr.next();
    gr;
</g:evaluate>

<g:get_groupby_list table="incident" />

${jvar_groupby_count} Total Options<br />

<select>
    <j:forEach items="${jvar_groupby_list}" var="jvar_group_field">
        <option>${jvar_group_field}</option>
    </j:forEach>
</select>
```

**get\_list**

Generates a list of fields associated with a particular list view.

**Library Glide****⊕ Parameters**

Attribute	Description	Type	Required	Default
form	Should be set to "list"	String	Yes	
ref	Reference to table to pull field list from	Reference	Yes	

**⊕ Returns***None***Example**

```
<g:evaluate var="incident" copyToRhino="true" object="true">
    sysparm_view="test";
    var gr = new GlideRecord('incident');
    gr.query();
    gr.next();
    gr;
</g:evaluate>
```

```
<select>
<g:get_list form="list" ref="incident" />
</select>
```

## get\_roles

Generates a list of available roles in the system

**Library** Glide

### ⊕ Parameters

Attribute	Description	Type	Required	Default
type	Controls which type of roles to return ("available", "selected")	String	Yes	
record	List of roles to exclude from the list	String	No	
readOnly	Controls whether the list is read only	Boolean	No	

### ⊕ Returns

None

## Example

```
<select>
  <g:get_roles type="available" record="" />
</select>
```

## gwt\_select\_list

Outputs the GWT multi-select list used in Slush Buckets

**Library** Glide

### ⊕ Parameters

Attribute	Description	Type	Required	Default
name	HTML Name of the element	String	Yes	
dropzone	Controls whether the list is a dropzone ("true")	Boolean	Yes	
readonly	Controls whether the list is editable	Boolean	No	

### ⊕ Returns

None

## Example

```
No documented example
```

## hdrftr\_cell\_data

*No Description*

**Library** Glide

 **Parameters**

*Unknown*

 **Returns**

*Unknown*

**Example**

```
No documented example
```

## hdrftr\_cell\_data

*No Description*

**Library** Glide

 **Parameters**

*Unknown*

 **Returns**

*Unknown*

**Example**

```
No documented example
```

## hdrftr\_layout

*No Description*

**Library** Glide

 **Parameters**

*Unknown*

 **Returns**

*Unknown*

**Example**

No documented example

## hdrftr\_layout

*No Description*

**Library** Glide

### ⊕ Parameters

*Unknown*

### ⊕ Returns

*Unknown*

### Example

No documented example

## help

*No Description*

**Library** Glide

### ⊕ Parameters

*Unknown*

### ⊕ Returns

*Unknown*

### Example

No documented example

## highlighter

Wraps instances of a search term in the given text in bold tags

**Library** Glide

### ⊕ Parameters

Attribute	Description	Type	Required	Default
search	The term to highlight in given text	String	Yes	
var	The variable to store the output HTML in	String	Yes	
Body	The text to search through	String	Yes	

## ⌚ Returns

*None*

## Example

```
<g2:highlighter search="ham" var="jvar_highlighted_text">
    Bacon ipsum dolor amet venison kevin brisket beef ribs, bacon ground round chuck jowl meatloaf turkey cow. Ground r
    Pork loin tri-tip biltong fatback alcatra tongue cow shank bacon ham andouille. Strip steak short loin prosciutto s
</g2:highlighter>

${jvar_highlighted_text}
```

## history

Generates a history view of a record

Library Glide

## ⊕ Parameters

Attribute	Description	Type	Required	Default
sysparm_table	The table of the record to pull history for	String	Yes	
sysparm_sys_id	The Sys Id of the record to pull history for	String	Yes	

## ⌚ Returns

*None*

## Example

```
<j:set var="sysparm_table" value="incident" />
<j:set var="sysparm_sys_id" value="46ee8c2fa9fe198100623592c70d643e" />
<g:history />
```

## html\_page

*No Description*

Library Glide

## ⊕ Parameters

Attribute	Description	Type	Required	Default
jvar_form_name	Reference to the table and record to output the page for	String	Yes	
jvar_html_template_name	The name of the template to use to output the page for	String	Yes	

## ⊕ Returns

*None*

### Example

```
<j:set var="jvar_form_name" value="incident.9c573169c611228700193229fff72400" />
<j:set var="jvar_html_template_name" value="form.xml" />

<g:html_page />
```

## if

Conditionally runs nested code if test passes

### Library Jelly

## ⊕ Parameters

Attribute	Description	Type	Required	Default
test	Expression to evaluate	String/JEXL	Yes	
Body	Block to execute if condition passes	Jelly/HTML/Text	Yes	

## ⊕ Returns

*None*

### Example

```
<j:set var="jvar_showHello" value="true" />
<j:if test="${jvar_showHello == true}">
    Hello World.
</j:if>
```

## image\_structure

Outputs a list of images in a directory

### Library Glide

## ⊕ Parameters

*None*

## ⊕ Returns

*None*

### Example

```
<select>
    <g:image_structure />
</select>
```

## impersonate\_button

Outputs the Impersonate User button

Library Glide

### ⊕ Parameters

*None*

### ⊕ Returns

*None*

#### Example

```
<g:impersonate_button />
```

## impersonate\_choices

Outputs a list of Impersonated Users

Library Glide

### ⊕ Parameters

*None*

### ⊕ Returns

*None*

#### Example

```
<g:impersonate_choices />
```

## impersonate\_dialog

Outputs the Impersonate User Dialog form

Library Glide

### ⊕ Parameters

*None*

### ⊕ Returns

*None*

#### Example

```
<g:impersonate_dialog />
```

## import\_vars

Parses URL parameters into Jelly variables

**Library** Glide

### ⊕ Parameters

Attribute	Description	Type	Required	Default
vars	URL Parameters to generate variables from	String	Yes	

### ⊕ Returns

Return	Description	Type
Varies	Output will vary on Parameter string	String

## Example

```
<g:import_vars vars="jvar_test=Hello World&jvar_name=Jelly Programmer" />
```

## include\_script

Includes a JavaScript file as a UI Script or externally

**Library** Glide

### ⊕ Parameters

Attribute	Description	Type	Required	Default
src	Source of the script to include	String	Yes	

### ⊕ Returns

None

## Example

```
<g:requires name="scripts/classes/GroupTreeNode.js"/>
```

## inline

Copies the given Jelly file to the current context

**Library** Glide

### ⊕ Parameters

Attribute	Description	Type	Required	Default
template	Name of the UI Macro or template to inline	String	Yes	

⊕ Returns

*None*

**Example**

```
<j:set var="jvar_message" value="Hello World" />
Message Outside Before: ${jvar_message}<br />
<g:inline template="my_function.xml" /><br />
Message Outside After: ${jvar_message}<br />
```

## insert

Executes a given Jelly file within the current context and returns the result inserting it inline

**Library** Glide

⊕ Parameters

Attribute	Description	Type	Required	Default
template	Name of the UI Macro or template to insert	String	Yes	

⊕ Returns

*None*

**Example**

```
<j:set var="jvar_message" value="Hello World" />
Message Outside Before: ${jvar_message}<br />
<g:insert template="my_function.xml" /><br />
Message Outside After: ${jvar_message}<br />
```

## insert\_form

*No Description*

**Library** Glide

⊕ Parameters

*Unknown*

⊕ Returns

*None*

**Example**

No documented example

## insert\_form

*No Description Available*

Library Glide

### ⊕ Parameters

Attribute	Description	Type	Required	Default
name	Name of the form to insert	String	Yes	

### ⊕ Returns

*None*

Example

\*None\*

## internal\_type\_options

Generates a choice list of Internal Types

Library Glide

### ⊕ Parameters

Attribute	Description	Type	Required	Default
includeContainerTypes	Controls whether to include form container types (splits)	Boolean	Yes	

### ⊕ Returns

*None*

Example

```
<select>
    <g:internal_type_options includeContainerTypes="true" />
</select>
```

## invoke

Invokes a method on a Java/JavaScript object

Library Jelly

### ⊕ Parameters

Attribute	Description	Type	Required	Default
method	Method to invoke	String	Yes	
on	Object to invoke method on	JEXL	Yes	
Body	arg tags that pass arguments to the invoke	Jelly	Varies	

### ⊕ Returns

None

### Example

