	Prototype JS		jQuery	
	Command / Function	Description	Command / Function	Description
Selecting DOM elements	**$(** elementId | element)	• If a string is passed to the dollar-function, it will return the DOM element with the matching ID attribute. • If an HTML element is passed, an extended version of the element is returned. • It is also possible to pass an array of elementIds or elements to the $-selector.	**$(** selector | element) or jQuery(selector [element)	The jQuery-dollar-function accepts a string containing a CSS selector which is then used to match a set of elements. • selector: If a selector string is passed to the dollar-function, it will return the DOM elements which match the given CSS-selector. • element: If an HTML element is passed, an extended version of the element is returned. The main difference is between the Prototype-dollar-function and the jQuery counterpart is, that the jQuery-version expects and CSS-selector - while the Prototype-version requires and element-id. So the jQuery-equivalent to the dollar-function whould rath be the Prototype-dollour-dollar-function (-> $$(selector)). **Note:** Because this function is the one that is used the most in host frameworks, this will be the parts of your JavaScript that will need the most attention during a migration. • A Prototype-based function which depends on code like this"var targetElement = **$('searchresultscontainer')**;" WILL certainly fail when using jQuery - even though the syntax and signature of both versions of the dollar-function is quite similar. The jQuery-variant needs to look like this to work: var targetElement = **$('#searchresultscontainer')**; The missing '#' in $-function calls is bound to be biggest source of bug during a migration process from Prototype JS to jQuery.
	$$(selector)	Takes an arbitrary number of CSS selectors and returns a document-order array of extended DOM elements that match any of them.	The task of selecting HTML elements which CSS selectors is performed by the simple dollar-function: $(selector) Visit these pages for more information about CSS selectors: • http://www.w3.org/TR/CSS2/selector.html • CSS Cheat Sheet (V2): http://www.addedbytes.com/cheat-sheets/css-cheat-sheet/ • CSS Selector Tutorial: http://css.maxdesign.com.au/selectutorial/	
	$(element**).select(** selector)	The select-function takes one or more CSS selectors and allows you to select HTML elements which are sub-nodes of the calling DOM element.	$(element).find(selector)	Gets the descendants of each element in the current set of matched elements, filtered by a selector. • This function works alike Prototype's select-function and can therefore easily be migrated with a simple search and replace.
Get / Set the content of DOM elements	**$F(** element) or **$(**inputelement**).getValue()**	Returns the value of a form control (textbox, textarea, radiobutton, checkbox, ...).	$('selector').**val()**	Returns the value of the calling element.
	$(element).**innerHTML**	Returns HTML code of all text nodes and child elements of the parent DOM node. The property can also be used to set the contents of an HTML element. But note, this property is not specific to the Prototype framework - but just a standard property of all HTML nodes.	$(element).html()	Gets the HTML source ode of the calling HTML element.
	$(element).**update(** text Or HTML code)	Sets the HTML code of the calling element.	$(element).html(text Or HTML code)	Sets the HTML source ode of the calling HTML element.
	$(element).innerHTML **.stripTags()** or $(element).**innerText**	Returns the contents of the calling element: stripped from any HTML tags.	$(element).text()	Returns the combined text contents of the calling element, including their descendants.
Update the Style of DOM elements	$(element).**setStyle(** style(s))	The Prototype setStyle functions modifies the CSS style properties of the calling HTML element.	$(element).css(propertyName, value)	Sets the value of a HTML element's style property. Example: • $(selector).css('backgroundColor', 'red')
	$(element).**getStyle(** propertyName)	Returns the given CSS property value of the calling HTML element.	$(element).css(propertyName)	Get the value of the calling HTML element's style property. Example: • $(element).css('backgroundColor') // -> 'red'
	$(element).**classNames()**	Returns a set of CSS class names assigned to the calling HTML element (e.g. ["class1", "class2", "class3"]).	$(element).attr('className')	The list of all CSS classes assigned to an HTML element can only be determined by reading the className-attribute. But unlike the Prototype-version, this approach will only return a simple string list of all CSS class names assigned to an HTML node (e.g. "class1 class2 class3").
