



LÆXON  
**LORE**

**Writing Lexon 0.3**

LÆXON  
**MANUAL**

0.3

This manual describes the workflow to create and use Lexon digital contracts on the Æternity blockchain. It covers writing, compiling and deploying of Lexon texts, and how to use the Lexon Æternity Token, LÆX.

## DISCLAIMERS

The information provided in this document is strictly for educational purposes. There are no warranties, express or implied. Any use of this information is at your own risk. The author does not assume and hereby disclaims any liability to any party for any loss, damage, or disruption. See <https://www.gnu.org/licenses/gpl-3.0.txt>

Lexon is not an all-purpose human language. An unambiguous language is desirable for programming and lawmaking but less so for other purposes of human communication.<sup>1</sup>

Lexon compiler output must be audited before using it in production. There is no warranty for fitness for any purpose, nor any other warranty, for the compiler output. See the License text at <https://lexon.org/license>.

The described tokens are not for investment; they may not work as a store of value. There may be no secondary market for the tokens. The token is not bought back by the issuer. The token does not represent a share in a company or IP. It does not make eligible for any payment.

## LICENSE

There is no claim to the products of the Lexon compiler. Any text you write in Lexon and anything you create using the Lexon compiler is yours or determined by arrangements you made.

This document, including its appendices, is licensed under the GNU Public License (GPL) version 3. The license text can be found at <https://www.gnu.org/licenses/gpl-3.0.txt>. Basically, you can quote, share or modify this document but must give credit and allow the same.

---

<sup>1</sup> Cf. appx. *The Principles of Newspeak* in G. Orwell, 1949, *Nineteen-Eighty-Four*. Orwell essentially argues that words must be ambiguous to be meaningful.

# INDEX

DISCLAIMERS.....	4
LICENSE.....	4
INDEX .....	5
<b>INTRODUCTION .....</b>	<b>6</b>
<b>QUICK START.....</b>	<b>7</b>
<b>MOVING PARTS.....</b>	<b>8</b>
<b>AUTHORING.....</b>	<b>9</b>
TAXONOMY OF DIGITAL CONTRACTS .....	9
GRAMMAR .....	11
THE DOUBLE EDGE OF LANGUAGE .....	18
NEXT STEPS .....	19
VOCABULARY .....	20
WORD LIST .....	23
EXAMPLES .....	40
<b>COMPILER.....</b>	<b>47</b>
OPERATION .....	47
EXAMPLE .....	48
OPTIONS .....	50
INTERNAL MODEL .....	52
TOKEN .....	53
<b>TUTORIAL .....</b>	<b>54</b>
PREREQUISITES .....	54
COMPILATION.....	55
DEPLOYMENT .....	57
<b>APPLICATION .....</b>	<b>63</b>
<b>CONCLUSION .....</b>	<b>65</b>
<b>APPENDIX .....</b>	<b>66</b>
LEXON GRAMMAR FORM.....	66
HARDENED EXAMPLE .....	67
ROBOTIC LAWS .....	67
<b>INDICES.....</b>	<b>68</b>
WORD LIST .....	69

---

# INTRODUCTION

---

Lexon is a plain-text programming language. This means that it reads like natural English and *digital contracts* written in Lexon that run on the blockchain can be read and understood by anyone, without requiring any knowledge of programming.

Lexon's digital contracts inherit their unstoppable power from smart contracts. That they are also readable by anyone – not just programmers – is their interface to the real world. They connect to the legal system, for far-reaching consequences: a digital contract cannot be broken and is a legal agreement.

To work its magic, Lexon perfects an AI language processing approach that had been researched for decades. Its contribution is 'Zen-like,' reducing permutation steps to stay closer to how human thought works: its internal data model retains human-readability. This makes Lexon transparent as well as precise and provides unparalleled agency.

The example contract below is a minimal escrow that is an agreement between a *payer* of  $\mathbb{A}$  coins, and a *receiver*. The *notary* decides whether the tokens should be paid out to the receiver or sent back to the payer. This could be the reaction to a corresponding deal being aborted, goods not arriving or being returned for any reason. The contract looks only at the payment side, with the notary in the role of the *oracle*, i.e. the connection from the blockchain to the real world.

As is, the contract could serve many different use cases. There could be many variations, including a scenario that works without oracles.

LEX Escrow.

"Payer" is a person.

