BlastLogic REST API - Script Charge Rule V2 API

- Overview
- Functions
- Object Interfaces
  - Interval Object
  - Deck Object
  - Deck List Object
  - Hole Object
  - Parameters Object
  - Hole Charge Plan Object
  - Load Table Object
  - Blast Product Object
  - Built-in Blast Product Object
  - Blast Product Family Object

Overview

This document describes the APIs available for use within the JavaScript function that executes the charge rule.

A script for a charge rule of type `script_v2` is a JavaScript function that takes three parameters: a Parameters Object, a Hole Object and a Built-in Blast Product Object (optional).
The JavaScript function is called once for every hole the charge rule is applied to and is responsible for assigning a blast product for every part of the hole.

For example:

```
function rule(params, hole, products)
{
  // Put 5m of stemming in top of hole and fill the rest with explosives.
  hole.fill(hole.upper(5), params.stemming);
  hole.fill(hole.unallocated(), params.explosive);
}
```

Functions

The scripting API provides some builtin functions that can be used in your scripts.

Function	Returns	Description
`isUndefined(`*x*`)`	`bool`	Query if a value is undefined.
`isNumber(`*x*`)`	`bool`	Query if a value is a number. Can be NaN or +/- Infinity or an ordinary number.
`isNaN(`*x*`)`	`bool`	Query if a value is the NaN (not a number) numeric value.
`isInteger(`*x*`)`	`bool`	Query if a value is an integer.
`isClose(`*x, y, tolerance*`)`	`bool`	Query if two numeric values are within a specified tolerance of each other. ie. Math.abs(x - y) < tolerance
`fromCelsiusToKelvin(`*t*`)`	`number`	Convert a number from degrees Celsius to Kelvin. New in BlastLogic 2.1.

`fromKelvinToCelsius(`*`t`*`)`	`number`	Convert a number from Kelvin to degrees Celsius. New in BlastLogic 2.1.
`fromFahrenheitToKelvin(`*`t`*`)`	`number`	Convert a number from degrees Fahrenheit to Kelvin. New in BlastLogic 2.1.
`fromKelvinToFahrenheit(`*`t`*`)`	`number`	Convert a number from Kelvin degrees Fahrenheit. New in BlastLogic 2.1.
`message(`*`msg`*`[, `*`params`*`])`	`undefined`	Output an informational message when running the script. Execution of the script continues after calling this function. Messages prefixed with '@' are not displayed by the Tablet client. Messages prefixed with '$' will always be shown by the Tablet client. 'msg' must be a string. 'params' is an optional list of parameters to be substituted using MDF text style placeholders like %s (string), %f (number), %b (bool).
`error(`*`msg`*`[, `*`params`*`])`	`undefined`	Raise an error, aborting further execution of the script. This will cancel calculating of the charge plan for the current hole. 'msg' must be a string. 'params' is an optional list of parameters to be substituted using MDF text style placeholders like %s (string), %f (number), %b (bool).
`assert(`*`condition`*`[, `*`msg`*`])`	`undefined`	Check that `condition` evaluates to true. Raises an `AssertionError` exception if `condition` is false. 'msg' is optional but if provided must be a string.

Object Interfaces

Interval Object

An interval is an ordered list of non-overlapping parts of a hole.

- an empty interval indicating no parts of the hole (length/span = 0, top/bottom=undefined)
- an interval corresponding to a single part of the hole (length = span = bottom - top)
- an interval corresponding to multiple parts of the hole, having several sub-intervals (length < span, span = bottom - top)

The interval object has the following properties/methods:

Member	Type	Description
`interval``.top`	float	The position of the top of the top-most interval in the list. May be `undefined` if the interval list is empty.