```
<j:invoke method="add" on="${jvar_animals}">
    <j:arg value="rabbit" />
</j:invoke>
```

## invokeStatic

Invokes a method on a Java/JavaScript object

Libary Jelly

### ⊕ Parameters

Attribute	Description	Type	Required	Default
var	Variable to store the result in	String	Yes	
scriptableName	Name of the JavaScript scriptable object	String	Yes	
method	Method on the object to call	String	Yes	
Body	arg tags that pass arguments to the invoke	Jelly	Varies	

### ⊕ Returns

Return	Description	Type
var	Result of the method called	Varies

### Example

```
<j:invokeStatic var="jvar_by_name" scriptableName="GlideUser" method="resolveNameFromSysID">
    <j:arg value="6816f79cc0a8016401c5a33be04be441"/>
</j:invokeStatic>
Name: ${jvar_by_name}
```

## item\_link

No Description

Libary Glide

### ⊕ Parameters

Attribute	Description	Type	Required	Default
show_link	Controls whether the link is generated	Boolean	Yes	

⊕ Returns

Return	Description	Type
jvar_show_link	<i>Unknown</i>	Boolean

Example

```
<g:item_link show_link="true" />
${jvar_show_link}
```

## kb\_advanced\_search

Outputs the KB Advanced Search form

Library Glide

⊕ Parameters

*None*

⊕ Returns

*None*

Example

```
No documented example
```

## kb\_header\_search

Outputs the KB Header Search Form

Library Glide

⊕ Parameters

*None*

⊕ Returns

*None*

Example

```
<g:kb_header_search />
```

## kb\_languages

Outputs the KB Languages Drop Down

**Library** Glide

 **Parameters**

*None*

 **Returns**

*None*

**Example**

```
No documented example
```

## kb\_page\_vcr

Outputs the KB Pagination Functions (Old style)

**Library** Glide

 **Parameters**

*None*

 **Returns**

*None*

**Example**

```
<g2:kb_page_vcr />
```

## kb\_v3\_vcr

Outputs the KB Pagination Functions (New style)

**Library** Glide

 **Parameters**

*None*

 **Returns**

*None*

**Example**

```
<g2:kb_v3_vcr />
```

## kb\_view\_attach\_button

Outputs the KB Attach Knowledge Button

**Library** Glide

### ⊕ Parameters

Attribute	Description	Type	Required	Default
kb_search_table	Table KB Search originated from	String	Yes	

### ⊕ Returns

*None*

### Example

```
<g:kb_view_attach_button kb_search_table="incident" />
```

## kb\_view\_edit\_button

Outputs the KB Edit Button

**Library** Glide

### ⊕ Parameters

*None*

### ⊕ Returns

*None*

### Example

```
<g:kb_view_edit_button />
```

## label\_spacing

Outputs an input box used for spacing

**Library** Glide

### ⊕ Parameters

*None*

### ⊕ Returns

*None*

**Has Body** None

**Example**

```
<g:label_spacing />
```

**ldap\_detail**

Creates a list of LDAP attributes

**Library** Glide

**⊕ Parameters**

Attribute	Description	Type	Required	Default
server	Server to get Attributes for	String	Yes	
ou	Organizational Unit - Use Unknown	String	No	
dn	Distinguished Name - Use Unknown	String	No	

**⊕ Returns**

*None*

**Example**

```
<g2:ldap_detail server="${sysparm_server_id}">
<ul>
    <j2:forEach var="jvar_attr" items="${jvar_ldap_attributes}">
        <li>${jvar_attr.label} - ${jvar_attr.value}</li>
    </j2:forEach>
</ul>
```

**lightweight\_glide\_list**

Outputs a Glide List element

**Library** Glide

**⊕ Parameters**

Attribute	Description	Type	Required	Default
jvar_list_data	Current value of the List	String	Yes	
jvar_reference	Table the List references	String	Yes	
jvar_can_write	Controls whether the list is editable	Boolean	Yes	
jvar_control_name	Name of the HTML Element	String	Yes	
jvar_not_focused	Controls whether the list has focus when the page loads	Boolean	Yes	

**⊕ Returns**

*None*

**Example**

```
<j:set var="jvar_list_data" value="" />
<j:set var="jvar_reference" value="sys_user" />
<j:set var="jvar_can_write" value="true" />
<j:set var="jvar_control_name" value="Favorite Users" />
<j:set var="jvar_not_focused" value="true" />
<g:lightweight.glide_list />
```

**list\_control**

Sets some variables pertaining to List views.

**Library** Glide

**⊕ Parameters**

Attribute	Description	Type	Required	Default
ref	Reference to the table a list is being generated for	Reference	Yes	

**⊖ Returns**

*None*

**Example**

```
<g:evaluate var="incident" copyToRhino="true" object="true">
    var gr = new GlideRecord('incident');
    gr.addQuery("active","true")
    gr.query();
    gr.next();
    gr;
</g:evaluate>

<g:list_control ref="incident" />

Control ID: ${jvar_list_control_id}<br />
Sys ID: ${jvar_list_control_sys_id}<br />
Control Query: ${jvar_list_control_query}<br />
Use Relationship Banner: ${jvar_use_relationship_banner}<br />
Relationship Banner: ${jvar_relationship_banner}<br />
List Edit Type: ${jvar_list_edit_type}<br />
Ref Qualifier Tag: ${jvar_list_edit_ref_qual_tag}<br />
```

**list\_custom**

Sets attributes to generate a list for a table with custom fields

**Library** Glide

**⊕ Parameters**

Attribute	Description	Type	Required	Default
table	The table list is being output for	String	Yes	
view	Specific view to use to generate the list	String	No	
elements	Elements to include as columns in the list	String	Yes	

## ⊕ Returns

*Unknown*

## Example

```
<g:list_custom table="incident" view=""" elements="number,short_description ""/>
<g:inline template="list2_default.xml" />
```

## list\_default

Sets attributes to generate a list for a table with default fields

### Library Glide

## ⊕ Parameters

Attribute	Description	Type	Required	Default
table	The Table the list is being output for	String	Yes	
view	Specific view to use to generate the list	String	Yes	

## ⊕ Returns

*None*

## Example

```
*None*
```

## list\_mechanic\_inner

Outputs the List Mechanic Slush Bucket

### Library Glide

## ⊕ Parameters

*None*

## ⊕ Returns

*None*

## Example

```
<g:evaluate>
    ListProperties.setHasListMechanic(true);
</g:evaluate>
<g:list_mechanic_inner />
```

## list\_mechanic2

Outputs the List Mechanic Button

### Library Glide

#### Parameters

None

#### Returns

None

### Example

```
<g:evaluate>
    ListProperties.setHasListMechanic(true);
</g:evaluate>
<g:list_mechanic2 />
```

## list\_record\_default

Sets additional attributes for generating a list

### Library Glide

#### Parameters

Attribute	Description	Type	Required	Default
properties	List of properties to used to control the list output	String	Yes	
query	Query to apply to the list. Should be an encoded query.	String	Yes	
Body	The body here should inline the list2.xml template	String	Yes	

#### Returns

None

### Example

```
<g2:list_record_default properties="${ListProperties.serialize()}" query="task=3dd5fda90f2fb1005c95059ce1050e70">
    <g:inline template="list2.xml" />
</g2:list_record_default>
```

## list\_record\_relationship

Sets additional attributes for generating a related list

### Library Glide

#### Parameters

Attribute	Description	Type	Required	Default
properties	List of properties used to control the list output	String	Yes	

parentID	Sys Id of the parent record for the relationship	String	Yes	
query	Query to apply to the list. Should be an encoded query.	String	No	
Body	The body here should inline the list2.xml template	String	Yes	

### ⊕ Returns

*Unknown*

### Example

```
<g2:list_record_relationship properties="${ListProperties.serialize()}" parentID="845f7e720f3231005c95059ce1050e33" que
    <g:inline template="list2.xml" />
</g2:list_record_relationship>
```

## list\_related

Sets attributes to generate a related list

### Library Glide

### ⊕ Parameters

Attribute	Description	Type	Required	Default
parentTable	Parent table of the record for the relationship	String	Yes	
related	Field the realationship is built on	String	Yes	

### ⊕ Returns

*Unknown*

### Example