	$(element).**hasClassName(** className)	Checks whether the calling element has the given CSS className.	$(element).hasClass(className)	Determine whether any of the matched elements are assigned the given class. The hasClassName function works exactly alike in both frameworks. The only difference is the slightly shorter name of the jQuery version.
	$(element).**toggleClassName(** className)	Toggles an element's className: If the element did not have the CSS class before - it will be assigned; otherwise it will be removed from the HTML element.	$(element).toggleClass(className)	Adds or removes one or more CSS class names for each element in the set of matched elements, depending on whether the element had the class name before or not. The toggleClassName function works exactly alike in both frameworks. The only difference is the slightly shorter name of the jQuery version.
	$(element).**addClassName(** className)	Adds the specified class name to the calling HTML element.	$(element).addClass(className(s))	Adds the specified class-name(s) to each element of the set of matched elements. The addClassName function works exactly alike in both frameworks. The only difference is the slightly shorter name of the jQuery version.
	$(element).**removeClassName(** className)	Removes the specified class name from the calling HTML element, if it was assigned before.	$(element).removeClass(className(s))	Removes the specified class-name(s) from each element in the set of matches elements. The removeClassName function works exactly alike in both frameworks. The only difference is the slightly shorter name of the jQuery version.
Get/set DOM element attributes	$(element).**readAttribute(** attributeName)	Returns the value of calling element's attribute or null if attribute name does not exist.	$(element).attr(attributeName)	Gets the value of an attribute for the first element in the set of matched elements.
	$(element).**writeAttribute(** attribute(s))	Adds, specifies or removes attributes to HTML elements.	$(element).attr(attributeName, value)	Creates a new element attribute and/or sets the value for the given attribute. The only difference is that the jQuery .attr() function is not able to remove attributes from HTML elements. For this task you must use the specialized .removeAttr() function.
	$(element).**getWidth**() $(element).**getHeight**()	Returns the width / height (number of pixels) of the calling HTML element.	$(element).width() $(element).height()	Get the current computed width / height for the first element in the set of matched elements.
	Prototype JS has no core functionality which allows you to set the width or height of HTML elements.		$(element).width(value) $(element).height(value)	Sets the width or height of the calling HTML element.
	$(element).**empty**()	Returns true if the element does not contain any characters (except whitespaces); otherwise false.	$(element).is(':empty')	Utilizes the .is() selector for checking if the current element has no child nodes (including text nodes). Returns true if the current element is empty; otherwise false.
Events	Event.**observe**(element, eventName, handler)	Registers an event handler on a DOM element.	$('selector').bind(eventType, handler)	Attach a event-handler to the set of matched elements. jQuery offers a long list of explicit event-functions in addition to the abstract bind-function for DOM-events. Examples: • click(): OnClick event • dbiClick(): Double-Click • hover(): Binds two event handlers to the matched elements, to be executed when the mouse pointer enters and leaves the elements. • keydown(): Is triggered when a key is pressed.
	Event.**stopObserving**(element, eventName, handler)	Unregisters one or more event handlers from the given element.	$(selector).unbind(eventtype, handler)	Removes a previously-attached event handler from the set of matched DOM-elements.
	document.observe('**dom:loaded**', function() { });	The dom:loaded event is fired immediately after the HTML document is fully loaded.	$(document).**ready()**	The document-ready event is fired as soon as the whole DOM has been assembled. I think the jQuery-version is much more reliable than the Prototype approach.
	Event.**findElement**(event, tagName)	Returns the element that fired the event.	event.target Example: • $("#button").click(function(e) { var btn = **$(event.target)**; });	The DOM element that initiated the event. Another way to access the element that caused the event is this: • $("#button").click(function(e) { var btn = **$(this)**; });
Ajax	new Ajax.**Request**(url[, options])	Creates and processes and AJAX request to the specified url.	$.ajax(settings)	Performs an asynchronous HTTP request (ajax).