interval.bottom	float	The position of the bottom of the bottom-most interval in the list. May be `undefined` if the interval list is empty.
interval.length	float	The sum of the lengths of all intervals in the list.
interval.span	float	The span of the interval list in the hole. Equal to `intervals.bottom - intervals.top`.
interval.intervals	array of Interval	Returns an array of Interval values, one for each sub-interval in this interval list. The order of the items in the list is from top-most to bottom-most. These objects only support `.top`, `.bottom` and `.length` members.
interval.upper(*length*)	Interval	Returns an Interval that corresponds to the upper-most 'length' parts of this interval list, splitting an interval if necessary. If 'length' is longer than this interval list then it just returns this interval list.
interval.lower(*length*)	Interval	Returns an Interval that corresponds to the lower-most 'length' parts of this interval list, splitting an interval if necessary. If 'length' is longer than this interval then it just returns this interval list.
interval.above(*depth*)	Interval	Returns an Interval that corresponds to the parts of the interval that are above the specified depth.
interval.below(*depth*)	Interval	Returns an Interval that corresponds to the parts of the interval that are below the specified depth.
interval.between(*top*, *bottom*)	Interval	Returns an Interval that corresponds to the parts of the interval list that are between the specified top/bottom depths.
interval.intersectionWith(*other*)	Interval	Intersects this interval list with another interval list, returning an interval list that corresponds to the overlapping parts of the two interval lists.
interval.unionWith(*other*)	Interval	Unions this interval list with another interval list, returning an interval list that covers both interval lists. Overlapping intervals are merged.
interval.nth(*n*)	Interval	Returns the nth sub-interval of this interval. The 0th interval is the top-most sub-interval, 1st interval is the second interval from the top, etc. Pass negative numbers to specify intervals from the bottom. eg -1 indicates bottom sub-interval, -2 indicates second-lowest sub-interval, etc. New in BlastLogic 1.1
interval.except(*other*)	Interval	Returns an Interval that corresponds to the parts of this interval that do not overlap the other interval. New in BlastLogic 1.1

interval.inverse()	Interval	Returns an Interval that is the complement of this interval. ie. An interval such that `interval.inverse().intersectionWith(interval)` is the empty interval and `interval.inverse().unionWith(interval)` is the infinite interval. New in BlastLogic 1.1
interval.expand(*length*)	Interval	Equal to *interval*.expandAbove(*length*).expandBelow(*length*). New in BlastLogic 1.1 Changed in BlastLogic 2.0 to shrink intervals when passed negative lengths.
interval.expandAbove(*length*)	Interval	Return a new interval that has each sub-interval expanded to have the sub-interval's 'top' set to 'length' above the existing 'top' value. Sub-intervals that would now be overlapping are merged. Passing a negative length causes each sub-interval to shrink rather than expand. Sub-intervals that have a length less-than or equal to the passed length will be removed. New in BlastLogic 1.1 Changed in BlastLogic 2.0 to shrink intervals when passed negative lengths.
interval.expandBelow(*length*)	Interval	Return a new interval that has each sub-interval expanded to have the sub-interval's 'bottom' set to 'length' below the existing 'bottom' value. Sub-intervals that would now be overlapping are merged. Passing a negative length causes each sub-interval to shrink rather than expand. Sub-intervals that have a length less-than or equal to the passed length will be removed. New in BlastLogic 1.1 Changed in BlastLogic 2.0 to shrink intervals when passed negative lengths.

Deck Object

(New in BlastLogic 2.1 Update 2)

A deck is part of the hole defined by an interval and a blast product. A deck can contain no blast product, denoting that it is yet to be allocated,

The deck object has the following properties/methods:

Member	Type	Description
deck.interval	Interval	The interval object for this deck that defines the top, bottom and length of this deck.
deck.product	Blast Product	The blast product for this deck. If the deck has yet to be assigned a blast product this property will have the value `null`.

deck.quantity	float	The quantity of packaged products to be loaded into this deck. This will return NaN if no blast products have been assigned or none of the decks use packaged products.
deck.mass	float	The mass of the deck in kilograms. If the product is undefined or if the mass could not be calculated (eg. due to an unknown hole diameter) then this will return NaN.
deck.volume	float	The volume of this deck in cubic metres. If the diameter of the hole is NaN, this will also return NaN.

Deck List Object

(New in BlastLogic 2.1 Update 2)

A deck list is an ordered list of non-overlapping decks in a hole. A deck list can contain:

- an empty deck indicating no parts of the hole (interval=undefined)
- a deck corresponding to a single part of the hole (length = span = bottom - top), a blast product (may be undefined if not yet allocated)
- a set of decks corresponding to multiple parts of the hole (length < span, span = interval.bottom - interval.top), each having their own blast product.

