AmortizationCalculation

Constants Summary

Number	NUMBER_UNLIMITED The numeric constant used to identify an unlimited number of repeated events.
Number	PERIOD_ANNUALY The numeric constant used to identify an annual period.
Number	PERIOD_BI_ANNUALLY The numeric constant used to identify a bi-annual period (twice every year).
Number	PERIOD_BI_MONTHLY The numeric constant used to identify a bi-monthly period (twice every month).
Number	PERIOD_DAILY The numeric constant used to identify a daily period.
Number	PERIOD_FOUR_MONTHLY The numeric constant used to identify a four-monthly period (once every four months).
Number	PERIOD_FOUR_WEEKLY The numeric constant used to identify a four-weekly period (once every four weeks).
Number	PERIOD_MONTHLY The numeric constant used to identify a monthly period.
Number	PERIOD_NONE The numeric constant used to identify that there is no period.
Number	PERIOD_QUARTERLY The numeric constant used to identify a quarterly period (once every three months).
Number	PERIOD_TWO_MONTHLY The numeric constant used to identify a two-monthly period (once every two months).
Number	PERIOD_TWO_WEEKLY The numeric constant used to identify a two-weekly period (once every two weeks).
Number	PERIOD_WEEKLY The numeric constant used to identify a weekly period.
Number	STARTDAY_NORMAL The numeric constant used to identify that the same start day should be used as the day of the month of the starting date of the event.

Method Summary

Boolean	addCompoundPeriodChange(newPeriod, date) Adds a compound period change.
Boolean	addLoan(amount, date) Adds a loan.
Boolean	addLoan(amount, firstDate, lastDate, period) Adds a loan.
Boolean	addLoan(amount, firstDate, lastDate, period, number) Adds a loan.
Boolean	addLoan(amount, firstDate, lastDate, period, number, startday) Adds a loan.
Boolean	addPayment(amount, date) Adds a payment.
Boolean	addPayment(amount, firstDate, lastDate, period) Adds a payment.
Boolean	addPayment(amount, firstDate, lastDate, period, number) Adds a payment.
Boolean	addPayment(amount, firstDate, lastDate, period, number, startday) Adds a payment.
Boolean	addRateChange(newRate, date) Sets a new interest rate.
Boolean	calculateAmortizationSchedule() Calculates the amortization schedule.
JSDataSet	getAmortizationSchedule() Gets the amortization schedule as a JSDataSet.
Number	getError() Returns the error that remains when solving for the unknown.
JSDataSet	getEvents() Returns all the amortization events - such as rate changes, loan events, payment events, compounding period changes.
Number	getRestBalance() Gets the rest balance after the amortization schedule.
Number	getUnknown() Returns the solveForUnknown value.

Boolean	isValidPeriod(period) Returns true if the period is valid, or false if the period is not valid.
Number	roundMoney(amount) Rounds a number up to the nearest cents.
Boolean	solveForUnknown() Returns true if successful or false if the call failed.
void	sortEvents() Sorts the amortization events ascending by date.

Constants Details

NUMBER_UNLIMITED

The numeric constant used to identify an unlimited number of repeated events.

Returns

Number

Sample

```
var c = plugins.amortization.newCalculation();
c.addPayment(500, new Date(2005, 1, 28), null,
                        plugins.amortization.AmortizationCalculation.PERIOD_DAILY, 5,
                        plugins.amortization.AmortizationCalculation.STARTDAY_NORMAL);

var c2 = plugins.amortization.newCalculation();
c2.addPayment(300, new Date(2006, 11, 24), new Date(2006, 12, 24),
                        plugins.amortization.AmortizationCalculation.PERIOD_BI_MONTHLY,
                        plugins.amortization.AmortizationCalculation.NUMBER_UNLIMITED, 30);
```

PERIOD_ANNUALY

The numeric constant used to identify an annual period.

Returns

Number

Sample

```
var c = plugins.amortization.newCalculation();
c.addPayment(500, new Date(2005, 1, 28), null,
                        plugins.amortization.AmortizationCalculation.PERIOD_DAILY, 5,
                        plugins.amortization.AmortizationCalculation.STARTDAY_NORMAL);

var c2 = plugins.amortization.newCalculation();
c2.addPayment(300, new Date(2006, 11, 24), new Date(2006, 12, 24),
                        plugins.amortization.AmortizationCalculation.PERIOD_BI_MONTHLY,
                        plugins.amortization.AmortizationCalculation.NUMBER_UNLIMITED, 30);
```

PERIOD_BI_ANNUALLY

The numeric constant used to identify a bi-annual period (twice every year).

