# How should GetElementById work in Caja?

Document.getElementById is one of the most-used functions in the JavaScript client side API. It is also one of the least object-capability-like operations. In caja, few objects should have access to the whole document. The simplistic solutions are that getElementById must be shut off (which is a severe usability/functionality problem) or it must be turned on (which is a severe security breach). This is an analysis of alternative implementations of functions that achieve results similar to those of Document.getElementById in an object-capability secure fashion.

One feature that all the options have in common is that a new function Node.getElementById has been created. This version of getElementById searches the subtree of the node for the id. Hence nodes can be used to convey authority usefully over parts of the document: if a module is handed a node, the module has authority to access all the nodes in that subtree, and also has a getElementById function for searching that area.

In this analysis, a strong concern is the question of, what happens when data is being shared by several mutually suspicious modules? Options that might get an Excellent rating if there were only one programmer for the page may get surprising poor marks on many criteria because the question lurks all the time, what happens when there are three people working in exclusive subparts of the page, or even more interesting, sharing parts of the page.

## Executive Summary

Based on a careful evaluation of how well each of the 7 possible choices could meet the 11 major criteria considered, GetElementsById appears to be the best choice.

The 7 choices considered were:

GetElementsById

Namespace arg in GetElem, SetHTML, SetAttr

DoubleDiv starts new namespace

MMU, All IDs Purely Unique

Caller-Relative namespace

Grantor of node rewraps for new namespace

Direct Usage from Node

The criteria used to evaluate the options were (in order of importance):

Fail Safe

Possibility of Implementation

Can be implemented to run fast

Naturalness of Evolution To Cooperative Activity

Surprisingness/Debugging Difficulty for Failures

Ease of Explanation

Naturalness of Operation

Low Probability of Accidental Failure/Does Not Require Emergence of Programmer NameSpacing

Allows ID-based CSS rules

Low Probability of Malicious Failure

Difficulty/Risk in Implementation

Of all of the choices considered, 4 were considered to be leading candidates.  These "top options" were:

GetElementsById

Namespace arg in GetElem, SetHTML, SetAttr

DoubleDiv starts new namespace

MMU, All IDs Purely Unique

Naturalness of Operation was the most significant factor leading to the choice of GetElementsById over Namespace arg in GetElem, SetHTML, SetAttr.

Naturalness of Operation was the most significant factor leading to the choice of GetElementsById over DoubleDiv starts new namespace.

Naturalness of Evolution To Cooperative Activity was the most significant factor leading to the choice of GetElementsById over MMU, All IDs Purely Unique.

## Introduction

The question of "How should GetElementById work in Caja?" was evaluated by means of a decision table.

| | Fail Safe | Possibility of Implementation | Can be implemented to run fast | Naturalness of Evolution To Cooperative Activity | Surprisingness/Debugging Difficulty for Failures | Ease of Explanation | Naturalness of Operation | Low Probability of Accidental Failure/Does Not Require Emer | Allows ID-based CSS rules | Low Probability of Malicious Failure | Difficulty/Risk in Implementati | Summary |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| GetElementsById | Yes | Yes | Good | Good | Excellent | Excellent | Excellent | Fair | Excellent | Poor | Good | Excellent |
| Namespace arg in GetElem, SetHTML | Yes | Yes | Excellent | Excellent | Excellent | Good | Fair | Excellent | Poor | Excellent | Good | Excellent |
| DoubleDiv starts new namespace | Yes | Yes | Excellent | Fair | Excellent | Good | Fair | Fair | Excellent | Good | Good | Excellent |
| MMU, All IDs Purely Unique | Yes | Yes | Excellent | Poor | Good | Fair | Excellent | Excellent | Poor | Excellent | Good | Excellent |
| Caller-Relative namespace | Yes | No | Excellent | Excellent | Excellent | Good | Excellent | Excellent | Fair | Excellent | N/A | Good |
| Grantor of node rewraps for new name | Yes | Yes | Good | Fair | Fair | Fair | Fair | Fair | Poor | Excellent | Good | Good |
| Direct Usage from Node | No | Yes | Excellent | Fair | Poor | Excellent | Excellent | Fair | Excellent | Poor | Excellent | Good |

Decision Table for How should GetElementById work in Caja?