"Receiver" is a person.

"Notary" is a person.

"Fee" is an amount.

The Payer pays an Amount into escrow, appoints the Receiver, appoints the Notary, and fixes the Fee.

CLAUSE: Pay Out.

The Notary may pay from escrow the Fee to themselves, and afterwards pay the remainder of the escrow to the Receiver.

CLAUSE: Pay Back.

The Notary may pay from escrow the Fee to themselves, and afterwards return the remainder of the escrow to the Payer.

*Source 1 – A Lexon digital contract example: Escrow.*

In the following chapters, we will discuss this example's grammar, compile this digital contract using the online Lexon compiler and deploy it to the blockchain, and interact with it, using *Æ Studio* at <https://studio.aepps.com/>.

---

# QUICK START

---

As an advanced crypto user, all you need to get started might be this:

To write your own digital contract, take an example and modify it. To understand the vocabulary and grammar, dive into the interactive vocabulary and <https://lexon.org/vocabulary> and use color highlighting.

To deploy and use a Lexon digital contract on the Æternity blockchain:

1. Paste a Lexon text into the text field of the **Lexon compiler** at <http://lexon.org/compiler>.
2. Click **compile**.
3. Copy + paste the result into a new source tab in the **Æ Studio** at <https://studio.aepps.com>.
4. Connect with the **Superhero** Æ wallet (or use the testnet without wallet).
5. Set an amount of Æ at the top, fill in the required fields below the **deploy** button + click deploy.
6. Interact with the contract using Æ Studio's function calls that appear below the deploy button.
7. Check state changes in Æ Studio's log and at the blockchain browser **aeScan** at <http://aescan.io>.

These steps are described and illustrated in detail in the tutorial from page 9.

---

# MOVING PARTS

---

## Lexon

Lexon unites developments in computational law, cryptography, computer sciences, AI<sup>2</sup> and linguistics<sup>3</sup> to achieve long-sought milestones in each field: digital contract analysis, legally enforceable smart contracts, self-documenting code, deterministic language processing, and an executable human language. The resulting accessibility and agency open new ways even to *think* about some of the more intractable-looking challenges of our times and solve them. It is perfect for *trustless private law*, i.e., legally enforceable agreements implemented on the blockchain.

## Digital Contracts

Blockchain smart contracts written in Lexon are called *digital contracts*. While lawmakers will need time to understand their new options, Lexon shines as a language for *private law*, i.e., contracting. Digital contracts are *legally enforceable* agreements.

## Æternity

Æternity is a layer-1 blockchain that is particularly well suited for fast and economic smart contracts.<sup>3</sup> It improves on Ethereum's functionalities, cleaning up concepts, reducing pitfalls and reduces response times to the point where it becomes feasible that end users of apps can interact directly with the chain. Æternity was started by a co-inventor of Ethereum, and implemented by most experienced telco and fintech engineers.

## Sophia

Sophia<sup>4</sup> is the language that smart contracts are programmed in for the Æternity blockchain. It is a purpose-built, 3<sup>rd</sup> generation functional language, designed to be as clear and safe as possible. Lexon users, however, do not need to learn Sophia to be able to create smart contracts.

## Lexon Compiler

The Lexon compiler<sup>5, 6</sup> accepts text adhering to Lexon's grammar and transposes this natural-language code to the functional language *Sophia*. It is a web3 æpp interacting with the Æternity blockchain to help creating new web3 æpps for this chain. It's payment mechanism is also implemented on Æternity.

## LÆX Tokens

Lexon Æternity tokens (LÆX)<sup>7</sup> provide metered access to the online Lexon compiler. One token buys one compilation of a Lexon text. The token is implemented as a smart contract on the Æternity blockchain and conforms to its fungible token standard AEX-9. It can be purchased at <https://lexon.org>.

---

<sup>2</sup> Lexon falls into the mould of symbolic AI, not machine learning (ML). Contrary to ML, Lexon is designed to provide agency, transparency, and precision. ML is complementary to Lexon: Lexon is a powerful device to control ML, while ML in turn will soon help creating Lexon texts.

<sup>3</sup> See <https://aeternity.com/aeternity-101>

<sup>4</sup> See <https://aeternity.com/#sophia>

<sup>5</sup> A compiler is basically a program that helps create other programs. It processes human-written files to create output that can be executed by a computer.