	new Ajax.**PeriodicalUpdater**(container, url[, options])	Periodically performs an AJAX request and updates a container's contents based on the response text.	There is no PeriodicalUpdater for jQuery, but you can build one yourself.
			
			var periodicalUpdater = function() {
			$.ajax({
			url: 'http://www.example.com/api/get/status.html',
			dataType: 'text',
			success: function(code) {
			$("#statusContainer").html(code);
			}
			});
			};
			
			setInterval(periodicalUpdater, 1000);	

	new Ajax.**Updater**(container, url[, options])	Updates the given container element with response text from the ajax-request to the specified url.	There is no Ajax.Updater for jQuery, but you can easily build one yourself:
			
			$.ajax({
			url: 'http://www.example.com/api/get/content.html',
			dataType: 'text',
			success: function(code) {
			$("#container").html(code);
			}
			});	

Create new DOM elements	**new Element**(tagName[, attributes])	Create a new DOM element (just like with document.createElement('div')).	There is no direct equivalent for the Prototype "new Element()"-command within the jQuery Core, but you can use a simple document.createElement(tagName) instead or use the following jQuery plugin: http://blogs.microsoft.co.il/blogs/basil/archive/2008/08/21/jquery-create-jquery-plug-in-to-create-elements.aspx Or you can simply create a DOM element by passing the html-code: • var link = $("<a>", { class: "link", text: "New Link", href: "http://api.jquery.com/jQuery/" }); • var link = $("");	
	Element.**insert**(element, { position: content })	Insert the content object before, after, at the top of, or at the bottom of element.	Instead of a single function for all purposes (inserting elements before or after a given element) jQuery has one specialized function for every single purpose: $(element).append() $(element).appendTo() $(element).prepend() $(element).prependTo() $(element).before() $(element).after() $(element).insertBefore() $(element).insertAfter() Even though I think the jQuery versions of "Element.insert()" are more versatile, I like the "simpler" Prototype-version better.	append(content): Inserts the content pointes as the last child of each element in the set of matched elements. appendTo(target): Inserts every element in the set of matches elements to the target element. prepend(content): Inserts the content element to the beginning of each element in the set of matched elements. prependTo(target): Insert every element in the set of matched elements to the beginning of the target element. before(target): Insert the content element before each element in the set of matched elements. after(target): Insert the content element after each element in the set of matched elements. insertBefore(target): Insert every element in the set of matched elements before the target element. insertAfter(target): Insert every element in the set of matched elements after the target element.
	$(element).**identify**()	Gets the element's id if it is assigned; otherwise a new id is generated, assigned and returned.	I guess because jQuery does not have a core function for creating new DOM elements, there was no need for a function which automatically assigns new ids to them. Workarounds: • Get id attribute with jQuery: $(element).**attr('id')** • Set id attribute with jQuery: $(element).attr('id', 'YourCustomId')	
Forms	Form.Element.**disable**()	Disables a form control - prevents user from editing the form elements.	There is no direct equivalent to Prototypes "Form.Element.disable()" in the jQuery core, but you can easily prohibit the editing of form elements with only very little code: • $('form:first').**attr("disabled", "disabled")** • $(formElement).find('input, textarea').each(function() { $(this).attr("readonly", "readonly"); });	
	Form.Element.**enable**()	Enables a previously disabled form control.	There is no direct equivalent to Prototypes "Form.Element.disable()" in the jQuery core, but you can easily allow the editing of form elements with only very little code: • $('form:first').**removeAttr("disabled")** • $(formElement).find('input, textarea').each(function() { $(this).**removeAttr**("readonly"); });	
Traversing Arrays, Sets and Collections	$$(selector).**each**(function(s, i) { var currentElement = s; })	A handy notation for the well known for-loop. The each-operator iterates over all members of the given set of matches elements.	.each(function(index, Element))	Iterate over a jQuery object, executing a function for each matched element. The syntax of the each-iterator is exactly alike in both frameworks, but can also be used slightly different in jQuery: • $(selector).each(function(){ var currentElement = **$(this)**; });
Miscellaneous functions & utilities	new **PeriodicalExecuter**(function(pe) { ... }, interval)	Executes a given function every n-seconds.	Again, there is no direct substitue in jQuery for a periodical updater, but it can be build quite easilty (without even depending on jQuery). Example of a simple (non-Query-specific) periodical-updater (*1000 milliseconds = 1 second): • var periodicalExecuter = function() { ... }; **setInterval**(periodicalExecuter, 1000);	