```
<g:list_related parentTable="task" related="task_sla.task" />
```

## list\_relationship

Sets attributes to generate a list from a System Relationship

### Library Glide

### ⊕ Parameters

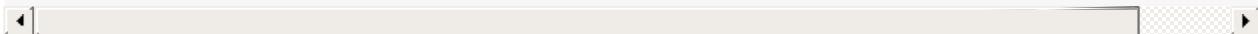
Attribute	Description	Type	Required	Default
parentTable	Parent table of the relationship	String	Yes	
related	Sys Id of the relationship	String	Yes	
view	View to apply to the list	String	Yes	
isEmbedded	Controls whether the list is embedded or not	Boolean	String	

## ⊕ Returns

*Unknown*

### Example

```
<g:list_relationship parentTable="sys_db_object" related="REL:4344f6f5bf1320001875647fcf0739ad" view="" isEmbedded="true">
```



## log\_file\_browser

Outputs the Log File Browser form

Library Glide

## ⊕ Parameters

*None*

## ⊕ Returns

*None*

### Example

```
<g:log_file_browser />
```

## macro

Specifies the start of a reusable block and its parameters

Library Glide

## ⊕ Parameters

Attribute	Description	Type	Required	Default
Varies	The name of an attribute for the Macro	String	No	

## ⊕ Returns

*None*

### Example

```
<g:macro message="Hello World" name="" />
```

## macro\_invoke

Calls a Macro

**Library Glide****⊕ Parameters**

Attribute	Description	Type	Required	Default
macro	The name of the UI Macro or template to invoke	String	Yes	
Varies	Any additional attributes required by the macro	Varies	No	

**⊕ Returns***None***Example**

```
<g:macro_invoke macro="show_cascade_del_objs_table" obj_list="${jvar_delObjList}" />
```

**mapping\_list**

Generates a choice list of Import Export fields depending on type

**Library Glide****⊕ Parameters**

Attribute	Description	Type	Required	Default
type	Specifies whether to pull "selected" or "available"	String	Yes	
source	Source to pull values from, "database" or "selected"	String	Yes	

**⊕ Returns***None***Example**

```
<j2:set var="sysparm_sys_id" value="8da2f3db2b74b1009728f70a89da157d" />
Available Fields
<select>
  <g2:mapping_list type="available" source="database" />
</select>
```

**mapping\_sample\_data**

Generates a list of Sample Data for a particular Import Export map

**Library Glide****⊕ Parameters***None***⊕ Returns**

*None***Example**

```
<j2:set var="sysparm_sys_id" value="8da2f3db2b74b1009728f70a89da157d" />
Available Fields
<select>
    <g2:mapping_list type="available" source="database" />
</select>
<br/>
Selected Fields
<select>
    <g2:mapping_list type="selected" source="database" />
</select>
<br/>
Selected External Fields
<select>
    <g2:mapping_list type="selected" source="selected" />
</select>
<br/>
Available External Fields
<select>
    <g2:mapping_list type="available" source="selected" />
</select>
<br/>
<g2:forEach var="jvar_field" items="${jvar_import_fields}">
    ${jvar_field.name}
    <ul>
        <j2:forEach var="jvar_field_value" items="${jvar_field.value9}">
            <li>${jvar_field_value}</li>
        </j2:forEach>
    </ul>
</j2:forEach>

```

**menu\_lists**

Generates a menu list for a given type.

**Library** Glide**⊕ Parameters**

Attribute	Description	Type	Required	Default
device_type	The device type ("browser","mobile")	String	Yes	

**⊕ Returns***None***Example**

```
<g:menu_lists device_type="browser" />
```

**merge\_tags\_input**

Outputs a form used to merge Tags

**Library** Glide

## ⊕ Parameters

*None*

## ⊖ Returns

*None*

### Example

```
No documented example
```

## messages

Renders a client side script with translated messages (if they exist)

Library Glide

## ⊕ Parameters

Attribute	Description	Type	Required	Default
<i>Body</i>	A list of messages to output on the client	Text	Yes	

## ⊖ Returns

*None*

### Example

```
<g:messages>
Run
</g:messages>
<script>
console.log( new GwtMessage().getMessage("Run"));
</script>
```

## mid\_tools\_progress\_worker

Outputs an AJAX Progress watcher for a MID server Job

Library Glide

## ⊕ Parameters

Attribute	Description	Type	Required	Default
<i>ecc_queue_id</i>	The ID of the Job to watch	String	Yes	

## ⊖ Returns

*None*

### Example

```
<g:mid_tools_progress_worker ecc_queue_id="${jvar_ecc_queue_id}" />
```

## mobile\_version\_switch

Outputs the link to Switch to Desktop version from mobile

**Library** Glide

### ⊕ Parameters

*None*

### ⊕ Returns

*None*

### Example

```
<g:mobile_version_switch />
```

## my\_groups

Generates a choice list of the current users Groups

**Library** Glide

### ⊕ Parameters

*None*

### ⊕ Returns

*None*

### Example

```
<select>
  <g:my_groups />
</select>
```

## new

Creates a new object of a specified type. Since Packages cannot be used this is unusable in ServiceNow. Objects can be created in g:evaluate blocks

**Library** Jelly

### ⊕ Parameters

*Unknown*

### ⊕ Returns

*None*

### Example

```
None
```

## no\_escape

Prevents escaping text within the body of the tag

Library Glide

### ⊕ Parameters

Attribute	Description	Type	Required	Default
<i>Body</i>	The text to prevent escaping for	Jelly/HTML/Text	Yes	

### ⊖ Returns

*None*

### Example

```
 ${LT}
<br />
<g:no_escape>
    ${LT}
</g:no_escape>
```

## noop

Performs no operation

Library Glide

### ⊕ Parameters

Attribute	Description	Type	Required	Default
noopAttribute	<i>Unknown</i>	String	Yes	
<i>Body</i>	Body which does nothing	String	Yes	

### ⊖ Returns

*None*

### Example

```
<g:noop noopAttribute="""">
    Nothing going on here.
</g:noop>
```

## olap\_cube\_options

*No Description*

**Library** Glide

### ⊕ Parameters

*Unknown*

### ⊖ Returns

*Unknown*

### Example

```
No documented example
```

## olap\_view

*No Description*

**Library** Glide

### ⊕ Parameters

*Unknown*

### ⊖ Returns

*Unknown*

### Example

```
No documented example
```

## options

Outputs a choice list for a field

**Library** Glide

### ⊕ Parameters

Attribute	Description	Type	Required	Default
ref	Reference to the field to output options for	Reference	Yes	
choiceValue	The selected value	String	No	

### ⊖ Returns

*None*

**Example**

```
<g:evaluate var="incident" copyToRhino="true" object="true">
    var gr = new GlideRecord('incident');
    gr.query();
    gr.next();
    gr;
</g:evaluate>

<select>
<g:options ref="incident.priority" choiceValue="3" />
</select>
```

**otherwise**

Nested tag within a `when` block which conditionally runs nested code if no sibling `when` tag passes

**Libary** Jelly

**⊕ Parameters**

Attribute	Description	Type	Required	Default
<code>Body</code>	Block to execute if condition is met	String	Yes	

**⊖ Returns**

*None*

**Example**

```
<j:otherwise>
    Not Sure
</j:otherwise>
```

**parse**

Requires an XMLReader which is not accessible in ServiceNow.

**Libary** Jelly

**⊕ Parameters**

*Unknown*

**⊖ Returns**

*None*

**Example**

*None*

**partial\_page**

Wraps a section in a "partialPage" span tag. This tag seems to have been used when creating charts.

### Library Glide

#### ⊕ Parameters

Attribute	Description	Type	Required	Default
name	The name of the partial page	String	Yes	
Body	Content to be included in the partial	Jelly/HTML/Text	Yes	

#### ⊖ Returns

None

### Example

```
<g2:partial_page name="myPartial">
<h4>Hello World</h4>
</g2:partial_page>
```

## preference

Gets a given User Preference

### Library Glide

#### ⊕ Parameters

Attribute	Description	Type	Required	Default
name	Name of the preference to get	String	Yes	
var	Variable to store the result in	String	Yes	
default	Default value if preference does not exist	String	Yes	

#### ⊖ Returns

Return	Description	Type
Specified 'var'	The preference value	String

### Example

```
<g:preference name="homepage" var="jvar_homepage" default="NONE" />
${jvar_homepage}
```

## ref\_element

Generates a list of unique values for a field over all records in a table

### Library Glide

## ⊕ Parameters

Attribute	Description	Type	Required	Default
name	Reference to the field to pull values from	Reference	Yes	
var	Variable to store the output in	String	Yes	
query	Query to apply to the table. Should be an encoded query	String	No	

## ⊖ Returns

Return	Description	Type
Specified 'var'	The preference value	String

## Example