<sup>6</sup> Online at <http://lexon.org/compiler>

<sup>7</sup> See *Token*, from pg. 22.

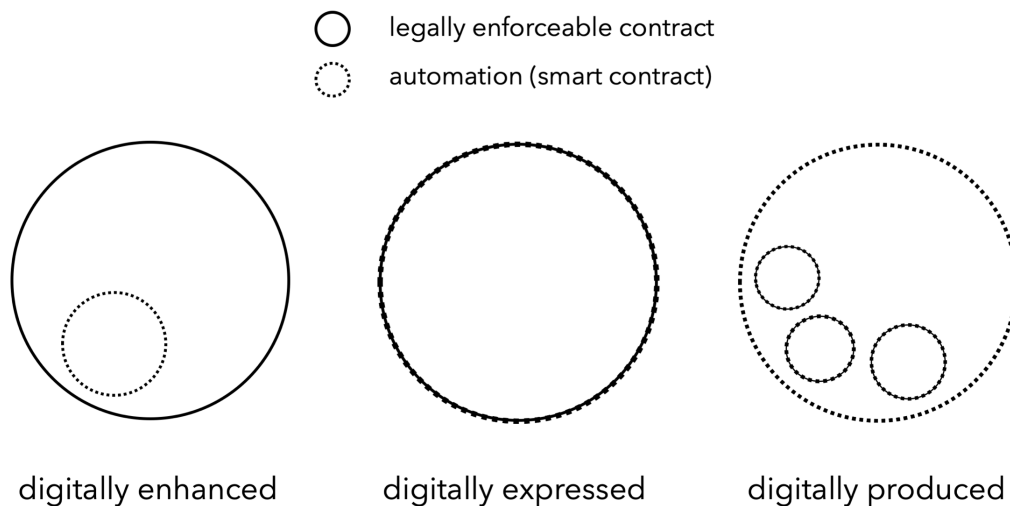


# AUTHORING

This chapter discusses the linguistics of Lexon, its grammar & vocabulary.

## TAXONOMY OF DIGITAL CONTRACTS

If we call a program running on a blockchain a ‘smart contract’, and the contract as lawyers know it ‘legally enforceable contract’ then we have three cardinal relationships that should be differentiated:<sup>8</sup>



**Digitally enhanced:** a legally enforceable contract is in part automated by a smart contract. The legally enforceable contract ‘includes’ the smart contract and conventional prose spells out parts that are outside the scope of the smart contract.

**Digitally expressed:** the smart contract is the legally enforceable contract. The code of the smart contract is the entire text of the legally enforceable contract. This becomes possible through the use of the Lexon language.

**Digitally produced:** a smart contract running on the blockchain initiates a multitude of legally enforceable contracts, one with each person interacting with the smart contract, which we will call a **Contract Factory**. This is a common pattern that holds e.g., for a crowdfunding smart contract.

In short:

<b>digitally enhanced</b>	→ the program is a part of a contract.
<b>digitally expressed</b>	→ program and contract are the same.
<b>digitally produced</b>	→ the program produces contracts.

Table 1 – Types of Digital Contracts

<sup>8</sup> This graphic & pg. 8 - 9 are in the public domain. 2020, H. Diedrich, C. L. Reyes.

There are nuances and overlaps but it is important to note that smart contracts and legally enforceable contracts are neither necessarily the same nor necessarily two different documents.

## Digital Enhancement

The typical ‘Ricardian Contract’<sup>9</sup> setup joins traditional contract prose with a blockchain smart contract, e.g. written in Ethereum’s Solidity and thus not readable for non-programmers. In this way, human-readable prose is joined with a blockchain component that will automatically perform part of the agreement. An example could be a loan with collateral where the exact conditions of the contract are laid out in the traditional contract’s prose, while the payments due for repayment are calculated by a smart contract and automatically deducted from the lender’s Ether<sup>10</sup> account. The off-chain prose might deal with exceptions e.g. the case that the lender stops access to his account. This constellation then is what we call *digital enhancement*. Lexon code can be used for these situations, too. And in fact, we predict that it will replace Solidity for most such cases because it will have strong upsides to do so.

