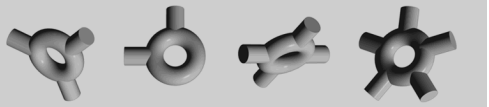
elements



Basic Commodity Constraints Package

TARMS Inc.

September 07, 2000

Typeset in LaTeX.

Contents

List of Figures

List of Tables

Package Description

Commodities can be grouped into collections of commodities tied together by constraints. These constraints can either be strictly enforced or simply desirable. The constraints can also be powerful enough to exactly determine the amounts of one commodity in terms of another or simply represent the enforcement of some sort of relationship.

This package provides a basic infrastructure for talking about commodity constraints. Packages which implement specific domains, such as FX or energy trading implement specific constraints on the commodities and trade types that the package provides.

European Monetary Union The use of common commodity constraint groups, §2.3 allows the capture of the EMU regulations in terms of constraints between the in-currencies and the Euro. The triangulation rules are enforced by the specification of constraints composition. §2.1 This approach is somewhat more abstract than directly encoding the EMU regulations, but allows greater flexibility and the ability to handle other currency groupings, should the need arise.

1 Use Cases

1.1 European Monetary Union

The European Monetary Union[1] consists of a group of currencies, the *in-currencies*, with fixed exchange rates against a single currency, the Euro (EUR). The in-currencies consist of ATS, BEF, DEM, ESP, FIM, FRF, IEP, ITL, NLG, PTE and EUR itself. Any other currency is termed an *out-currency*.

The European Monetary Union imposes a peculiar set of calculations, intended to eliminate rounding anomalies.

To convert amount x_{CUA} in in-currency CUA to an equivalent amount in in-currency CUB, the conversion process *must* (by legislation) follow the *triangulation* rules. Firstly, x_{CUA} is first converted into x_{EUR} using the fixed EUR/CUA exchange rate. Secondly, x_{EUR} is then rounded to accuracy of not less than 3 decimal places to form x'_{EUR}. Thirdly, x'_{EUR} is converted to x_{CUB} via the fixed EUR/CUB exchange rate; this amount is then rounded to whatever normal currency accuracy CUB has. Triangulation is only required when an amount is to be paid or received — an accounting amount — triangulation need not be used to calculate analytics, etc.

There is no legislated rule to be followed when converting between an out-currency and an in-currency. However, general practice is to convert the out currency to a EUR amount first and then convert that amount to the in-currency. Similarly, an in-currency amount is converted to an out currency first by converting it to an equivalent EUR amount and then to the out-currency amount.

The non-EUR in-currencies are expected to be replaced by the EUR in 2002. In January, 2002 Euro notes and coins will be introduced. In July, 2002 the old in-currencies will cease to be legal tender. In the meantime, the in-currencies can be viewed as visible instances of a “true” EUR amount.

2 Interfaces

2.1 CommodityConstraint

A constraint between two commodities. This object defines the relationship between the two commodities.

Some constraints are *composeable*, indicating that the relationship between two commodities can be constructed out of the relationship between two other constraints with a common commodity.

Some constraints are *functional*, indicating that the predicate relationship between the two commodities can be expressed as an invertible functional relationship between the two commodities.

Constraints may be *soft*, indicating that the constraint is a desired characteristic, rather than a mandatory one. Soft constraints, when broken, cause a warning rather than an error.

2.1.1 Relationships

	Class	Description	Notes
↓	CommodityConstraintModel §3.4		
↔	ComposedCommodityConstraint-Model §3.5	second 0..n	
↔	ComposedCommodityConstraint-Model §3.5	first 0..n	
↔	CommodityConstraintGroup-Model §3.1	constraints 0..n	◊

↓:Realized by ↔:Association →:Navigable ◊:Aggregate ♦:Composite

2.1.2 Operations

Commodity firstCommodity() firstCommodity

The first commodity. Return the first commodity that makes up the constraint.

Commodity secondCommodity() secondCommodity

The second commodity. Return the second commodity that makes up the constraint.

Commodity otherCommodity(Commodity commodity) otherCommodity
commodity: Commodity The commodity to match against.
Raises: CommodityConstraintException

The commodity other than the supplied commodity. If commodity is the same as the first commodity, then return the second commodity. If commodity is the same as the second commodity, then return the first commodity. If commodity matches neither the first or second commodity, then raise a CommodityConstraint exception.