The deck list object has the following properties/methods:

Member	Type	Description
deckList.interval	Interval	The union of all deck intervals for a deck list.
deckList.decks	Deck	Returns an array of Deck values, one for each deck in this deck list. The order of the items in the list is from top-most to bottom-most. The intervals of decks in the list are guaranteed to be disjoint. ie. For any two distinct decks in the list, a and b it is guaranteed that `a.interval.intersectionWith(b.interval).length == 0`
deckList.decksAbove(*depth*)	Deck List	The `depth` parameter is specified in metres measured along the hole from the collar. If the depth is at the intersection between two decks, both decks will be included. Returns a deck list that corresponds to the decks in the hole that have any part that sits above the supplied depth. ie. that have `deck.interval.top < depth`
deckList.decksBelow(*depth*)	Deck List	The `depth` parameter is specified in metres measured along the hole from the collar. If the depth is at the intersection between two decks, only the lower of the two decks will be included. Returns a deck list that corresponds to the decks in the hole that have any part that sits below the supplied depth. ie. that have `deck.interval.bottom > depth`

deckList.decksIntersecting(*interval*)	Deck List	Returns a deck list that corresponds to the decks in the hole that intersect the supplied interval eg. if a deck list contained a deck from 0 - 5m filled with drill cuttings, and the subsequent call was hole.decksIntersecting (1, 3) - the 5m drill cuttings deck would be returned. If the bottom of a deck intersects with the top of an interval, this deck will not be included. Conversely, if the top of a deck intersects with the bottom of an interval, this deck will not be included.
deckList.deckAtDepth(*depth*)	Deck	Returns the deck at the given depth. If the depth is a the intersection between two decks, the lower of the two decks is returned. If the depth is above or below the hole, an empty deck is returned.

Hole Object

A hole object contains information about the current state of the hole.

The hole object has the following properties/members:

Member	Type	Description
hole.id	string	A string that contains the ID of this hole. This can be used to perform special handling for holes with certain hole ids. eg. To test if a hole ID starts with 'A' and some digits. `if (/^A\d+/.test(hole.id)) ...`
hole.properties	object	An object that contains hole property and drilling property values for this hole. • **Float** fields have the hole's numeric value if defined, otherwise NaN if the value is undefined for this hole. • **FloatRange** fields have an array of 2 numbers. The first number specifies the lower bound of the range The second number specifies the upper bound of the range. Has the value 'null' if the float range is undefined for this hole. • **String** fields have the hole's string value if defined, otherwise the empty string. • **Boolean** fields have the hole's value if defined (true/false), otherwise 'null' if no value is set. • **Select** fields have a string containing the selected option's value, otherwise 'null' if no value is set. • **Multi-select** fields have an array of strings containing the select option's values. Equal to the empty array if no options are selected.

hole.length	float	The current length of the hole in metres. This will use all available information on the hole to determine the most up-to-date hole length. If no dip information is available then this will be the as-drilled length if that is defined, otherwise the target drill depth, otherwise the target charge depth. A charge rule is required to specify decks that completely fill fill this length of the hole.
hole.targetLength	float	The planned target charge depth of the hole in metres. This is the length of the hole from the actual collar location and angle to achieve the same toe RL as the design charge toe. Some blast designs may wish to add a 'backfill' deck at the bottom of the hole if the `hole.length` is significantly greater than the `hole.targetLength`. This will be the special value NaN if it is unknown.
hole.diameter	float	The actual diameter of the hole in metres. This will be the special value NaN if it is unknown.
hole.collar	object	A structure containing `.x`, `.y` and `.z` members corresponding to the hole's actual collar coordinates.
hole.toe	object	A structure containing `.x`, `.y` and `.z` members corresponding to the hole's actual toe coordinates.
hole.bearing	float	The actual bearing of the hole in radians from north towards east.
hole.angle	float	The actual angle of the hole in radians from vertical towards `hole.bearing`.
hole.burden	float	The distance in metres between this hole and the next row of holes in the direction of the blast.
hole.spacing	float	The distance in metres between this hole and its neighbouring holes in the same row.
hole.targetStandoff	float	The distance in metres between the targetLength (target charge depth) and the reference surface (which may or may not be related to an ore surface). A positive targetStandoff value indicates the reference surface is below target charge depth (targetLength), where as a negative value indicates the reference surface is above target charge depth. This is a downhole distance.
hole.waterLength	float	The length of standing water in the bottom of the hole. This will be the special value NaN if the water length is unknown.
hole.wetLength	float	The length of wet-sides in the bottom of the hole. This will be the special value NaN if the wet-sides length is unknown.