Returns

Number

Sample

```
var c = plugins.amortization.newCalculation();
c.addPayment(500, new Date(2005, 1, 28), null,
                        plugins.amortization.AmortizationCalculation.PERIOD_DAILY, 5,
                        plugins.amortization.AmortizationCalculation.STARTDAY_NORMAL);

var c2 = plugins.amortization.newCalculation();
c2.addPayment(300, new Date(2006, 11, 24), new Date(2006, 12, 24),
                        plugins.amortization.AmortizationCalculation.PERIOD_BI_MONTHLY,
                        plugins.amortization.AmortizationCalculation.NUMBER_UNLIMITED, 30);
```

PERIOD_BI_MONTHLY

The numeric constant used to identify a bi-monthly period (twice every month).
TODO: this period is not supported yet.

Returns

Number

Sample

```
var c = plugins.amortization.newCalculation();
c.addPayment(500, new Date(2005, 1, 28), null,
                        plugins.amortization.AmortizationCalculation.PERIOD_DAILY, 5,
                        plugins.amortization.AmortizationCalculation.STARTDAY_NORMAL);

var c2 = plugins.amortization.newCalculation();
c2.addPayment(300, new Date(2006, 11, 24), new Date(2006, 12, 24),
                        plugins.amortization.AmortizationCalculation.PERIOD_BI_MONTHLY,
                        plugins.amortization.AmortizationCalculation.NUMBER_UNLIMITED, 30);
```

PERIOD_DAILY

The numeric constant used to identify a daily period.

Returns

Number

Sample

```
var c = plugins.amortization.newCalculation();
c.addPayment(500, new Date(2005, 1, 28), null,
                        plugins.amortization.AmortizationCalculation.PERIOD_DAILY, 5,
                        plugins.amortization.AmortizationCalculation.STARTDAY_NORMAL);

var c2 = plugins.amortization.newCalculation();
c2.addPayment(300, new Date(2006, 11, 24), new Date(2006, 12, 24),
                        plugins.amortization.AmortizationCalculation.PERIOD_BI_MONTHLY,
                        plugins.amortization.AmortizationCalculation.NUMBER_UNLIMITED, 30);
```

PERIOD_FOUR_MONTHLY

The numeric constant used to identify a four-monthly period (once every four months).

Returns

Number

Sample

```
var c = plugins.amortization.newCalculation();
c.addPayment(500, new Date(2005, 1, 28), null,
                        plugins.amortization.AmortizationCalculation.PERIOD_DAILY, 5,
                        plugins.amortization.AmortizationCalculation.STARTDAY_NORMAL);

var c2 = plugins.amortization.newCalculation();
c2.addPayment(300, new Date(2006, 11, 24), new Date(2006, 12, 24),
                        plugins.amortization.AmortizationCalculation.PERIOD_BI_MONTHLY,
                        plugins.amortization.AmortizationCalculation.NUMBER_UNLIMITED, 30);
```

PERIOD_FOUR_WEEKLY

The numeric constant used to identify a four-weekly period (once every four weeks).

Returns

Number

Sample

```
var c = plugins.amortization.newCalculation();
c.addPayment(500, new Date(2005, 1, 28), null,
                        plugins.amortization.AmortizationCalculation.PERIOD_DAILY, 5,
                        plugins.amortization.AmortizationCalculation.STARTDAY_NORMAL);

var c2 = plugins.amortization.newCalculation();
c2.addPayment(300, new Date(2006, 11, 24), new Date(2006, 12, 24),
                        plugins.amortization.AmortizationCalculation.PERIOD_BI_MONTHLY,
                        plugins.amortization.AmortizationCalculation.NUMBER_UNLIMITED, 30);
```

PERIOD_MONTHLY

The numeric constant used to identify a monthly period.