## Digitally Expressed

But the novelty with Lexon is that the legal contract prose can now itself be the program that is executed on the blockchain. Program and contract can virtually be the same as can be observed in the example on pg. 6, where a simple escrow agreement is articulated in Lexon, with the document serving the dual purpose of expressing the ‘meeting of the minds’ on the one hand but being a program on the other, ready for deployment to the blockchain as is. We call this *digitally expressed*, because there is now only one document that serves as legally enforceable contract that one would show to a judge if needed, and doubles as program on the blockchain.

## Digitally Produced & Contract Factory

Very often, however, Lexon code will be used to program a system that offers multiple individuals to enter into contracts, which are each created ad hoc, e.g. at the time a prospect signs off on a purchase, or a membership in a DAO. In this case, one smart contract results into multiple legal agreements. A useful example for this pattern is a ticket vending machine: such a machine can extend an offer to potential buyers of a ticket, e.g. for public transport. When a buyer puts money into the machine it will ‘decide’ whether to issue the ticket or not. The money has to be enough, the machine needs to check some other conditions, e.g. whether it still has enough paper to print on. Likewise, a smart contract always has the ‘last word’ whether it will initiate an agreement based on the user input, or not. It can send money back that was sent to it if a condition is not met as needed. The money might be too little, or the deadline or a ceiling for a crowdfunding drive might have been passed. If all is good, the smart contract will accept the offer of the user and a legal agreement commences, usually between the user and the creator of the smart contract. The result is a set of many individual cookie cutter contracts e.g. between many buyers and one seller. We want to call smart contracts that act like this *Contract Factories* – borrowing from a well-known pattern name in computer sciences – and the individual legally enforceable contracts *digitally produced*.

---

<sup>9</sup> Ian Grigg, 1996 – [https://en.wikipedia.org/wiki/Ricardian\\_contract](https://en.wikipedia.org/wiki/Ricardian_contract)

<sup>10</sup> Ether is the name of the crypto currency of the Ethereum blockchain.

## GRAMMAR

### Names

Names can consist of multiple words, i.e., include spaces. Clauses are often named for partial sentences, e.g., "**Service Performed as Agreed**," so that they can organically be built into other sentences.<sup>11</sup> This is a major pillar on which Lexon's readability rests. Within those names, no restrictions apply. The effective vocabulary across Lexon contracts is therefore of unlimited size.

**Changing the definition names does not change the logic of the contract.**

LEX Payment.

"Payer" is a person.

"Payee" is a person.

"Payment" is an amount.

The Payer pays a Payment to the Payee.

This is a payment, nothing more. The Payer pays an amount to the Payee. This is not even 'really' a contract, because it is so simple.

LEX Transfer.

"**Sender**" is a person.

"**Receiver**" is a person.

"**Sum**" is an amount.

The Sender pays a Sum to the Receiver.

This is the same contract as above, just spelled out using different names. When this contract is signed and deployed to a blockchain, the persons named will have to be named with their real names or at least a blockchain address, to clearly identify them.

That comes later though. At the time of writing, this document is a template and the names defined in it are placeholders. All that is known is that they must be a person or an amount.

---

<sup>11</sup> ☺

### *Synonyms*

If you use a synonym verb, it does not change the logic of the contract.

LEX Transfer.

"Sender" is a person.  
 "Receiver" is a person.  
 "Sum" is an amount.

The Sender **transfers** a Sum to the Receiver.

This is the same contract as the previous one, just spelled out using a different *verb* with the same meaning: **transfer** instead of **pay**.

Note that the verbs in Lexon are predefined and few. You cannot just invent them as with the names (nouns). I.e. the names 'Sender', 'Receiver', and 'Sum' can be replaced by almost any other words you can come up with. But for 'pay' the synonyms are precisely 'transfer' and 'return.' No other words will work.

This is a fundamental difference between nouns and verbs in Lexon. Nouns can be chosen freely, verbs need to be used as intended, looked up in examples or the reference to see what will work.

### *Neutral Names*

LEX Transfer.

"A" is a person.  
 "B" is a person.  
 "C" is an amount.

A transfers C to B.

This is the same contract as the previous one, just reducing the definitions to neutral one-letter names.

### *Articles*

Articles (**a, an, the**) can be left out.

LEX Payment.

"Payer" is person.  
 "Payee" is person.  
 "Payment" is amount.

Payer pays Payment to Payee.