hole .temperature	float	The last measured temperature of the hole in Kelvin. This will be the special value NaN if the temperature has never been measured. New in 2.1.
hole .previousTemperature	float	The previous measured temperature of the hole in Kelvin. This will be the special value NaN if the temperature has only been measured once. New in 2.1.
hole.temperatureAge	float	The number of days since the temperature was measured relative to when the rule was executed, typically greater than 0. This will be the special value NaN if the temperature has never been measured. New in 2.1.
hole.previousTemperatureAge	float	The number of days since the previous temperature was measured relative to when the rule was executed, typically greater than 0. This will be the special value NaN if the temperature has only been measured once. New in 2.1.
hole.blastVolume(*length*)	float	The approximate volume of rock surrounding this hole from collar to specified length. This can be used in rules that calculate volume of explosives based on a target powder-factor (PF) and rock volume. This may be a calculated simply using *hole*.burden * *hole*.spacing * cos(*hole*.angle) * length or may use some more sophisticated algorithms that perform volumetric analysis of nearby holes to determine this value more accurately. Returns the blast volume in cubic metres if there is enough information to calculate it, otherwise returns NaN.
hole.all()	Interval	Returns an interval that corresponds to the entire hole. ie. [0, *hole*.length]
hole.upper(*length*)	Interval	Returns an interval that corresponds to the upper 'length' of the hole. ie. [0, min(*hole*.length, *length*)]
hole.lower(*length*)	Interval	Returns an interval that corresponds to the lower 'length' of the hole. ie. [max(*hole*.length - *length*, 0), *hole*.length]
hole.above(*depth*)	Interval	Returns an interval that corresponds to the part of the hole above the specified depth. ie [0, min(*depth*, *hole*.length)]
hole.below(*depth*)	Interval	Returns an interval that corresponds to the part of the hole below the specified depth. ie [max(*depth*, 0), *hole*.length]
hole.between(*a*, *b*)	Interval	Returns an interval that corresponds to the part of the hole between the two specified depths, a and b. ie [max(min(*a*, *b*), 0), min(max(*a*, *b*), *hole*.length)]

hole.labelled(*label*)	Interval	Returns an interval that corresponds to the part of the hole labelled with the specified interval label. New in BlastLogic 1.1

Parameters Object

The parameters object gives the script access to the custom parameter values for the charge rule.

The parameters object is a dictionary/object whose properties are named according to the `'id'` of the corresponding field in the `'fields'` array

The following field types are supported by blast-design objects.

Field Type	Value Type	Notes
Float	`float`	Float fields are exposed as float values. An undefined value (ie `'null'` in JSON) is represented as the `NaN` value in JavaScript.
Boolean	`boolean`	Boolean fields are exposed as boolean values. ie. `true`, `false` or `null`.
String	`string`	String fields are exposed as string values. eg. "Foo" An undefined value (ie. `'null'` in JSON) is represented as the empty string, "", value in JavaScript.
Select	`string`	Select fields are exposed as string values - the value corresponding to the option's 'value' rather than its 'label'. An undefined value (ie. `'null'` in JSON) is represented as the null value in JavaScript.
Multi Select	`string[]`	Multi select fields are exposed as an array of string values - the values correspond to the option's 'values' rather than its 'labels'. An undefined value (ie. `'null'` in JSON) is represented as the null value in JavaScript.
Blast Product	Blast Product	A blast-product field is exposed as an object that contains information about the blast product. The blast-product is retrieved from the server and used to populate a Blast Product object.
Blast Product Family	Blast Product Family	A blast-product family field is exposed as an object that contains information about the group of blast products that comprise that blast product family. A script can use the blast product family object to choose a particular blast product in the family.
Load Table	Load Table	A load-table field is exposed as an object that can be used to apply the rules defined in the table. See the Load Table Object section below for details.
Interval Label	Interval Label	An interval-label field is exposed as an object that can be used with `hole.labelled()` to determine the parts of the hole labelled with the specified label.