Boolean isSoft() isSoft

Is this a soft (non-mandatory) constraint? Return true if this constraint is a soft constraint, false otherwise.

Boolean isComposeable() isComposeable

Is this constraint composeable? Return true if this constraint can be composed with another constraint with a common commodity, false otherwise.

Boolean isFunctional() isFunctional

Can this constraint be turned into a transformation function? Return true if this constraint can be converted into a form $a_2 = f(a_1)$, where a_1 and a_2 are SimpleCashflows of the first and second commodities, respectively and f is a function such that the constraint predicate is always true for a_1 and a_2. The function f must be invertible ($a_1 = f^{-1}(a_2)$).

Functional constraints go one step further than the tests of ordinary constraints and allow the transformation of one amount of a commodity into another.

Boolean predicate(SimpleCashflow amount1, SimpleCashflow amount2)

predicate

amount1: SimpleCashflow The amount of the first commodity.
amount2: SimpleCashflow The amount of the second commodity.

The constraint. Return true if whatever constraint that this constraint represents is satisfied by amount1 of the one commodity and amount2 of the other commodity.

Note that the arguments are dated cashflows. Cashflows, rather than simple amounts, are used to allow for constraints which vary over time.[1]

Set<Commodity> commodities() commodities

The commodities that take part in the constraint. Return the two (or more) commodities that are constrained by this constraint.

SimpleCashflow firstAmountFrom(SimpleCashflow a) firstAmount-From
a: SimpleCashflow The amount of the second commodity.
Raises: CommodityConstraintException

Convert an amount of the second commodity into an amount of the first commodity. If this constraint is functional, then return $g(a)$ where g is the function that converts amounts of the second commodity into amounts of the first commodity. ($g = f^{-1}$) If this constraint is not functional, then raise a CommodityConstraintException.

[1] An example would be a constraint based on a forward FX curve.

SimpleCashflow secondAmountFrom(SimpleCashflow a) secondAmount-From
a: SimpleCashflow The amount of the second commodity.
Raises: CommodityConstraintException

Convert an amount of the first commodity into an amount of the second commodity. If this constraint is functional, then return $f(a)$ where f is the function that converts amounts of the first commodity into amounts of the second commodity. If this constraint is not functional, then raise a CommodityConstraintException.

CommodityConstraint composeWith(CommodityConstraint constraint)

composeWith

constraint: CommodityConstraint The constraint with which to compose this constraint.
Raises: CommodityConstraintException

Compose this constraint with another constraint to create a new constraint. This constraint and the supplied constraint must have a common commodity, otherwise a CommodityConstraintException is raised.

Return a new constraint, c such that

$$c.predicate(a_1, a_2) \Leftrightarrow \exists a_3 this.predicate(a_1, a_3) \wedge arg.predicate(a_3, a_2)$$

The amount of a_3 must be rounded to the smallest intermediate amount of the intermediate commodity, if that commodity has an intermediate result.

In practical terms, this usually means that one of the composing constraints is functional.

Reportable compatibleWith(CommodityConstraint constraint) compatibleWith
constraint: CommodityConstraint The constraint to test this constraint against.

Constraint compatibility test. Test this constraint against the argument constraint for compatibility. The two constraints are compatible if the conjunction of the constraints is satisfiable: they either refer to different commodities, or there is at least one pair of amounts for both commodities that satisfies both constraints.

If the two constraints are incompatible, return a warning Reportable if either constraint is soft and an error Reportable if neither constraint is soft. If the two constraints are compatible, return a null Reportable.

2.2 CommodityConstraintGroup

A related group of commodities. Commodity groups allow the construction of a set of relationships between various commodities. These relationships can then be

used to constrain the trading of these commodities.

Commodity constraint groups can be *global*, indicating that they apply to all transactions. Since global constraint groups need to be readily accessible, only reference data commodity groups can be global.

2.2.1 Relationships

	Class	Description	Notes
⇑	Identifiable		
⇓	CommonCommodityConstraint-Group §2.3		
↓	CommodityConstraintGroup-Model §3.1		
↓	CommodityConstraintGroupReferenceDataModel §3.3		
↔	CommodityConstraintGroupReferenceDataModel §3.3	model 0..1	

⇑:Inherits ⇓:Inherited by ↓:Realized by ↔:Association →:Navigable ◊:Aggregate ⧫:Composite

2.2.2 Operations

Set<Commodity> commodities() commodities

The grouped set of commodities. Return the set of commodities that have been grouped together.