```
<g:ref_element name="incident.short_description" var="jvar_test" query="active=true" />
<ul>
    <j:forEach items="${jvar_test}" var="jvar_item">
        <li>${jvar_item}</li>
    </j:forEach>
</ul>
```

## reference\_decoration

Outputs a field decoration for a Reference Field of a set type

### Libary Glide

## ⊕ Parameters

Attribute	Description	Type	Required	Default
onclick	JavaScript to Execute when the decoration is clicked	String	Yes	
id	HTML ID of the Element	String	Yes	
field	Field decoration is tied to	String	Yes	
image	Image to use for the decoration	String	Yes	
title	The title to be used for the decoration	String	Yes	

## ⊖ Returns

None

## Example

```
<g:reference_decoration id="show_workflow.${ref}" field="${ref}"
    onclick="showWorkflow('${ref}')"" title="${gs.getMessage('Show Workflow')}"
    image="images/workflow/wkfl_definition.gifx" />
```

## reference\_decoration\_custom

Outputs a custom decoration for a Reference Field

**Library Glide****⊕ Parameters**

Attribute	Description	Type	Required	Default
onclick	JavaScript to Execute when the decoration is clicked	String	Yes	
id	HTML ID of the Element	String	Yes	
field	Field decration is tied to	String	Yes	
Body	Content to be used for the decoration	Jelly/HTML/Text	Yes	

**⊕ Returns***None***Example**

```
<g:reference_decoration_custom id="${jvar_n}" field="${ref}"
    onclick="showRelatedList('${ref}'); ">
    <span class="tab_square_button tab_header_button tab_ref_contribution icon-tree-right"
        aria-label="${HTML:gs.getMessage('Show related incidents')}>
    </span>
</g:reference_decoration_custom>
```

**reference\_options**

Generates a choice list of all fields of 'document\_id' types in all tables

**Library Glide****⊕ Parameters**

Attribute	Description	Type	Required	Default
table	<i>None</i>	String	No	

**⊕ Returns***None***Example**

```
<select>
    <g:reference_options />
</select>
```

**reminder\_field\_name\_options**

Generates a choice list of Reminder Field values

**Library Glide****⊕ Parameters**

Attribute	Description	Type	Required	Default
tableName	The table to list fields for	String	Yes	

⊕ **Returns**

*None*

**Example**

```
<select>
  <g:reminder_field_name_options tableName="incident" />
</select>
```

## remote\_import

*No Description*

**Library** Glide

⊕ **Parameters**

*Unknown*

⊕ **Returns**

*None*

**Example**

No documented example

## remote\_import

*No Description*

**Library** Glide

⊕ **Parameters**

*Unknown*

⊕ **Returns**

*Unknown*

**Example**

No documented example

## remove

Deletes a given Jelly variable

### Library Jelly

#### Parameters

Attribute	Description	Type	Required	Default
var	Variable to delete	String	Yes	

#### Returns

None

### Has Body

#### Example

```
<j:remove var="jvar_name" />
```

## render\_component

Renders a component

### Library Glide

#### Parameters

Attribute	Description	Type	Required	Default
componentName	Name of the component to render :)	String	Yes	

#### Returns

None

#### Example

```
<g:render_component componentName="XXXXXX" />
```

## render\_content\_block\_CONTENT\_TYPE

Renders a content block for a CONTENT TYPE

### Library Macro

#### Parameters

Varies

#### Returns

None

**Example**

```
*None*
```

**report\_choice\_list***No Description***Library** Glide**⊕ Parameters***Unknown***⊕ Returns***None***Example**

```
No documented example
```

**report\_choice\_list***No Description***Library** Glide**⊕ Parameters***Unknown***⊕ Returns***Unknown***Example**

```
No documented example
```

**reportmessage**

Outputs the given response messages for a field

**Library** Glide**⊕ Parameters**

Attribute	Description	Type	Required	Default
field	Field to output the message for	Reference	Yes	

## ⊕ Returns

*None*

### Example

```
<g:evaluate var="incident" copyToRhino="true" object="true">
    var gr = new GlideRecord('incident');
    gr.query();
    gr.next();
    gr.short_description.setError('Short Description should not contain Jelly.');
    gr;
</g:evaluate>

Message:
<g:reportmessage field="incident.short_description" />
<br/>
Element:
<g:element ref="incident.short_description" />
```

## requires

Includes a JavaScript file on the system

### Library Glide

## ⊖ Parameters

Attribute	Description	Type	Required	Default
name	The name of the script to output	String	Yes	

## ⊕ Returns

*None*

### Example

```
<g:requires name="scripts/classes/GroupTreeNode.js"/>
```

## response\_options

Generates a list of unique values for a field over all records in a table

### Library Glide

## ⊖ Parameters

Attribute	Description	Type	Required	Default
table	The table to pull values from	String	Yes	
field	The field to pull unique values from	String	Yes	

## ⊕ Returns

None

### Example

```
<select>
    <g:response_options table="incident" field="priority" />
</select>
```

## sc\_catalog\_item\_href

Library Glide

### ⊕ Parameters

None

### ⊕ Returns

None

### Example

None

## scope

Creates a Jelly scope

Library Jelly

### ⊕ Parameters

Attribute	Description	Type	Required	Default
Body	Code to be executed in the scope	Jelly/HTML/Text	Yes	

### ⊕ Returns

None

### Example

```
<j:scope>
    <j:set var="jvar_name" value="Domenic" />
    In Scope: ${jvar_name} <br/>
</j:scope>
Out of Scope: ${jvar_name}
```

## scripted\_options

Generates a choice list by calling a function within a script include

Library Glide

## ⊕ Parameters

Attribute	Description	Type	Required	Default
selected	The selected value	String	No	
script	The name of the Script Include to run	String	Yes	
function	The name of the function to run	String	Yes	

## ⊕ Returns

*None*

### Example

```
<select>
    <g:scripted_options selected="Requested Item" script="WFStageSet" function="getSetNames" />
</select>
```

## scripted\_table\_options

*No Description*

**Library** Glide

## ⊕ Parameters

*Unknown*

## ⊕ Returns

*None*

### Example

No documented example

## scrollable\_area

Outputs a Scrolling Area similar to that used for News

**Library** Glide

## ⊕ Parameters

Attribute	Description	Type	Required	Default
height	The height of the scrollable area	String	Yes	
Body	The content to scroll	Jelly/HTML/Text	Yes	

## ⊕ Returns

*None*

**Example**

```
<g2:scrollable_area height="100px">
  <p>Here Is Some Content</p>
</g2:scrollable_area>
```

**search\_table\_options**

Outputs option tags for each searchable table

**Library** Glide**⊕ Parameters**

*None*

**⊕ Returns**

*None*

**Example**

```
<select>
  <g:search_table_options />
</select>
```

**section**

Outputs the body of the tag in a new section on a form

**Library** Glide**⊕ Parameters**

Attribute	Description	Type	Required	Default
Body	Content to be included in the section	Jelly/HTML/Text	Yes	

**⊕ Returns**

*None*

**Example**

```
<j:set var="jvar_section_caption" value="My Section" />
<g:section>
  <h3>This is My Section</h3>
</g:section>
```

**section\_list**

Renders a ServiceNow form header

**Library Glide****⊕ Parameters**

Attribute	Description	Type	Required	Default
headerOnly	Controls whether only the header should be output	String	Yes	
headerSpacer	Controls whether a spacer should be output	String	Yes	
headerSimple	<i>Unknown</i>	String	Yes	

**⊕ Returns***None***Example**

```
<g:evaluate var="incident" copyToRhino="true" object="true">
    var gr = new GlideRecord('incident');
    gr.query();
    gr.next();
    gr;
</g:evaluate>

<j:set var="ref" value="incident" />
<g:section_list headerOnly="true" headerSpacer="false" headerSimple="false"/>
```

**session\_notifications**

Outputs the session Notifications widget

**Library Glide****⊕ Parameters***None***⊕ Returns***None***Example**

```
<g:evaluate>
    gs.addErrorMessage("This is an error.");
</g:evaluate>
<g:session_notifications />
```

**set**

Sets a given Jelly variable to a value

**Library Jelly****⊕ Parameters**

Attribute	Description	Type	Required	Default
var	The variable to store the result in	String	Yes	
value	The expression or value to set the variable to	String/JEXL	Yes	

### ⊕ Returns

None

### Has Body

### Example