Hole Charge Plan Object

The hole charge plan object is used to construct the load plan for the particular hole the charge rule is running against.

The hole charge plan object initially contains a single deck that spans the entire hole (interval of `[0, hole.length]`) and has an undefined product assigned.
It contains a number of helper methods that can be used to assign intervals of the hole with other blast products.
A charge rule must assign

This class has the same interface as provided by the **Hole Object** and further adds the following methods to the interface.

Member	Type	Description
hole.localisedWaterDepth(*radius*)	float	This function calculates the depth down the hole of the upper-most water level of all holes within specified `radius`. Returns `NaN` if none of the surrounding holes have any recorded water in the hole.
hole.localisedWetDepth(*radius*)	float	This function calculates the depth down the hole of the upper-most wet sides level of all holes within specified `radius`. Returns `NaN` if none of the surrounding holes have any recorded water in the hole.
hole.nearbyHoles(*radius*)	array of Hole	Returns an array of Hole objects for all holes with collars within the specified radius of this hole. Does not include this hole in the list.
hole.unallocated()	Interval	The interval corresponding to the unallocated parts of the hole. ie the parts of the hole that have not yet had a product allocated to them.
hole.totalExplosiveMass()	float	Returns the total mass of explosives (in kg) allocated to the hole so far.
hole.allExplosives()	Interval	The interval corresponding to the parts of the hole that have been allocated explosive decks. Adjacent/touching explosive decks are merged in to one interval, so this effectively returns an interval for each 'explosive column'.
hole.allExplosivesBelow(*depth*, *tolerance*=0)	Interval	Returns an interval for each of the explosive decks in the plan below the specified depth. A tolerance can optionally be specified that will allow the interval list to extend up to 'tolerance' above 'depth' if the top of the deck overlapping the specified depth extents by less than tolerance above depth in order to avoid splitting that deck into too small a deck if the returned interval range were to be replaced with a different product. This can be used to select all of the explosive products below a particular water level and replace them with a water-resistant product.

hole.productsBelow(*products*, *depth*, *tolerance*=0)	Interval	Returns an interval list for each of the decks in the plan containing a product in the 'products' list that extend below the specified depth. A tolerance can optionally be specified that will allow the interval list to extend up to 'tolerance' above 'depth' if the top of the deck overlapping the specified depth extents by less than tolerance above depth in order to avoid splitting that deck into too small a deck if the returned interval range were to be replaced with a different product. This can be used to select all of the explosive products below a particular water level and replace them with a water-resistant product.
hole.allDecks()	Deck List	Returns a deck list that corresponds to all the decks in the hole. New in BlastLogic 2.1 Update 2
hole.decksAbove(*depth*)	Deck List	Returns a deck list that corresponds to the decks in the hole that sit above, as well as intersect with the supplied depth. If the depth is at the intersection between two decks, both decks will be included. New in BlastLogic 2.1 Update 2
hole.decksBelow(*depth*)	Deck List	Returns a deck list that corresponds to the decks in the hole that sit below, as well as intersect with the supplied depth. If the depth is at the intersection between two decks, only the lower of the two decks will be included. New in BlastLogic 2.1 Update 2
hole.decksIntersecting(*interval*)	Deck List	Returns a deck list that corresponds to the decks in the hole that intersect the supplied interval ie. if a hole contained a deck from 0 - 5m filled with drill cuttings, and the subsequent call was hole.decksIntersecting(1, 3) - the 5m drill cuttings deck would be returned. If the bottom of a deck intersects with the top of an interval, this deck will not be included. Conversely, if the top of a deck intersects with the bottom of an interval, this deck will not be included. New in BlastLogic 2.1 Update 2
hole.deckAtDepth(*depth*)	Deck	Returns the deck at the given depth. If the depth is at the intersection between two decks, the lower of the two decks is returned. If the depth is above or below the hole, an empty deck is returned. New in BlastLogic 2.1 Update 2
hole.fill(*interval, product*)	Interval	Fill the specified interval list with the specified product. Returns the interval that is filled.