Returns

Number

Sample

```
var c = plugins.amortization.newCalculation();
c.addPayment(500, new Date(2005, 1, 28), null,
                        plugins.amortization.AmortizationCalculation.PERIOD_DAILY, 5,
                        plugins.amortization.AmortizationCalculation.STARTDAY_NORMAL);

var c2 = plugins.amortization.newCalculation();
c2.addPayment(300, new Date(2006, 11, 24), new Date(2006, 12, 24),
                        plugins.amortization.AmortizationCalculation.PERIOD_BI_MONTHLY,
                        plugins.amortization.AmortizationCalculation.NUMBER_UNLIMITED, 30);
```

PERIOD_NONE

The numeric constant used to identify that there is no period.

Returns

Number

Sample

```
var c = plugins.amortization.newCalculation();
c.addPayment(500, new Date(2005, 1, 28), null,
                        plugins.amortization.AmortizationCalculation.PERIOD_DAILY, 5,
                        plugins.amortization.AmortizationCalculation.STARTDAY_NORMAL);

var c2 = plugins.amortization.newCalculation();
c2.addPayment(300, new Date(2006, 11, 24), new Date(2006, 12, 24),
                        plugins.amortization.AmortizationCalculation.PERIOD_BI_MONTHLY,
                        plugins.amortization.AmortizationCalculation.NUMBER_UNLIMITED, 30);
```

PERIOD_QUARTERLY

The numeric constant used to identify a quarterly period (once every three months).

Returns

Number

Sample

```
var c = plugins.amortization.newCalculation();
c.addPayment(500, new Date(2005, 1, 28), null,
                        plugins.amortization.AmortizationCalculation.PERIOD_DAILY, 5,
                        plugins.amortization.AmortizationCalculation.STARTDAY_NORMAL);

var c2 = plugins.amortization.newCalculation();
c2.addPayment(300, new Date(2006, 11, 24), new Date(2006, 12, 24),
                        plugins.amortization.AmortizationCalculation.PERIOD_BI_MONTHLY,
                        plugins.amortization.AmortizationCalculation.NUMBER_UNLIMITED, 30);
```

PERIOD_TWO_MONTHLY

The numeric constant used to identify a two-monthly period (once every two months).

Returns

Number

Sample

```
var c = plugins.amortization.newCalculation();
c.addPayment(500, new Date(2005, 1, 28), null,
                        plugins.amortization.AmortizationCalculation.PERIOD_DAILY, 5,
                        plugins.amortization.AmortizationCalculation.STARTDAY_NORMAL);

var c2 = plugins.amortization.newCalculation();
c2.addPayment(300, new Date(2006, 11, 24), new Date(2006, 12, 24),
                        plugins.amortization.AmortizationCalculation.PERIOD_BI_MONTHLY,
                        plugins.amortization.AmortizationCalculation.NUMBER_UNLIMITED, 30);
```

PERIOD_TWO_WEEKLY

The numeric constant used to identify a two-weekly period (once every two weeks).

Returns

Number

Sample

```
var c = plugins.amortization.newCalculation();
c.addPayment(500, new Date(2005, 1, 28), null,
                        plugins.amortization.AmortizationCalculation.PERIOD_DAILY, 5,
                        plugins.amortization.AmortizationCalculation.STARTDAY_NORMAL);

var c2 = plugins.amortization.newCalculation();
c2.addPayment(300, new Date(2006, 11, 24), new Date(2006, 12, 24),
                        plugins.amortization.AmortizationCalculation.PERIOD_BI_MONTHLY,
                        plugins.amortization.AmortizationCalculation.NUMBER_UNLIMITED, 30);
```

PERIOD_WEEKLY

The numeric constant used to identify a weekly period.

Returns

Number

Sample

```
var c = plugins.amortization.newCalculation();
c.addPayment(500, new Date(2005, 1, 28), null,
                        plugins.amortization.AmortizationCalculation.PERIOD_DAILY, 5,
                        plugins.amortization.AmortizationCalculation.STARTDAY_NORMAL);

var c2 = plugins.amortization.newCalculation();
c2.addPayment(300, new Date(2006, 11, 24), new Date(2006, 12, 24),
                        plugins.amortization.AmortizationCalculation.PERIOD_BI_MONTHLY,
                        plugins.amortization.AmortizationCalculation.NUMBER_UNLIMITED, 30);
```

STARTDAY_NORMAL

The numeric constant used to identify that the same start day should be used as the day of the month of the starting date of the event.