Articles and some other words in Lexon are called 'fillers'. They have a big role in making a text easy to read for a human being but are irrelevant to the automation of the contract on the blockchain. Obviously, articles can fundamentally change the meaning of a contract to the human reader. It's on the writer to not abuse them. Reining in the possibilities for abuse of fill words is a high priority for future Lexon tools (cf. pg. 18).

## Sentence Structure

Lexon's basic sentence grammar follows that of English, requiring, in this order: *subject, verb, object*. Verb and object are grouped together as *predicate*.

In the boxes below, square brackets [ ] mean 'optional' and the ellipsis ... means 'potentially more of the same'.

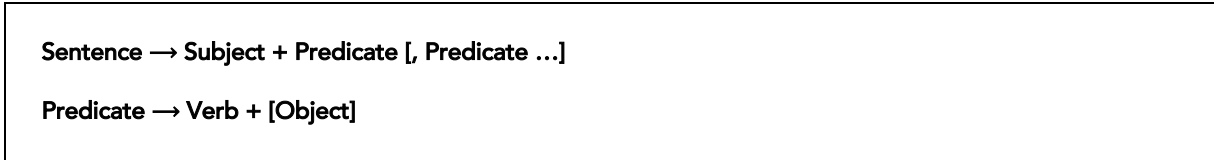


Fig 1 – Lexon Sentence Rule

These sentences are the main carrier of information in Lexon code. They form the body of **RECITALS** and **CLAUSES**.

The choice of verbs in Lexon is very restricted, while subject and object can each be any blockchain addresses – or legal person for that matter. Within a Lexon contract they will be given an arbitrary name alluding to its function (e.g. **Payer**), which adds meaning for the reader.

The freedom to name variables any way you want is a trait Lexon shares with all modern programming languages. Lexon goes further towards readability by not requiring any artificial style, like Camel Case or Snake Case.<sup>12</sup> It also allows spaces as part of the names, which enhances readability markedly.

Lexon also knows a number of passive constructs that operate on any given subject, e.g.: something **is certified**.

## Document Structure

Technically, the Lexon *document structure* is part of its '*grammar*', because that's how computer languages are defined. This concerns everything beyond those parts that are corollaries of natural language grammar. There is no such thing as a document structure in natural language, but there is in both contracting and programming.

On the highest level, the Lexon code can be embedded into legal contract prose. *Within* the Lexon parts then, the basic structure is:



Fig 2 – Lexon Simple Document Rule

<sup>12</sup> Originally as convention in programming, the style of writing variable names: e.g., as **firstName** (Camel Case) or **first\_name** (Snake Case). The intent is to leave out spaces. Lexon though allows to write: **First Name**.

The Escrow example reflects these parts:

<b>LEX Escrow</b>	<b>Head</b>
"Payer" is a person. "Payee" is a person. "Arbiter" is a person. "Fee" is an amount.	<b>Definitions</b>
The Payer pays an Amount into escrow, appoints the Payee, appoints the Arbiter, and fixes the Fee.	<b>Recital</b>
CLAUSE: Pay Out. The Arbiter may pay from escrow the Fee to themselves, and afterwards pay the remainder of the escrow to the Payee.	<b>Clause</b>
CLAUSE: Pay Back. The Arbiter may pay from escrow the Fee to themselves, and afterwards return the remainder of the escrow to the Payer.	<b>Clause</b>

Source 2 – Lexon document structure

This order makes it harder to write ambiguous agreements. It reflects a common sequence of the parts of a paper contract.

The internal model that the compiler creates during the translation is shown in chapter *INTERNAL Model*, pg. 52. It visualizes the relationships that the compiler actually ‘understands’ from the sentence in *Source 13*, expressing a linguistic structure as a binary tree.

The reduced grammar of Lexon forces sentences to be written straightforwardly, even when nested and verbose. The fact that the grammar is parseable by a computer guarantees mathematical unambiguity even though many redundant ways of expressing the same meaning have been enabled. The grammar still provides a one-way funnel; the flexibility is not bidirectional: the same can be articulated in many different ways but each way has only one meaning. It is exactly this that is achieved by limiting English grammar to a *controlled grammar*.

A minimal contract can be very short and only needs to have the **HEAD** and one sentence of **RECITAL** – or instead of a recital at least one **CLAUSE**.

However, as spelled out below, more complex **Contract Factories** (pg.10) will see the pattern of **head, definitions, recital, clauses** repeated multiple times over: first within a section called **TERMS**, then within one or more **CONTRACTS** sections.