hole.fillPackaged(*interval* , *product*, *quantity*)	Interval	Fills the given interval with the specified amount of packaged products. If the interval length is not a multiple of the product length, then it is rounded to the closest multiple (Note: the fixed length for a fixed volume product is calculated from the diameter of the hole). Quantity is simply the count of fixed products to use for the interval. Returns the interval that is
hole.fillToMass(*interval*, *product*, *maxMass*)	Interval	Fills the given interval with the given product, up to the given mass. This function calls onto hole.fillPackaged or hole.fill as appropriate. If the interval is too small for the given mass, then the interval will be filled as much as possible (for fixed length decks, this means that more products are specified than the interval length). If the mass is required is smaller than the interval, then the interval filled is the bottom of the section. Returns the interval that is
hole.fillWithTable(*interval*, *table*, *targetExplosiveMass*, *wetDepth*)	void	Set the charge plan for the part of the hole corresponding to Interval List using the rules in the specified Load Table. The table is evaluated using the length of the intervals and the specified wet depth to calculate decks to use for those intervals as if they were connected. eg. `var waterDepth = hole.localisedWaterDepth (parameters.waterRadius); var explosiveMass = hole.blastVolume (hole.targetLength) * parameters.powderFactor; hole.fillWithTable (hole.above (hole.targetLength), parameters.loadTable, explosiveMass,`

hole.addPrimer(*depth*, *products*)	void	Add a primer at the specified depth. The products parameter must be an array of objects of type BlastProduct or BlastProductFamily. A primer comprises one or more blast products and must include either: • one down-hole delay product • two down-hole delay products (for dual-detonator boosters) • one detonating-cord product The primer can additionally contain an optional booster blast product. If the object is a BlastProductFamily then the `selectProduct(depth)` method is called on the object to select the appropriate lead
hole.addPrimers(*interval*, *products*, *maxSpacing*, *bottomStandoff*, *topStandoff*)	void	Add a number of primers within the specified `interval` to the hole with at most maxSpacing distance between the primers. Adds primers at depths within each sub-interval such that the bottom-most primer is `bottomStandoff` above the bottom of the sub-interval and the top-most primer is at least `topStandoff` below the top of the sub-interval. If a sub-interval is shorter than `bottomStandoff + topStandoff` then a single primer is placed in the middle of the sub-interval. For sub-intervals longer than `bottomStandoff + maxSpacing` multiple primers will be added within the sub-interval. The number of primers will be `Math.round((subInterval.length - bottomStandoff) / maxSpacing)`. These primers will be placed evenly spaced between `bottomStandoff` above the bottom and `topStandoff` below the top of the sub-interval.

Load Table Object

A load table object allows the user to specify basic load designs in tabular form.

The load table functionality is described in BAMS Design - Charge Rule Load Tables.

> **TODO**
>
> Document details of the LoadTable object.
>
> Not really required as load table objects are typically just passed into calls to `hole.fillWithTable()`.

Blast Product Object

> TODO: This needs to be extended to support different product types (bulk explosive, packaged explosive, air, drill cuttings, stemming, gas bag)

The blast product object contains information about a particular blast product defined in the BlastLogic database.

A blast product has the following members:

Member	Type	
product.name	string	The name of this blast product.
product.isExplosive	boolean	Whether this blast product is an explosive product.
product.isDeckProduct	boolean	Whether this blast product is a deck product.
product.isPrimerProduct	boolean	Whether this blast product is a primer product.
product.isPackagedProduct	boolean	Whether this blast product is a packaged product.
product.isWaterResistant	boolean	Whether this blast product is resistant to water in the hole. Provided the "Static water" or "Dynamic water" values in the **Sleep times** table are non-zero, the product will be considered usable in water conditions. Non-explosive products are all considered water resistant, except for gas bags. New in BlastLogic 2.1.
product.isDampResistant	boolean	Whether this blast product is resistant to wet sides in the hole. Provided the "Dewatered" value in the **Sleep times** table is non-zero, the product will be considered usable in damp conditions. Non-explosive products are all considered damp resistant. New in BlastLogic 2.1.
product.density	float	The open cup density in kg/m^3 of the blast product. If the product is not measured by mass then this will be NaN.
product.length	float	For blast products that have a fixed length, this will be the length of the blast product. If the product does not have a fixed length then this will be NaN.
product.mass	float	For blast products that have a fixed length or fixed volume, this will be the mass of the blast product. If this product is not a packaged product then this will be NaN.
product.volume	float	For blast products that have a fixed volume, this will be the volume of the blast product. If the product does not have a fixed volume then this will be NaN.
product.delay	float	For down hole delay products, this will be the delay of the blast product. Otherwise, this will be NaN.