Alternative choices considered are listed down the left side of the table.  The criteria used to evaluate the various options are listed along the top.  Initially entered in no particular order, both the choices and the criteria were then repositioned according to importance of criteria and effectiveness of individual choices in meeting them.

As criteria are evaluated and weights assigned according to which factors are considered to be most significant, the factors are sorted from left to right in order of importance (i.e., the factor considered by the decision maker to be most significant in meeting overall needs ends up in the leftmost position).

Similarly, as choices are evaluated according to effectiveness in meeting criteria, the best choices migrate to the top of the list.  When the process is complete, the best choice should emerge at the top.

As selection alternatives and the criteria to be used in evaluating them are entered into the table, weights are assigned to each of the evaluation factors so that they are ranked in order of their importance in fulfilling the overall task.

For the decision "How should GetElementById work in Caja?," the criteria used to evaluate the choices, and their weightings, were:

Fail Safe - High

Possibility of Implementation - High

Can be implemented to run fast - High

Naturalness of Evolution To Cooperative Activity - High

Surprisingness/Debugging Difficulty for Failures - Medium

Ease of Explanation - Medium

Naturalness of Operation - Medium

Low Probability of Accidental Failure/Does Not Require Emergence of Programmer NameSpacing - Low

Allows ID-based CSS rules - Low

Low Probability of Malicious Failure - Low

Difficulty/Risk in Implementation - Low

Among the 7 choices considered, 4 were considered to be "top options."  (A top option is defined as follows:  If the choice immediately following the preferred choice is rated in the same rating category as the recommended selection, then all choices in that category are considered top options.  If the second ranking choice is in a different category, the top options are considered to be the recommended choice plus all choices in the same category as the second-place option.  Thus, the "top options" list will always have at least two choices in it and may include all of the choices considered in the entire table.)

For the decision of "How should GetElementById work in Caja?," the top options were:

GetElementsById

Namespace arg in GetElem, SetHTML, SetAttr

DoubleDiv starts new namespace

MMU, All IDs Purely Unique

## Discussion of Requirements

The criteria used in this decision making process were:

### Fail Safe (overall importance: High)

If it does not fail safe, the security will be breached, violating a principle goal of caja.

### Possibility of Implementation (overall importance: High)

If you can't implement it, it is useless. Hence this gets the highest rating.

### Can be implemented to run fast (overall importance: High)

While not quite as important as failing safe, good performance is crucial.

### Naturalness of Evolution To Cooperative Activity (overall importance: High)

### Surprisingness/Debugging Difficulty for Failures (overall importance: Medium)

### Ease of Explanation (overall importance: Medium)

### Naturalness of Operation (overall importance: Medium)

### Low Probability of Accidental Failure/Does Not Require Emergence of Programmer NameSpacing (overall importance: Low)

### Allows ID-based CSS rules (overall importance: Low)

### Low Probability of Malicious Failure (overall importance: Low)

As long as the result of a malicious failure is only a denial of service because it fails safe, there are so many ways malice can cause similar DOS failures that this is unimportant.

### Difficulty/Risk in Implementation (overall importance: Low)

Google has the resources to implement even a difficult solution to this problem because it is so crucial. Even if breachable bugs are introduced because of the complexity of the solution, it is an acceptable solution to issue a patch and correct the problem after deployment. Hence the difficulty/risk is unimportant

## The Top Choices

The decision-making process has identified 4 of the choices as "top options."  They were:

### GetElementsById

2 functions are attached to the Node: getElementById(id), which throws an exception if there is more than one node with the id, and getElementsById(id), which returns an array with all the matching nodes.