Set<CommodityConstraint> constraintsOn(Commodity commodity) constraintsOn
commodity: Commodity

The constraints on a particular commodity. Return the set of constraints that this commodity is required to obey.

Set<CommodityConstraint> constraintsBetween(Commodity commodity1, Commodity commodity2) constraintsBetween
commodity1: Commodity The first commodity to be constrained.
commodity2: Commodity The second commodity to be constrained.

The set of constraints between two commodities. Return the set of constraints that this commodity group imposes between commodity1 and commodity2. Both commodity1 and commodity2 are order-independent.

Boolean isGlobal() isGlobal

Is this a global constraint? Return true if this constraint group is globally applicable, false if this constraint group only applies to the current environment.

2.3 CommonCommodityConstraintGroup

A constraint group where all constraints are defined in terms of a functional, non-soft constraint against a common commodity. The constraint that applies to two (non-common) commodities within the group is found by composing the two constraints across the common commodity.

Since all constraints are functional, the common commodity group can be considered to be a commodity in its own right, with the common commodity acting as a stand-in for the group as a whole.

2.3.1 Relationships

	Class	Description	Notes
⇑	CommodityConstraintGroup §2.2		
⇑	Commodity		
↓	CommonCommodityConstraintGroup-Model §3.2		

⇑:Inherits ↓:Realized by

2.3.2 Operations

PrimitiveCommodity commonCommodity() commonCommodity

The common commodity. Return the commodity that is used as a common commodity.

Set<CommodityConstraint> constraintsOn(Commodity commodity) constraintsOn
commodity: Commodity

The constraints on a particular commodity. Return the set of constraints that this commodity is required to obey. This set of constraints includes constraints composed across the common commodity with all other constraints in the group.

Set<CommodityConstraint> constraintsBetween(Commodity commodity1, Commodity commodity2) constraintsBetween
commodity1: Commodity The first commodity to be constrained.
commodity2: Commodity The second commodity to be constrained.

The set of constraints between two commodities. Return the set of constraints that this commodity group imposes between commodity1 and commodity2. Both commodity1 and commodity2 are order-independent.

If neither commodity1 or commodity2 are the common commodity, then construct the constraints by composing all constraints from commodity1 to the common commodity with all constraints from commodity2 to the common commodity.

Number smallestPhysicalAmount() smallestPhysicalAmount

The smallest physical amount of this commodity. Delegate to the common commodity.

Number smallestExchangableAmount() smallestExchangableAmount

The smallest exchangeable amount of this commodity. Delegate to the common commodity.

Number smallestIntermediateAmount() smallestIntermediateAmount

The smallest amount of this commodity to use as an intermediate value in calculations. Delegate to the common commodity.

Enumeration roundingConvention() roundingConvention

How to round amounts of this commodity. Delegate to the common commodity.

Integer decimals() decimals

The number of decimal places to print an amount of this commodity to. Delegate to the common commodity.

printAmountFormal(OutputStream stream, Number amount, Language language) printAmountFormal

stream: OutputStream The stream to print onto.
amount: Number The amount of the commodity.
language: Language The language to use for numeric formatting conventions. The default value is Language.local().

Print some amount of this commodity in a formal manner. Delegate to the common commodity.

printAmountInformal(OutputStream stream, Number amount, Language language) printAmountInformal

stream: OutputStream The stream to print onto.
amount: Number The amount of the commodity.
language: Language The language to use for numeric formatting conventions. The default value is Language.local().

Print some amount of this commodity in an informal manner. Delegate to the common commodity.

3 Classes

3.1 CommodityConstraintGroupModel

An implementation of the CommodityConstraintGroup interface. This class is implemented as an aggregation of a set of constraints on various commodities.

3.1.1 Relationships

	Class	Description	Notes
↑	CommodityConstraintGroup §2.2		
↑	Validatable		
⇓	CommonCommodityConstraintGroup-Model §3.2		
↔	CommodityConstraint §2.1	constraints 0..n	→

⇓:Inherited by ↑:Realizes ↔:Association →:Navigable ◊:Aggregate ♦:Composite

3.1.2 Attributes

identifier: String Unique identifier (may be nil if this is a non-reference data constraint group).

3.1.3 Operations

Reportable validate() validate

Validate this constraint group. Return the compositions of testing every constraint in the aggregation against every other constraint in the aggregation.