```
<j:set var="jvar(firstName" value="Domenic" />
```

## set\_if

Sets a variable conditionally based on a given test

### Library Glide

### ⊕ Parameters

Attribute	Description	Type	Required	Default
var	The variable to store the result in	String	Yes	
test	The condition that will evaluate	JEXEL	Yes	
true	The value to set the variable to if the test is true	String/JEXL	Yes	
false	The value to set the variable to if the test is False	String/JEXL	Yes	

### ⊕ Returns

None

### Example

```
<j:set var="jvar_isHelloWorld" value="true" />
<g:set_if var="jvar_message" test="${jvar_isHelloWorld=='true'}" true="Hello World." false="Not Hello World."/>
${jvar_message}
```

## setjs

Sets a variable using JavaScript (instead of a JEXL expression)

### Library Glide

### ⊕ Parameters

Attribute	Description	Type	Required	Default
var	The variable to store the result in	String	Yes	
expression	The JavaScript expression to evaluate	JavaScript		

## ⊕ Returns

*None*

### Example

```
<j:set var="jvar_random" value="${Math.random()}" />
Output: ${jvar_random}
<g:setjs var="random" expression="Math.random()" />
Output: ${random}

<g:evaluate>
  gs.log(jvar_random)
  gs.log(random)
</g:evaluate>
```

## setProperties

Sets a property on an object created with new which is not usable in ServiceNow.

### Library Jelly

## ⊕ Parameters

*Unknown*

## ⊕ Returns

*None*

### Example

```
None
```

## solution\_button

Outputs the Knowledge Base Solution Button

### Library Glide

## ⊕ Parameters

*None*

## ⊕ Returns

*None*

### Example

```
<g:solution_button/>
```

## summarize\_question

Outputs a request item Question and Answer

**Library** Glide

### ⊕ Parameters

Attribute	Description	Type	Required	Default
question	Reference to a Service Catalog question	Reference	Yes	

### ⊕ Returns

None

### Example

```
<j:forEach var="jvar_question" items="${set.getFlatQuestions()}>
    <j:if test="${jvar_question.isVisibleSummary()}">
        <g:summarize_question question="${jvar_question}" />
    </j:if>
</j:forEach>
```

## switch

Evaluates nested `case` blocks as a single set of conditions and runs blocks if their value matches the given variable

**Library** Jelly

### ⊕ Parameters

Attribute	Description	Type	Required	Default
on	The variable to switch on	String/JEXL	Yes	
Body	Set of cases to compare to the var	Jelly	Yes	

### ⊕ Returns

None

### Example

```
<j:set var="jvar_showMessage" value="Hello" />

<j:switch on="${jvar_showMessage}">
    <j:case value="Hello">
        Hello
    </j:case>
    <j:case value="Goodbye">
        Goodbye
    </j:case>
    <j:case value="Hello">
        Hello Again.
    </j:case>
    <j:default>
        Not Sure
    </j:default>
```

```
</j:switch>
```

## sysevent\_name\_options

Generates a choice list of System Events

**Library** Glide

### ⊕ Parameters

*None*

### ⊕ Returns

*None*

#### Example

```
<select>
  <g:sysEvent_name_options />
</select>
```

## system\_properties

Outputs a list of System Properties depending on the filters

**Library** Glide

### ⊕ Parameters

Attribute	Description	Type	Required	Default
exclude	List of property groups to exclude	String	Yes	
include	List of property groups to include	String	Yes	

### ⊕ Returns

*None*

#### Example

```
<g2:system_properties exclude="" include="CSS" />
```

## table\_options

Generates a choice list of tables

**Library** Glide

### ⊕ Parameters

*None*

### ⌚ Returns

*None*

### Example

```
<select>
    <g:table_options/>
</select>
```

## text\_search\_widget

Outputs the Text Search Widget

Library Glide

### ⌚ Parameters

*None*

### ⌚ Returns

*None*

### Example

```
None
```

## textarea

Outputs a text area with the given id, class and body

Library Glide

### ⌚ Parameters

Attribute	Description	Type	Required	Default
id	ID of the HTML Element	String	Yes	
class	Class of the HTML Element	String	Yes	
Body	Value of the textarea	String	Yes	

### ⌚ Returns

*None*

### Example

```
<g:textarea id="my_textarea" class="form-control">
Heres some text
Heres another line
</g:textarea>
```

## thread

Runs the Jelly in the body of the tag in a separate thread. This has no effect on UI Pages and Macros.

**Library** Jelly

### ⊕ Parameters

*Unknown*

### ⊕ Returns

*None*

### Example

```
None
```

## tokenize

Creates a List from the body of the tag and stores it in a variable

**Library** Glide

### ⊕ Parameters

Attribute	Description	Type	Required	Default
delim	Delimiter used to split the string	String	Yes	
var	Variable to store the output in	String	Yes	
Body	Text to split apart	String	Yes	

### ⊕ Returns

*None*

### Example

```
<g:tokenize delim=""\n" var="jvar_test">
this
is
a
test
of
tokenize
</g:tokenize>

${jvar_test.size()}
${jvar_test}
```

## ui\_button

Outputs a Button

**Libary Glide****⊕ Parameters**

Attribute	Description	Type	Required	Default
action	The name of a UI Action to be tied to the button	String	Yes	
Body	The content of the Button	String	Yes	

**⊕ Returns***None***Example**

```
<g:ui_button action="save">${gs.getMessage('Save')}</g:ui_button>
```

**ui\_checkbox**

Outputs a Checkbox

**Libary Glide****⊕ Parameters**

Attribute	Description	Type	Required	Default
id	The ID of the HTML Element	String	Yes	
name	The name of the HTML Element	String	Yes	
value	The value of the HTML Element	String	Yes	

**⊕ Returns***None***Example**

```
<g:ui_checkbox id="my_checkbox" name="my_checkbox" value="false" />
```

**ui\_choice\_input\_field**

Outputs a Choice List

**Libary Glide****⊕ Parameters**

Attribute	Description	Type	Required	Default
id	The ID of the HTML Element	String	Yes	
name	The Name of the HTML Element	String	Yes	

label	The text of the Label for the HTML Element	String	Yes	
mandatory	Controls whether the field is mandatory	String	Yes	
for	Should match the Name of the element	String	Yes	
onchange	JavaScript to execute when the field is changed	String	Yes	
Body	List of options for the Choice List	Jelly/HTML	Yes	

### ⊕ Returns

None

### Example

```
<g:ui_choice_input_field id="test_choice" name="test_choice" for="test_choice" label="Test Choice" mandatory="true" on
<option value="none">${gs.getMessage('-- None --')}</option>
<option value="First">First</option>
</g:ui_choice_input_field>
```



## ui\_choicelist

Outputs a Choice List

Libary Glide

### ⊕ Parameters

Attribute	Description	Type	Required	Default
class	The class of the HTML Element	String	Yes	
name	The name of the HTML Element	String	Yes	
table	The table the choices should be pulled from	String	Yes	
field	The field the choices should be pulled from	String	Yes	

### ⊕ Returns

None

### Example

```
<g:ui_choicelist class="form-control" name="myquantity" table="sc_cart_item" field="quantity" />
```

## ui\_date

Outputs a Date Picker

Libary Glide

### ⊕ Parameters

Attribute	Description	Type	Required	Default

<b>name</b>	The name of the HTML Element	String	Yes	
<b>id</b>	The ID of the HTML Element	String	Yes	
<b>value</b>	The value of the HTML Element	String	Yes	
<b>onchange</b>	The JavaScript to execute when the value is changed	String	Yes	

#### ⊕ Returns

*None*

#### Example

```
<g:ui_date name="birthday" id="birthday" value="2014-07-11" onchange="datePicked(this.value);"/>
```

## ui\_date\_time

Outputs a Date Time Picker

Library Glide

#### ⊕ Parameters

Attribute	Description	Type	Required	Default
<b>name</b>	The name of the HTML Element	String	Yes	
<b>id</b>	The ID of the HTML Element	String	Yes	
<b>value</b>	The value of the HTML Element	String	Yes	
<b>onchange</b>	The JavaScript to execute when the value is changed	String	Yes	

#### ⊕ Returns

*None*

#### Example