Returns

Number

Sample

```
var c = plugins.amortization.newCalculation();
c.addPayment(500, new Date(2005, 1, 28), null,
                        plugins.amortization.AmortizationCalculation.PERIOD_DAILY, 5,
                        plugins.amortization.AmortizationCalculation.STARTDAY_NORMAL);

var c2 = plugins.amortization.newCalculation();
c2.addPayment(300, new Date(2006, 11, 24), new Date(2006, 12, 24),
                        plugins.amortization.AmortizationCalculation.PERIOD_BI_MONTHLY,
                        plugins.amortization.AmortizationCalculation.NUMBER_UNLIMITED, 30);
```

Method Details

addCompoundPeriodChange

Boolean **addCompoundPeriodChange** (newPeriod, date)

Adds a compound period change.

Parameters

{Number} newPeriod
{Date} date

Returns

Boolean

Sample

```
var c = plugins.amortization.newCalculation();
c.addRateChange(r, new Date(2005, 0, 1));
c.addCompoundPeriodChange(12, new Date(2005, 0, 1));
c.addLoan(2000, new Date(2005, 0, 1));
c.addPayment(500, new Date(2005, 1, 28), null, 12, 5, 31);
```

addLoan

Boolean **addLoan** (amount, date)

Adds a loan.

Parameters

{Number} amount
{Date} date

Returns

Boolean

Sample

```
var c = plugins.amortization.newCalculation();
c.addRateChange(r, new Date(2005, 0, 1));
c.addCompoundPeriodChange(12, new Date(2005, 0, 1));
c.addLoan(2000, new Date(2005, 0, 1));
c.addPayment(500, new Date(2005, 1, 28), null, 12, 5, 31);
```

addLoan

Boolean **addLoan** (amount, firstDate, lastDate, period)

Adds a loan.

Parameters

{Number} amount
{Date} firstDate
{Date} lastDate
{Number} period

Returns

Boolean

Sample

```
var c = plugins.amortization.newCalculation();
c.addRateChange(r, new Date(2005, 0, 1));
c.addCompoundPeriodChange(12, new Date(2005, 0, 1));
c.addLoan(2000, new Date(2005, 0, 1));
c.addPayment(500, new Date(2005, 1, 28), null, 12, 5, 31);
```

addLoan

Boolean **addLoan** (amount, firstDate, lastDate, period, number)

Adds a loan.

Parameters

{Number} amount
{Date} firstDate
{Date} lastDate
{Number} period
{Number} number

Returns

Boolean

Sample

```
var c = plugins.amortization.newCalculation();
c.addRateChange(r, new Date(2005, 0, 1));
c.addCompoundPeriodChange(12, new Date(2005, 0, 1));
c.addLoan(2000, new Date(2005, 0, 1));
c.addPayment(500, new Date(2005, 1, 28), null, 12, 5, 31);
```

addLoan

Boolean **addLoan** (amount, firstDate, lastDate, period, number, startday)

Adds a loan.

Parameters

{Number} amount
{Date} firstDate
{Date} lastDate
{Number} period
{Number} number
{Number} startday

Returns

Boolean

Sample

```
var c = plugins.amortization.newCalculation();
c.addRateChange(r, new Date(2005, 0, 1));
c.addCompoundPeriodChange(12, new Date(2005, 0, 1));
c.addLoan(2000, new Date(2005, 0, 1));
c.addPayment(500, new Date(2005, 1, 28), null, 12, 5, 31);
```

addPayment

Boolean **addPayment** (amount, date)

Adds a payment.

Parameters

{Number} amount
{Date} date

Returns

Boolean

Sample

```
var c = plugins.amortization.newCalculation();
c.addRateChange(r, new Date(2005, 0, 1));
c.addCompoundPeriodChange(12, new Date(2005, 0, 1));
c.addLoan(2000, new Date(2005, 0, 1));
c.addPayment(500, new Date(2005, 1, 28), null, 12, 5, 31);
```

addPayment

Boolean **addPayment** (amount, firstDate, lastDate, period)

Adds a payment.

Parameters

{Number} amount
{Date} firstDate
{Date} lastDate
{Number} period

Returns

Boolean

Sample

```
var c = plugins.amortization.newCalculation();
c.addRateChange(r, new Date(2005, 0, 1));
c.addCompoundPeriodChange(12, new Date(2005, 0, 1));
c.addLoan(2000, new Date(2005, 0, 1));
c.addPayment(500, new Date(2005, 1, 28), null, 12, 5, 31);
```

addPayment

Boolean **addPayment** (amount, firstDate, lastDate, period, number)

Adds a payment.