### Namespace arg in GetElem, SetHTML, SetAttr

Each module is assigned an unmentionable namespace by the container. IDs written by modules are transformed during writing to be a concatenation of the id with the namespace. SetAttr and setInnerHTML must receive a namespace; Node.getElementById may receive an optional namespace. The getElementById algorithm is: if there is a namespace arg, then first search the Node's subtree for id+namespace. This allows the creator of the node to reliably find the id that he himself wrote. If such a node is not found (or no namespace is given), search for the id without a namespace (the form of the id if the Container wrote it). If this is not found, search for (any namespace)+id. If 2 or more are found, throw an exception, if there is only one, return it.

### DoubleDiv starts new namespace

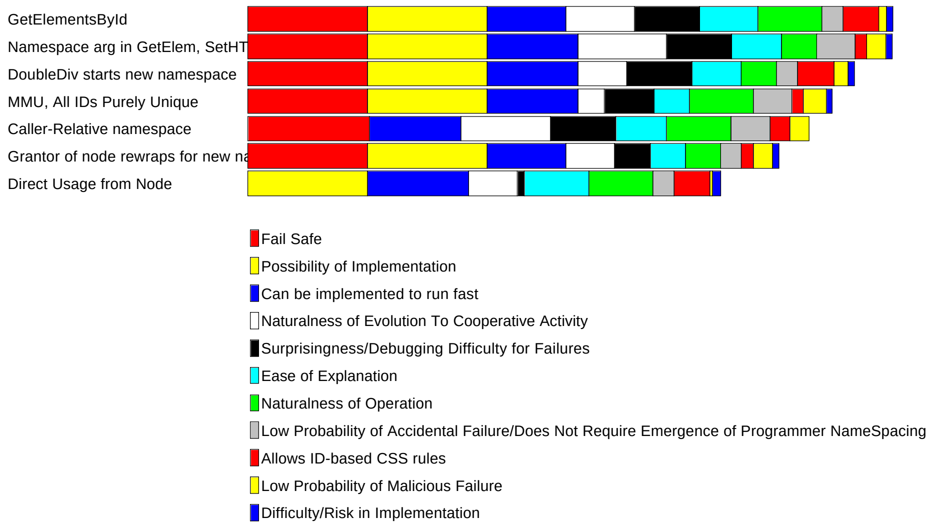
The DoubleDiv, which is used in the tamed API for controlling real-estate layout, is also used to start a new namespace area.

### MMU, All IDs Purely Unique

Every id that is created is associated with an unmentionable name and put in a map for the module assigning the id. If the module that wrote the id looks for the id, the id is looked up in the map, and the module receives the node. No other module can find the node by name; all searching of nodes in a shared context require object-oriented reference-passing, and getElementById is inoperable in this context.

## Comparisons among Choices

Relative strengths of the various choices in each of the factors is illustrated in the following graph:



Relative Strengths

### GetElementsById versus Namespace arg in GetElem, SetHTML, SetAttr

GetElementsById was considered to be a better choice than Namespace arg in GetElem, SetHTML, SetAttr in 4 of the 11 criteria considered.  Of these, the critical  factors were:

Naturalness of Operation

Allows ID-based CSS rules

Ease of Explanation

### Namespace arg in GetElem, SetHTML, SetAttr versus DoubleDiv starts new namespace

Namespace arg in GetElem, SetHTML, SetAttr was considered to be a better choice than DoubleDiv starts new namespace in 3 of the 11 criteria considered.  Of these, the critical factor was:

Naturalness of Evolution To Cooperative Activity

### DoubleDiv starts new namespace versus MMU, All IDs Purely Unique

DoubleDiv starts new namespace was considered to be a better choice than MMU, All IDs Purely Unique in 5 of the 11 criteria considered.  Of these, the critical  factors were:

Allows ID-based CSS rules

Naturalness of Evolution To Cooperative Activity

Surprisingness/Debugging Difficulty for Failures

The reason DoubleDiv starts new namespace received a rating of Fair for Naturalness of Evolution To Cooperative Activity was:  Double div suddenly becomes necessary for purposes other than real estate management when going from 1 person to 2

## Conclusion

After a careful evaluation of each option, GetElementsById appears to be the best choice.