```
<g:ui_date_time name="start_time" value="${start_time}"/>
```

## ui\_form

Outputs a form wrapper

Library Glide

#### ⊕ Parameters

Attribute	Description	Type	Required	Default
<b>id</b>	The ID of the HTML Element	String	Yes	
<b>form_class</b>	The class of the HTML Element	String	Yes	
	The JavaScript to execute when the form is			

	submitted		
Body	The content of the form	Jelly/HTML/Text	Yes

### ⊕ Returns

None

### Example

```
<g:ui_form id="my_form" form_class="form-horizontal" onsubmit="return someFunction('ok');"">
<g:ui_date name="birthday" id="birthday" value="2014-07-11" onchange="datePicked(this.value);"" />
</g:ui_form>
```

## ui\_header

Outputs a form header

Library Glide

### ⊕ Parameters

Attribute	Description	Type	Required	Default
Body	Content of the header	String	Yes	

### ⊕ Returns

None

### Example

```
<g:ui_header>My Form</g:ui_header>
```

## ui\_input\_field

Outputs a text input field

Library Glide

### ⊕ Parameters

Attribute	Description	Type	Required	Default
label	The label of the HTML Element	String	Yes	
name	The name of the HTML Element	String	Yes	
value	The value of the HTML Element	String	Yes	
onkeyup	The JavaScript to execute when a key is pressed	Yes		
onchange	The JavaScript to execute when the field is changed	Yes		
readonly	Controls whether the field is editable			
style	The style of the HTML Element	String	Yes	

## ⌚ Returns

*None*

## Example

```
<g:ui_input_field label="My Input Field" name="my_field" value="Hello World"/>
```

## ui\_language\_select

Outputs a the Language Selector if enabled on the Instance

### Library Glide

## ⊕ Parameters

*None*

## ⌚ Returns

*None*

## Example

```
<g:ui_language_select />
```

## ui\_multiline\_input\_field

Outputs a text area

### Library Glide

## ⊕ Parameters

Attribute	Description	Type	Required	Default
name	The name of the HTML Element	String	Yes	
rows	The number of rows to display	Number	Yes	
size	The size of the HTML Element	Number	No	
label	The label of the HTML Element	String	No	
value	The value of the HTML Element	String	No	
mandatory	Controls whether the field is required	Boolean	No	
onchange	The JavaScript to execute when the field is changed	String	No	
onkeyup	The JavaScript to execute when a key is pressed	String	No	
oncontextmenu	<i>Unknown</i>	String	No	
readonly	Controls whether the field is editable	Boolean	No	
style	The style of the HTML Element			

## ⌚ Returns

None

### Example

```
<g:ui_multiline_input_field name="my_large_text" rows="4" size="100%" label="My Large Text Field" value="" mandatory="f
```



## ui\_progress\_worker

Outputs an AJAX Progress watcher for a worker

Library Glide

## ⌚ Parameters

Attribute	Description	Type	Required	Default
worker_id	The ID of the worker to output the watcher for	String	Yes	

## ⌚ Returns

None

### Example

```
<g:ui_progress_worker worker_id="${sysparm_pworker_sysid}" />
```

## ui\_reference

Outputs a reference field

Library Glide

## ⌚ Parameters

Attribute	Description	Type	Required	Default
name	The name of the HTML Element			
table	The table the field refers to	String	Yes	
value	The initial value of the field	String	Yes	
displayvalue	<i>Unknown</i>	<i>Unknown</i>	No	
show_popup	Controls whether the popup icon is displayed	Boolean	No	
show_lookup	Controls whether the lookup icon is displayed	Boolean	No	
completer	Sets the type of completer class to use (AJAXReferenceCompleter, AJAXTableCompleter)	String	No	AJAXReferenceCompleter
	Adds additional columns to the	String		

columns	search if using AJAXReferenceCompleter	(Comma-Separated)	No	
order_by	Field to sort the lookup fields by	String	No	
query	Query to apply to the lookup	String (Encoded Query)	No	

### ⊕ Returns

*None*

### Example

```
<g:ui_reference class="text"
    name="newAgendaTaskUser"
    id="newAgendaTaskUser"
```

## ui\_select\_list\_addremove\_control

Outputs a button set used for Slushbuckets

Library Glide

### ⊕ Parameters

Attribute	Description	Type	Required	Default
name	The Name of the HTML control	String	Yes	
selectable	<i>Unknown</i>			
leftzone	The name of the Left Zone Control	Yes		
rightzone	The name of the Right Zone Control	Yes		

### ⊕ Returns

*None*

### Example

```
<g:ui_select_list_addremove_control name="source_fields" selecttable="selectTableSource" leftzone="source_fields_leftzc
[<] [>]
```

## ui\_select\_option

Outputs an option for a choice list

Library Glide

### ⊕ Parameters

Attribute	Description	Type	Required	Default
text	Display value for the option	String	Yes	

text	Display value for the option	String	Yes	
value	Actual value for the option	String	No	
selected	Controls whether the option is selected	Boolean	No	

### ⊕ Returns

*None*

### Example

```
<g:ui_select_option text="My Option" value="my_option" selected="true" />
```

## ui\_slushbucket

Outputs a Slush Bucket

Library Glide

### ⊕ Parameters

Attribute	Description	Type	Required	Default
up_down	Controls whether to show the Up and Down controls	Boolean	No	

### ⊕ Returns

*None*

### Example

```
<g:ui_slushbucket up_down="false" />
```

## ui\_spacer

Outputs a spacer

Library Glide

### ⊕ Parameters

Attribute	Description	Type	Required	Default
pixels	Width in pixels, of the spacer	Integer	Yes	

### ⊕ Returns

*None*

### Example

```
<g:ui_spacer pixels="20" />
```

## ui\_table

Outputs a table wrapper

**Library** Glide

### ⊕ Parameters

Attribute	Description	Type	Required	Default
<i>Body</i>	Table Contents	Jelly/HTML	Yes	

### ⊕ Returns

*None*

### Example

```
<table>
  <tr>
    <td>
      <g:ui_header>My Form</g:ui_header>
    </td>
  </tr>
</table>
```

## ui\_theme\_changer

Outputs the Theme Changer if enabled on the Instance

**Library** Glide

### ⊕ Parameters

*None*

### ⊕ Returns

*None*

### Example

```
<g:ui_theme_changer />
```

## ui\_timezone\_changer

Outputs the Time Zone Picker

**Library** Glide

### ⊕ Parameters

*None*

 **Returns**

*None*

**Example**

```
<g:ui_timezone_changer />
```

## ui\_update\_set\_picker

Outputs the Update Set Picker

**Library** Glide

 **Parameters**

*None*

 **Returns**

*None*

**Example**

```
<g:ui_update_set_picker />
```

## update\_table\_options

Outputs an option set for tables capturable in Update Sets

**Library** Glide

 **Parameters**

*None*

 **Returns**

*None*

**Example**

```
<select>
    <g:update_table_options />
</select>
```

## useBean

Creates an instance of a JavaBeans component. Since no instances of JavaBeans are used it is unknown if this tag is usable or not.

**Library** Jelly

## ⊕ Parameters

*Unknown*

## ⊖ Returns

*None*

### Example

```
None
```

## useList

Creates a List variable from a given set of comma separated values

**Library** Jelly

## ⊕ Parameters

Attribute	Description	Type	Required	Default
var	The variable to store the result in	String	Yes	
items	An comma separated list of values	String/JEXL	Yes	

## ⊖ Returns

*None*

**Has Body**

### Example

```
<j:useList var="jvar_animals" items="${['mouse', 'cat', 'dog']}"/>
```

## vsplit

Creates a column within a two-column layout

**Library** Glide

## ⊕ Parameters

Attribute	Description	Type	Required	Default
Body	The contents of the column	Jelly/HTML/Text	Yes	

## ⊖ Returns

*None*

### Example

```
<g:vsplit>
  <h4>Column 1,1</h4>
  <p>Some content...</p>
</g:vsplit>
```

## vsplitter

Creates a two-column layout

Library Glide

### ⊕ Parameters

Attribute	Description	Type	Required	Default
<i>Body</i>	The contents of the layout	Jelly/HTML/Text	Yes	

### ⊕ Returns

*None*

## Example