Set<Commodity> commodities() commodities

The grouped set of commodities. Return the union of all commodities specified in the aggregated constraints.

Set<CommodityConstraint> constraintsOn(Commodity commodity) constraintsOn
commodity: Commodity

The constraints on a particular commodity. Return those constraints in the aggregation that have commodity as one of their commodities.

Set<CommodityConstraint> constraintsBetween(Commodity commodity1, Commodity commodity2) constraintsBetween
commodity1: Commodity The first commodity to be constrained.
commodity2: Commodity The second commodity to be constrained.

The set of constraints between two commodities. Return the set of constraints in the aggregation that have both commodity1 and commodity2 as commodities.

Boolean isGlobal() isGlobal

Is this a global constraint? Return false. Only constraint groups that are reference data can be global.

3.2 CommonCommodityConstraintGroupModel

An implementation of the CommonCommodityConstraintGroup interface. The common commodity is stored as an attribute.

3.2.1 Relationships

	Class	Description	Notes
⇑	CommodityConstraintGroupModel §3.1		
↑	CommonCommodityConstraintGroup §2.3		

⇑:Inherits ↑:Realizes

3.2.2 Attributes

commonCommodity: PrimitiveCommodity The common commodity.

3.2.3 Operations

Reportable validate() validate

Validate this constraint group. In addition to superclass validation, all constraints in the aggregation must have the common commodity as one commodity, all constraints in the aggregation must be functional and must not be soft.

3.3 CommodityConstraintGroupReferenceDataModel

An implementation of the CommodityConstraintGroup interface as a piece of reference data. This reference data wraps an instance implementing either the CommodityConstraintGroup §2.2 interface or the CommonCommodityConstraintGroup §2.3 interface, with most implementations of the operations defined in the interface being delegated to the associated model.

Use of the reference data models provides a common depot for the storage and retrieval of constraint groups. Only reference data commodity group constraints can be global.

3.3.1 Relationships

	Class	Description	Notes
⇑	ReferenceDataModel		
↑	CommodityConstraintGroup §2.2		
↔	CommodityConstraintGroup §2.2	model 1..1	→

⇑:Inherits ↑:Realizes ↔:Association →:Navigable ◊:Aggregate ♦:Composite

3.3.2 Attributes

isGobal: Boolean = false Indicates whether this constraint group must be applied across all transactions.

3.3.3 Operations

Boolean isGlobal() isGlobal

Is this constraint group globally valid? Return the isGlobal attribute.

Note that reference data models for commodity constraint groups can be global, something that is prohibited in “bare” commodity constraint groups.

«Static Method» Set<CommodityConstraint> globalConstraintsOn(Commodity commodity) globalConstraintsOn

commodity: Commodity

The constraints on a particular commodity. Return the union of all constraints on the commodity from all global constraint groups.

«Static Method» Set<CommodityConstraint> globalConstraintsBetween(Commodity commodity1, Commodity commodity2) globalConstraintsBetween

commodity1: Commodity The first commodity to be constrained.
commodity2: Commodity The second commodity to be constrained.

The set of constraints between two commodities. Return the union of all constraints between commodity1 and commodity2 from all global constraint groups.

3.4 CommodityConstraintModel

An abstract implementation of the CommodityConstraint interface.

Subclasses implement the necessary constraints, using whatever domain-specific knowledge applies to the commodities. For example, constraints that peg one currency to another are part of the FX packages.

3.4.1 Relationships

	Class	Description	Notes
↑	CommodityConstraint §2.1		
⇓	ComposedCommodityConstraint-Model §3.5		

⇓:Inherited by ↑:Realizes

3.4.2 Operations

CommodityConstraint composeWith(CommodityConstraint constraint) composeWith

constraint: CommodityConstraint The constraint to compose with.
Raises: CommodityConstraintException

Compose this constraint with another constraint to create a new constraint. Create a new ComposedCommodityConstraintModel §3.5 with this constraint and the argument constraint as the first and second constraints.

3.5 ComposedCommodityConstraintModel

A constraint constructed from two other constraints.

3.5.1 Relationships

	Class	Description	Notes
⇑	CommodityConstraintModel §3.4		
↔	CommodityConstraint §2.1	second 1..1	→
↔	CommodityConstraint §2.1	first 1..1	→

⇑:Inherits ↔:Association →:Navigable ◊:Aggregate ♦:Composite

3.5.2 Operations

Boolean predicate(Number amount1, Number amount2) predicate
amount1: Number The amount of the first commodity.
amount2: Number The amount of the second commodity.