## Appendix I:  Discussion of Options

### GetElementsById

2 functions are attached to the Node: getElementById(id), which throws an exception if there is more than one node with the id, and getElementsById(id), which returns an array with all the matching nodes.

### Namespace arg in GetElem, SetHTML, SetAttr

Each module is assigned an unmentionable namespace by the container. IDs written by modules are transformed during writing to be a concatenation of the id with the namespace. SetAttr and setInnerHTML must receive a namespace; Node.getElementById may receive an optional namespace. The getElementById algorithm is: if there is a namespace arg, then first search the Node's subtree for id+namespace. This allows the creator of the node to reliably find the id that he himself wrote. If such a node is not found (or no namespace is given), search for the id without a namespace (the form of the id if the Container wrote it). If this is not found, search for (any namespace)+id. If 2 or more are found, throw an exception, if there is only one, return it.

Namespace arg in GetElem, SetHTML, SetAttr was judged Fair in the matter of Naturalness of Operation. It would take about 4 minutes to get use to passing the extra argument, in an hour you'd not even remember doing it another way. While the namespace arg is required when writing ids (with setAttr or setInnerHTML), when doing getElementById the namespace is optional; it not present, the function looks for the id without a namespace (as would have been created by the container), and if not found, the search would continue for id with any namespace.

### DoubleDiv starts new namespace

The DoubleDiv, which is used in the tamed API for controlling real-estate layout, is also used to start a new namespace area.

DoubleDiv starts new namespace was judged Fair in the matter of Naturalness of Evolution To Cooperative Activity. Double div suddenly becomes necessary for purposes other than real estate management when going from 1 person to 2

### MMU, All IDs Purely Unique

Every id that is created is associated with an unmentionable name and put in a map for the module assigning the id. If the module that wrote the id looks for the id, the id is looked up in the map, and the module receives the node. No other module can find the node by name; all searching of nodes in a shared context require object-oriented reference-passing, and getElementById is inoperable in this context.

MMU, All IDs Purely Unique was judged Fair in the matter of Ease of Explanation. Hard to explain for programmer about to start sharing cooperatively.

### Caller-Relative namespace

Each module is associated with a namespace, and when the module invokes Node.getElementById, we look at which module is making the call to determine which namespace is associated with it, and use the getElementById algorith described earlier for the "namespace arg" option. The caller-relative namespace has the advantage that the namespace does not have to be explicitly passed as an arg.

Caller-Relative namespace was judged Excellent in the matter of Naturalness of Operation. Operation is exactly the same as normal javascript, with or without multiple sharers

Caller-Relative namespace was judged N/A in the matter of Difficulty/Risk in Implementation. already listed as not implementable.

### Grantor of node rewraps for new namespace

Calls to raw Node.getElementById use simple ids. If you want to grant someone the authority to edit nodes in a subtree, but you want to ensure his id names do not collide with yours, first wrap the node with a wrapper that has a namespace. The wrapper applies the namespace to each id created with it.

Grantor of node rewraps for new namespace was judged Fair in the matter of Naturalness of Evolution To Cooperative Activity. Rewrapping suddenly becomes important when you go from having one person working a subtree to 2 people working the subtree

Grantor of node rewraps for new namespace was judged Fair in the matter of Ease of Explanation. Requires explanation when programmer goes to start sharing cooperatively.

### Direct Usage from Node

This implements a straight Node.getElementById that is identical to Document.getElementById, but only returns nodes found in the subtree. If there are two nodes with the same id, a random one is returned based on subtleties of the implementation of the API on the particular client.

## Appendix II: References