```
<g:vsplitter>
  <g:vsplit>
    <h4>Column 1,1</h4>
    <p>Some content...</p>
  </g:vsplit>
  <g:vsplit>
    <h4>Column 1,2</h4>
    <p>Some content...</p>
  </g:vsplit>
</g:vsplitter>
```

## when

Nested tag within a `when` block which conditionally runs nested code if test passes

Library Jelly

### ⊕ Parameters

Attribute	Description	Type	Required	Default
<i>test</i>	Condition to test	String/JEXL	Yes	
<i>Body</i>	Block to execute if condition passes	Jelly/HTML/Text	Yes	

### ⊕ Returns

*None*

## Example

```
<j:when test="${jvar_showMessage=='Hello'}">
  Hello
</j:when>
```

## while

Loops executing the body of the tag while the test condition evaluates true

Library Jelly

### ⊕ Parameters

Attribute	Description	Type	Required	Default
test	Condition to test	String/JEXL	Yes	
Body	Block to execute while condition is true	Jelly/HTML/Text	Yes	

### ⊕ Returns

None

### Example

```
<j:set var="jvar_showMessage" value="true" />
<j:while test="${jvar_showMessage}">
    Hello World
    <j:set var="jvar_showMessage" value="false" />
</j:while>
```

## whitespace

Preserves padded whitespace for content within the scope of the tag

Library Jelly

### ⊕ Parameters

Attribute	Description	Type	Required	Default
Body	Content for which whitespace will not be trimmed	Jelly/HTML/Text	Yes	

### ⊕ Returns

None

### Example

```
<j2:whitespace>
    Not <strong>here</strong>.
</j2:whitespace>
```

## with

No Description

**Libary Glide**

**⊕ Parameters**

*Unknown*

**⊕ Returns**

*None*

**Example**

```
No documented example
```

## wizard

*No Description*

**Libary Glide**

**⊕ Parameters**

*Unknown*

**⊕ Returns**

*None*

**Example**

```
No documented example
```

## wizard\_choice\_list

*No Description*

**Libary Glide**

**⊕ Parameters**

*Unknown*

**⊕ Returns**

*None*

**Example**

```
No documented example
```

## yellow\_annotation

Outputs a Yellow Warning form annotation

## Library Glide

### ⊕ Parameters

Attribute	Description	Type	Required	Default
<i>Body</i>	Text for the annotation	String	Yes	

### ⊕ Returns

*None*

## Example