product.calculateLength(*mass*, *diameter*)	float	A helper function that calculates the length (in metres) of a hole with specified diameter (in metres) that will be filled by specified mass (in kg) of product. If the `product.density` is NaN then this will return `NaN`. When used with packaged products, this returns the minimum length required to fill up to the given mass. This means that the mass used to calculate the length will be the smallest mass greater the given mass such that a whole number of packaged products are used. Packages are assumed to only fit 1 per length (ie, double packing product "sausages" tubes into a hole is not taken into account).
product.calculateLengthForDensity(*mass*, *diameter*, *density*)	float	Similar to *product*.calculateLengthForDensity(*mass*, *diameter*, *density*), however this function takes in a density (kg/m^3). This provides a more accurate calculation when using gassed products at a specific hole depth. *Deprecated in BlastLogic 2.1 Update 3*.
product.calculateMass(*length*, *diameter*)	float	A helper function that calculates the mass (in kg) of product required to fill a hole with specified diameter (in metres) to the specified length (in metres). If the `product.density` is NaN then this will return `NaN`. When used with packaged products, this returns the maximum mass that can be filled into the given length. Packages are assumed to only fit 1 per length (ie, double packing product "sausages" into a hole is not taken into account).
product.calculateMassForDensity(*length*, *diameter*, *density*)	float	Similar to *product*.calculateMassForDensity(*length*, *diameter*, *density*), however this function takes in a density (kg/m^3). This provides a more accurate calculation when using gassed products at a specific hole depth. *Deprecated in BlastLogic 2.1 Update 3*.
product.averageInHoleDensity(*depth*, *openCupDensity*)	float	Calculates the average in hole density (kg/m^3) for the given product, at the specified depth. This is only for use with gassed products, where the product density table has been added to the Blast Product Catalogue. If no density table has been provided for this product, the function will throw an error. *Deprecated in BlastLogic 2.1 Update 3*.
product.toeDensity(*depth*, *openCupDensity*)	float	Calculates the toe density (kg/m^3) for the given product, at the specified depth. This is only for use with gassed products, where the product density table has been added to the Blast Product Catalogue. If no density table has been provided for this product, the function will throw an error. *Deprecated in BlastLogic 2.1 Update 3*.

product.densityAtDepth(*depth*)	float	Calculates the density of a gassed product at the specified depth. This provides a more accurate calculation when using gassed products at a specific hole depth. New in BlastLogic 2.1 Update 3.
product.maximumViableDepth()	float	Maximum depth supported by this products density table. This is only applicable to bulk explosive products. Product columns that exceed this length have the potential to be desensitised. Products with no density table will return a value of positive infinity. New in BlastLogic 2.1 Update 3.

Built-in Blast Product Object

A small number of Blast Products have been built-in to the script. They behave like any other Blast Product, using the properties and functions defined in the above table - Blast Product Object.

The addition of the products object is optional . It has the following members:

Member	Type	Description
products.air	Blast Product	The built-in "Air" product. New in BlastLogic 2.1
products.drillCuttings	Blast Product	The built-in "Drill Cuttings" product. New in BlastLogic 2.1

Blast Product Family Object

Member	Type	Description
family.name	string	The name of this blast product family.
family.products	Array	Array of the blast products that belong in this family.
family.selectProduct(*depth*, *delay*)	BlastProduct	Returns a blast product that has a length of "depth + 2" and with a delay as specified. If no delay is given (or NaN is given as the delay) then the first product that has the required length is returned.

Filtered Blast Product Family Object

Member	Type	Description
family.family	BlastProductFamily	The blast product family this object filters on.
family.delay	float	The delay that is used as a filter for this family. If this is NaN, all blast products are displayed, otherwise only the blast products with the specified delay are shown.

family.selectProduct(*depth*)	BlastProduct	Returns a blast product from the blast product family that has a length of "depth + 2". The delay used to select this product is family.delay.