The constraint. Return true if

$$\exists a \quad firstConstraint.predicate(amount1, a) \wedge secondConstraint.predicate(amount2, a)$$

using the intermediate rounding practises of a, or, similarly, for other arrangements of the first and second constraints and common commodities.

Note that intermediate rounding can mean subtle variations of the values accepted by predicate() and the values produced by firstAmountFrom() and secondAmountFrom(). To prevent ambiguity, if both the first and second constraints are functional, then predicate is defined to be true if the amounts match either the calculation from firstAmountFrom() or the calculation from secondAmountFrom(). This situation is unfortunate, but reflects the difficulties of limited-precision arithmetic.

SimpleCashflow firstAmountFrom(SimpleCashflow a) firstAmount-From
a: SimpleCashflow The amount of the second commodity.
Raises: CommodityConstraintException

Convert an amount of the second commodity into an amount of the first commodity.

Assuming a suitable arrangement of the first and second constraints and common commodities, calculate first $x = firstConstraint.firstAmountFrom(a)$, then $x' = round(x)$ using the common commodity's intermediate rounding practices and then return $secondConstraint.secondAmountFrom(x)$ where x is the amount in the common commodity. Similar arrangements hold for other constraint placements and common commodities.

SimpleCashflow secondAmountFrom(SimpleCashflow a) secondAmount-From
a: SimpleCashflow The amount of the first commodity.

Raises: CommodityConstraintException

Convert an amount of the first commodity into an amount of the second commodity.

Assuming a suitable arrangement of the first and second constraints and common commodities, calculate first $x = secondConstraint.firstAmountFrom(a)$ then calculate $x' = round(x)$ using the common commodity's intermediate rounding practices and then return $firstConstraint.secondAmountFrom(x)$ where x is the amount in the common commodity. Similar arrangements hold for other constraint placements and common commodities.

Boolean isSoft() isSoft

Is this a soft (non-mandatory) constraint?

$$this.isSoft \Leftrightarrow firstConstraint.isSoft \vee secondConstraint.isSoft$$

Boolean isFunctional() isFunctional

Can this constraint be turned into a transformation function?

$$this.isFunctional \Leftrightarrow firstConstraint.isFunctional \wedge secondConstraint.isFunctional$$

4 Exceptions

4.1 CommodityConstraintException

An exception raised whenever a CommodityConstraint §2.1 cannot be imposed or composed.

4.1.1 Operations

CommodityConstraint constraint() constraint

The constraint which could not be used.

5 Associations

Table 1: Basic Commodity Constraints— Associations

Association Role	Class	Card.	Notes
second			
secondConstraint	CommodityConstraint §2.1	1..1	→
composition	ComposedCommodityConstraint-Model §3.5	0..n	
first			
firstConstraint	CommodityConstraint §2.1	1..1	→
composition	ComposedCommodityConstraint-Model §3.5	0..n	
model			
model	CommodityConstraintGroup §2.2	1..1	→
wrapper	CommodityConstraintGroupReferenceDataModel §3.3	0..1	
constraints			
constraints	CommodityConstraint §2.1	0..n	→
group	CommodityConstraintGroup-Model §3.1	0..n	◊

→:Navigable ◊:Aggregate ♦:Composite

5.1 second

Role: secondConstraint *Navigable* CommodityConstraint, 1..1.
Role: composition ComposedCommodityConstraintModel, 0..n.
The second constraint to compose with.

5.2 first

Role: firstConstraint *Navigable* CommodityConstraint, 1..1.
Role: composition ComposedCommodityConstraintModel, 0..n.
The first constraint in the composition.

5.3 model

Role: model *Navigable* CommodityConstraintGroup, 1..1.
Role: wrapper CommodityConstraintGroupReferenceDataModel, 0..1.
The model for the reference data.

5.4 constraints

Role: constraints *Navigable* CommodityConstraint, 0..n.
Role: group *Aggregate* CommodityConstraintGroupModel, 0..n.
The constraints that make up a constraint group.