```
<g:yellow_annotation>Here is my Yellow Annotation. This may be a warning.</g:yellow_annotation>
```

# JavaScript Objects Reference

## GlideController

Initializes a GlideController context used by other Jelly Objects.

Property/Method	Description	Returns
GlideController()	Constructor for the class	
exists(String name)	Checks that a global object exists	Boolean
getCache()	<i>Unknown</i>	Object
getGlobal(String name)	Gets a global variable	Object
putGlobal(String name, Object value)	Adds a global variable	<i>None</i>
removeGlobal(String string)	Removes a global variable	<i>None</i>

## GlideEmitter

Used to execute and return Jelly to the client

Property/Method	Description	Returns
Emitter(Object <i>GlideJellyContext</i> gjc, Object <i>XMLOutput</i> output)	Constructor	
Emitter()	Constructor (empty)	
emitJelly(String script)	Runs Jelly	<i>None</i>
emitClientScriptWithEval(String javaScript, Number phase)	<i>Unknown</i>	<i>None</i>

## GlideEvaluator

Used to evaluate expressions and conditions

Property/Method	Description	Returns
GlideEvaluator()	Constructor	
evaluateStringWithGlobals(String expression, Object globals)	Evaluates an expression with variables	Object
evaluateString(String expression)	Evaluates an expression	Object
evaluateCondition(String condition)	Evaluates a condition	Boolean

## GlideJellyContext

Creates a Jelly Context (requires a GlideController)

Property/Method	Description	Returns
GlideJellyContext(GlideController gc)	Constructor	
getGlideForm()	Gets the Glide Form for the Context	Object

## GlideJellyRunner

Runs a Jelly Script and returns the result

Property/Method	Description	Returns
GlideJellyRunner(GlideController gc)	Constructor	
execute(String xml)	Executes an XML script	String
run(String script)	Alias for runFromScript	String
runFromScript(String script)	Runs a script	String
runFromTemplate(String template)	Runs a system template	String
runMacro(String name)	Runs a UI macro	String
setEscaping(boolean escaping)	Enables escaping	<i>None</i>
setRenderProperties( Object (RenderProperties) rp)	Sets the Render Properties	<i>None</i>
setTwoPhase(boolean b)	Enables second phase	<i>None</i>
setVariable(String name, Object variable)	Sets a variable and its value	<i>None</i>

## GlideListProperties

Controls properties used for rendering lists.

Property/Method	Description	Returns
GlideListProperties()	Constructor	
canChangeView()	Gets the Change View flag	Boolean
canGroup()	Gets the Group flag	Boolean
canSort()	Gets the sort flag	Boolean
getAggregates()	Gets the aggregates list	Object
getCanHideNav()	Gets the Hide Nav flag	Boolean
getCssName()	Gets the CSS name	String
getDynamicTable()	Gets the Dynamic Table property	String
getFieldList()	Gets the field list as an Array	Array
getFields()	Gets the table fields	Object
getFieldString()	Gets the fields as a string	String
getFirstRow()	Gets the first row	Number
getFixedQuery()	Gets the fixed query	String
getGrandTotalRows()	Gets the total rows	Number
getLastRow()	Gets the last row	Number
getListClass()	Gets the list class	String
getListControlID()	Gets the control ID	String
getListEditInsertRow()	Gets the List Edit Insert row flag	Boolean
getListEditTag()	Gets the list edit tag	String

getListEditType()	Gets the list edit type	String
getListID()	Gets the list ID	String
getListName()	Gets the list name	String
getOrderBy()	Gets the order by value	String
getOrderColumn()	Gets the order column	String
getParameter(String name)	Gets the specified parameter	String
getParentID()	Gets the parent ID	String
getParentTable()	Gets the parent table	String
getRelated()	Gets the Related table	String
getRelatedField()	Gets the related field	String
getRelationship()	Gets the system relationship name	String
getRowIndent()	Gets the Row Indent value	Number
getRowsPerPage()	Gets the rows per page	Number
getTable()	Gets the table name	String
getTitle()	Gets the table title	String
getTitleSingular()	Gets the table title (singular)	String
getTotalRows()	Gets the total rows	Number
getView()	Gets the view name	String
hasActions()	Gets the has actions flag	Boolean
hasBottomNav()	Gets the has bottom nav flag	Boolean
hasBottomVCR()	Gets the has bottom VCR controls flag	Boolean
hasFilter()	Gets the has filter flag	Boolean
hasHeader()	Gets the has header flag	Boolean
hasHeaderContextMenu()	Gets the has Header Context Menu flag	Boolean
hasHierarchy()	Gets the has Hierarchy flag	Boolean
hasListMechanic()	Gets the has List Mechanic flag	Boolean
hasPopup()	Gets the has Popup flag	Boolean
hasRowContextMenu()	Gets the has Row Context Menu flag	Boolean
hasSearch()	Gets the has Search flag	Boolean
hasTitle()	Gets the has Title flag	Boolean
hasTitleContextMenu()	Gets the has Title Context Menu flag	Boolean
hasTopButtons()	Gets the has Top Buttons flag	Boolean
hasTopNav()	Gets the has Top Nav flag	Boolean
hasTopNavActionButtons()	Gets the has Top Nav Action Buttons flag	Boolean
hasTopVCR()	Gets the has Top VCR flag	Boolean
init()	Initializes the object	<i>None</i>
initListID()	Generates the list ID	<i>None</i>
isCollapsible()	Gets the is Collapsible flag	Boolean

isCollapsed()	Gets the is Collapsed flag	Boolean
isEmbedded()	Gets the is Embedded flag	Boolean
isGrouped()	Gets the is Grouped flag	Boolean
isHideRows()	Gets the is Hide Rows flag	Boolean
isOmitPrintButton()	Gets the is Omit Print Button flag	Boolean
isRefList()	Gets the is Ref List flag	Boolean
isRelatedList()	Gets the is Related List flag	Boolean
isRelationship()	Gets the is Relationship flag	Boolean
isSaveFilterHidden()	Gets the is Save Filter Hidden flag	Boolean
isShow()	Gets the is Show flag	Boolean
isShowLinks()	Gets the is Show Links flag	Boolean
isShowNoRecordsMessage()	Gets the is Show No Records Message flag	Boolean
isToggleHeader()	Gets the is Toggle Header flag	Boolean
isUserList()	Gets the is User List flag	Boolean
serialize()	Serializes all the Objects properties	String
setCanChangeView(Boolean onOff)	Sets the Can Change View flag	None
setCanGroup(Boolean onOff)	Sets the Can Group flag	None
setCanHideNav(Boolean onOff)	Sets the Can HideNav flag	None
setCanSort(Boolean onOff)	Sets the Can Sort flag	None
setCollapsable(Boolean onOff)	Sets the Collapsible flag	None
setCollapsed(Boolean onOff)	Sets the Collapsed flag	None
setContextMenus(Boolean onOff)	Sets the Context Menus flag	None
setCssName(String name)	Sets the Css Name	None
setEmbedded(Boolean onOff)	Sets the Embedded flag	None
setFixedQuery(String query)	Sets the Fixed Query	None
setGrouped(Boolean onOff)	Sets the Grouped flag	None
setHasActions(Boolean onOff)	Sets the Has Actions flag	None
setHasBottomNav(Boolean onOff)	Sets the Has Bottom Nav flag	None
setHasBottomVCR(Boolean onOff)	Sets the Has Bottom VCR flag	None
setHasBreadcrumbs(Boolean onOff)	Sets the Has Breadcrumbs flag	None
setHasFilter(Boolean onOff)	Sets the Has Filter flag	None
setHasHeader(Boolean onOff)	Sets the Has Header flag	None
setHasHeaderContextMenu(Boolean onOff)	Sets the Has Header Context Menu flag	None
setHasHierarchy(Boolean onOff)	Sets the Has Hierarchy flag	None
setHasListMechanic(Boolean onOff)	Sets the Has List Mechanic flag	None
setHasPopup(Boolean onOff)	Sets the Has Popup flag	None
setHasRowContextMenu(Boolean onOff)	Sets the Has Row Context Menu flag	None

setHasSearch(Boolean onOff)	Sets the Has Search flag	None
setHasTitle(Boolean onOff)	Sets the Has Title flag	None
setHasTitleContextMenu(Boolean onOff)	Sets the Has Title Context Menu flag	None
setHasTopButtons(Boolean onOff)	Sets the Has Top Buttons flag	None
setHasTopNav(Boolean onOff)	Sets the Has Top Nav flag	None
setHasTopNavActionButtons(Boolean onOff)	Sets the Has Top Nav Action Buttons flag	None
setHasTopVCR(Boolean onOff)	Sets the Has Top VCR flag	None
setHiddenQuery(String query)	Sets the Hidden Query flag	None
setHideRows(Boolean onOff)	Sets the Hide Rows flag	None
setListEditType(String type)	Sets the List Edit Type	None
setListID(String id)	Sets the List ID	None
setListName(String name)	Sets the List Name	None
setOmitPrintButton(Boolean onOff)	Sets the Omit Print Button flag	None
setOrderColumn(String columnName)	Sets the Order Column	None
setParameter(String name, String value)	Sets a parameter value)	None
setRowIndent(int rowIndent)	Sets the Row Indent	None
setSaveFilterHidden(Boolean onOff)	Sets the Save Filter Hidden flag	None
setShowLinks(Boolean onOff)	Sets the Show Links flag	None
setShowNoRecordsMessage(Boolean onOff)	Sets the Show No Records Message flag	None
setTitle(String title)	Sets the Title	None
setToggleHeader(Boolean onOff)	Sets the Toggle Header flag	None
setUserQuery(String query)	Sets the User Query	None
setVCR(Boolean onOff)	Sets the VCR flag	None
setView(String view)	Sets the View	None

## GlideRenderProperties

Used to control rendering of and pass parameters to Jelly Pages

Property/Method	Description	Returns
GlideRenderProperties()	Constructor	
genEmptyForm()		Boolean
getEncodedQuery()	Gets the Encoded Query	String
getGaugeHeight()	Gets the gauge Height	Number
getGaugeId()	Gets the Gauge ID	String
getGaugeName()	Gets the Gauge Name	String
getGaugeType()	Gets the Gauge Type	String
getGaugeWidth()	Gets the Gauge Width	Number
getListControl()		SysListControl

getMedia()	Gets the media type	String
getParameters()	Gets the page Parameters	Object
getParameterValue(String name)	Gets the parameter value	String
getReferringURL()	Gets the referring URL	String
getViewID()	Gets the View	String
getViewManager()	Gets the View Manager	Object
getWindowID()	Gets the Window ID	String
getWindowProperties()	Gets the Window Properties	Object
isChartDetailOnReport()	Gets the Chart Detail On Report flag	Boolean
isDialog()	Gets the Dialog flag	Boolean
isHomePage()	Gets the Home Page flag	Boolean
isInteractive()	Gets the Interactive flag	Boolean
isMaintainOrder()	Gets the Maintain Order flag Returns Maintain order flag	Boolean
isManyToMany()	Gets the Many To Many flag	Boolean
isMultipleUpdate()	Gets the Multiple Update flag	Boolean
isOneToMany()	Gets the One To Many flag	Boolean
isPopup()	Gets the Popup flag	Boolean
isPortal()	Gets the Portal flag	Boolean
isPreview()	Gets the Preview flag R	Boolean
isPrint()	Gets the Print flag	Boolean
isReadOnly()	Gets the Read Only flag	Boolean
isRelatedList()	Gets the Related List flag	Boolean
isSearch()	Gets the Search flag	Boolean
isSmallCaption()	Gets the Small Caption flag	Boolean
setDialog(Boolean dialog)	Sets the Dialog flag	<i>None</i>
setListControl(Object SysListControl listControl)	Sets the List Control object	<i>None</i>
setMedia(String media)	Sets the media type	<i>None</i>
setPrint(Boolean b)	Sets the Print flag	<i>None</i>
setReadOnly(Boolean b)	Sets the Read Only Flag	<i>None</i>
setSmallCaption(Boolean smallCaption)	Sets the Small Caption flag	<i>None</i>
setView(String viewName)	Sets the view Name	<i>None</i>
setView(String viewName, String startingViewName)	Sets the view name	<i>None</i>
toString()	Serializes all properties to a string	String
useRelatedRecordManager(String type) ("mtm","otm")	Gets the Related Record Manager flag	Boolean
useSlushbucket(String type) ("mtm","otm")	Gets the Slushbucket Flag	Boolean

# Licenses and Copyrights

---

## ServiceNow

ServiceNow, its associated Glide Libraries, all code and all respective Trademarks are property of ServiceNow, Inc. Your use of ServiceNow implies acceptance with their Terms of Use and License.

## Jelly

Copyright (c) The Apache Foundation

Apache License, Version 2.0 Apache License Version 2.0, January 2004 <http://www.apache.org/licenses/>

### TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

#### 1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by

Licensor and subsequently incorporated within the Work.

1. Grant of Copyright License.

Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

1. Grant of Patent License.

Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

1. Redistribution.

You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

You must give any other recipients of the Work or Derivative Works a copy of this License; and You must cause any modified files to carry prominent notices stating that You changed the files; and You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License. You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

1. Submission of Contributions.

Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions.

Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

1. Trademarks.

This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

1. Disclaimer of Warranty.

Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

#### 1. Limitation of Liability.

In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

#### 1. Accepting Warranty or Additional Liability.

While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

### END OF TERMS AND CONDITIONS

### APPENDIX: How to apply the Apache License to your work

To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "[]"  
replaced with your own identifying information. (Don't include the brackets!) The text should be enclosed in the appropriate  
comment syntax for the file format. We also recommend that a file or class name and description of purpose be included on  
the same "printed page" as the copyright notice for easier identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the  
License. You may obtain a copy of the License at

<http://www.apache.org/licenses/LICENSE-2.0>

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS"  
BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the  
specific language governing permissions and limitations under the License.

## Twitter Bootstrap

The MIT License (MIT)

Copyright (c) 2011-2015 Twitter, Inc

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated  
documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use,  
copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the  
Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

## DataTables

Copyright (C) 2008-2015, SpryMedia Ltd.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

## Raphael

Copyright © 2008 Dmitry Baranovskiy

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

## justGage

Copyright Bojan Djuricic (@Toorshia)

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING

BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.