Parameters

{Number} amount
{Date} firstDate
{Date} lastDate
{Number} period
{Number} number

Returns

Boolean

Sample

```
var c = plugins.amortization.newCalculation();
c.addRateChange(r, new Date(2005, 0, 1));
c.addCompoundPeriodChange(12, new Date(2005, 0, 1));
c.addLoan(2000, new Date(2005, 0, 1));
c.addPayment(500, new Date(2005, 1, 28), null, 12, 5, 31);
```

addPayment

Boolean **addPayment** (amount, firstDate, lastDate, period, number, startday)

Adds a payment.

Parameters

{Number} amount
{Date} firstDate
{Date} lastDate
{Number} period
{Number} number
{Number} startday

Returns

Boolean

Sample

```
var c = plugins.amortization.newCalculation();
c.addRateChange(r, new Date(2005, 0, 1));
c.addCompoundPeriodChange(12, new Date(2005, 0, 1));
c.addLoan(2000, new Date(2005, 0, 1));
c.addPayment(500, new Date(2005, 1, 28), null, 12, 5, 31);
```

addRateChange

Boolean **addRateChange** (newRate, date)

Sets a new interest rate.

Parameters

{Number} newRate
{Date} date

Returns

Boolean

Sample

```
var c = plugins.amortization.newCalculation();
c.addRateChange(r, new Date(2005, 0, 1));
c.addCompoundPeriodChange(12, new Date(2005, 0, 1));
c.addLoan(2000, new Date(2005, 0, 1));
c.addPayment(500, new Date(2005, 1, 28), null, 12, 5, 31);
```

calculateAmortizationSchedule

Boolean **calculateAmortizationSchedule** ()

Calculates the amortization schedule.

Returns

Boolean

Sample

```
plugins.amortization.calculateAmortizationSchedule();
```

getAmortizationSchedule

JSDataSet **getAmortizationSchedule** ()

Gets the amortization schedule as a JSDataSet.

Returns

JSDataSet

Sample

```
plugins.amortization.getAmortizationSchedule();
```

getError

Number **getError** ()

Returns the error that remains when solving for the unknown.
Please note that the error should be less or equal to 1E-8 - otherwise, the solveForUnknown value is incorrect.

Returns

Number

Sample

```
var c = plugins.amortization.newCalculation();
// sets the rate to -1 for unknown.
c.addRateChange(-1, new Date(2005, 0, 1));
c.addCompoundPeriodChange(12, new Date(2005, 0, 1));
c.addLoan(2000, new Date(2005, 0, 1));
var lastDate = null;
var period = 12;
var number_count = 5;
var startday = 31;
c.addPayment(500, new Date(2005, 1, 28), lastDate, period,number_count, startday);
// solves for the interest rate.
c.solveForUnknown();
// gets the interest rate and the error in the calculation.
// which should be small (otherwise the calculation did
// not converge for some reason.
var r = c.getUnknown();
var e = c.getError();
```

getEvents

JSDataSet **getEvents** ()

Returns all the amortization events - such as rate changes, loan events, payment events, compounding period changes.

Returns

JSDataSet

Sample

```
plugins.amortization.getEvents();
```

getRestBalance

Number **getRestBalance** ()

Gets the rest balance after the amortization schedule.

Returns

Number

Sample

```
var rb = plugins.amortization.getRestBalance();
```

getUnknown

Number **getUnknown** ()

Returns the solveForUnknown value.

Returns

Number

Sample

```
plugins.amortization.getUnknown();
```

isValidPeriod

Boolean **isValidPeriod** (period)

Returns true if the period is valid, or false if the period is not valid.

Parameters

{Number} period

Returns

Boolean

Sample

```
var v_period = plugins.amortization.isValidPeriod(12);
```

roundMoney

Number **roundMoney** (amount)

Rounds a number up to the nearest cents.

Parameters

{Number} amount

Returns

Number

Sample

```
//rounds the number up to 34.35
var rm = plugins.amortization.roundMoney(34.349384);
```

solveForUnknown

Boolean **solveForUnknown** ()

Returns true if successful or false if the call failed.

Returns

Boolean

Sample

```
plugins.amortization.solveForUnknown();
```

sortEvents

void **sortEvents** ()

Sorts the amortization events ascending by date.

Returns

void

Sample

```
plugins.amortization.sortEvents();
```