Figure 1: Class Diagram— Examples

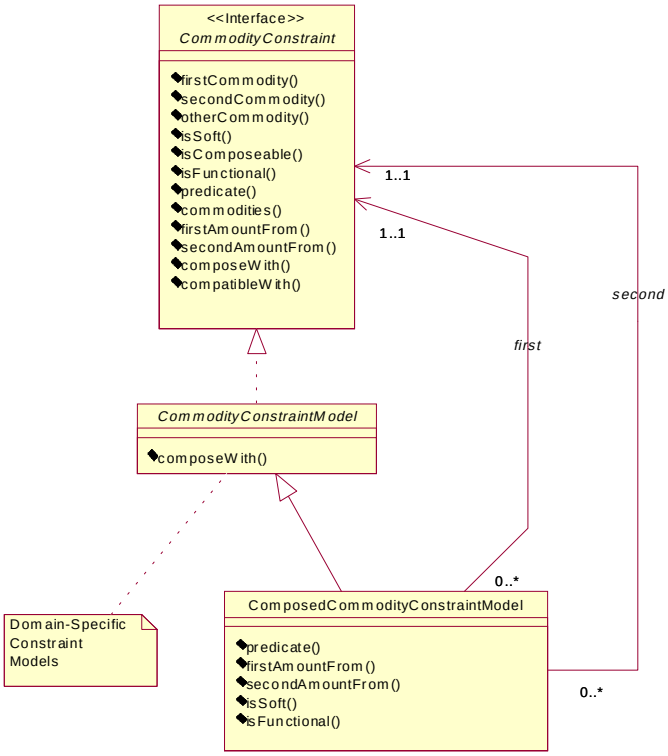


Figure 2: Class Diagram— Commodity Constraints

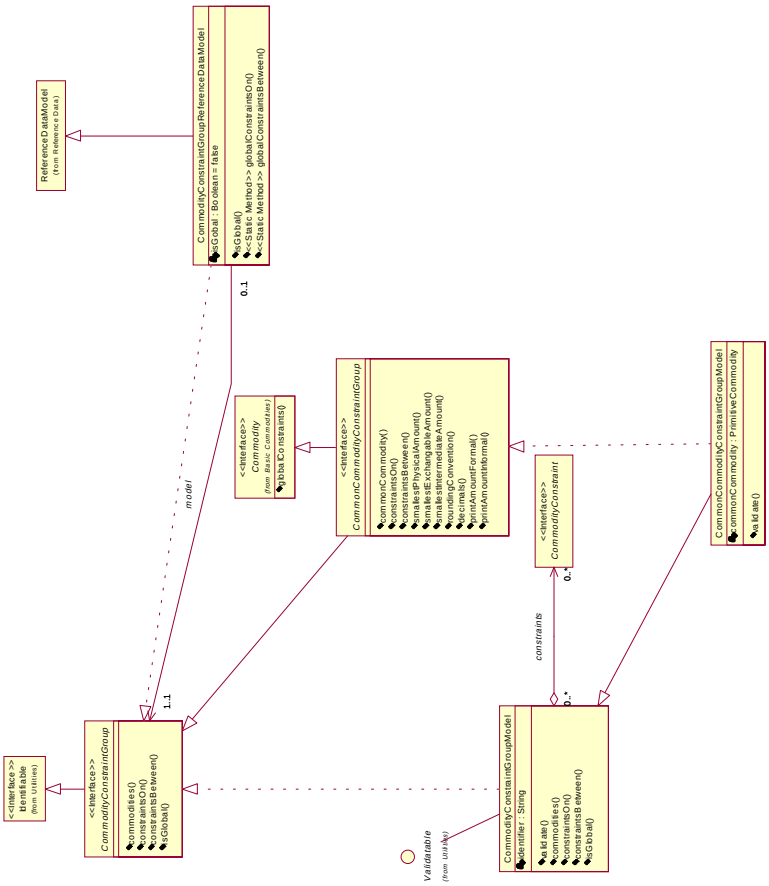


Figure 3: Class Diagram— Commodity Constraint Groups

6 Extensions to the Basic Commodities Package

6.1 Commodity

6.1.1 Relationships

	Class	Description	Notes
⇓	CommonCommodityConstraint-Group §2.3		

⇓:Inherited by

6.1.2 Operations

Set<CommodityConstraint> globalConstraints() globalConstraints

The set of constraints on this commodity.
Return the set of global constraints that apply to this commodity.

References

[1] *The Association for the Monetary Union of Europe*. http://amue.lf.net.