The **TERMS** define all aspects that are true for the entire digital contract code. The **CONTRACTS** describe individual agreements between only two parties. If more than one type of such agreement is part of the system, there will be as many **CONTRACTS** sections.

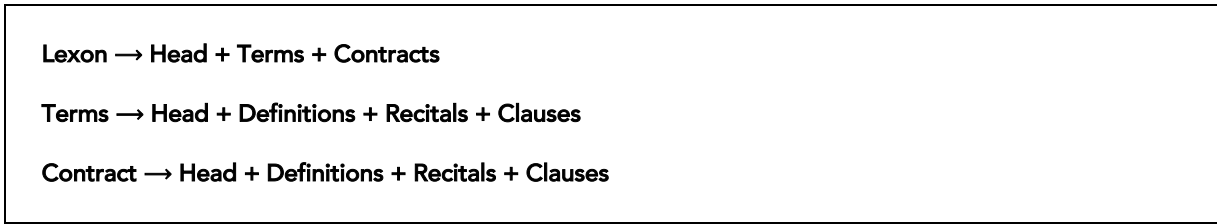


Fig 3 – Lexon Complete Document Rules

This can be visualized as follows:

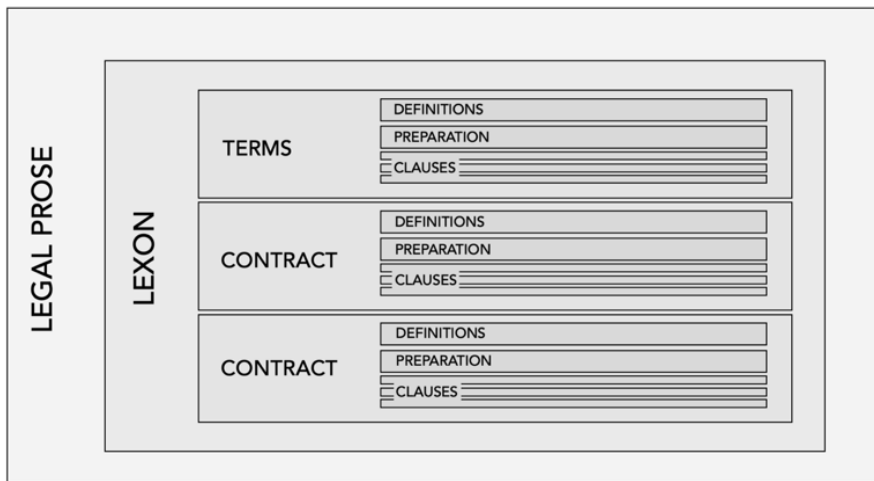


Fig 4 – Lexon Complete Document Rules (graphical)

Most elements given above are optional. Many digital contracts will be simple. This structure is anything but random though and carries the more complex ones.

Within the individual **CLAUSES**, the pattern is:<sup>13</sup>



Fig 5 – Lexon Clause Rule

<sup>13</sup> This is due to change in the next version of Lexon.

The above rule can also be expressed visually as follows:

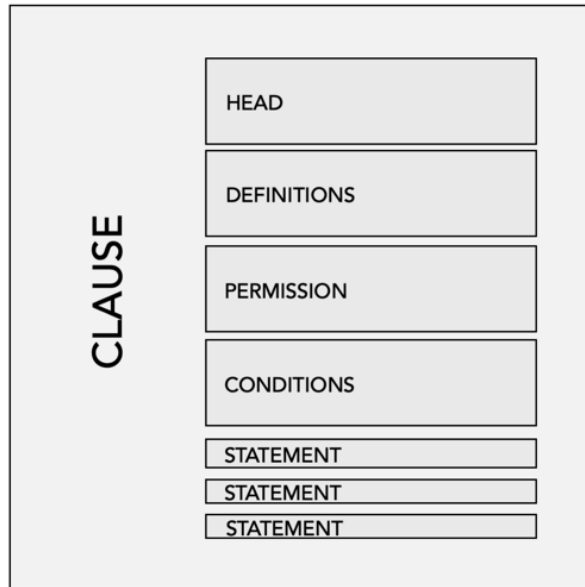


Fig 6 – Lexon Clause Rule (graphical)

Beyond these rules, some terms in the Lexon vocabulary are irregular and have to be learned for each individual term in the vocabulary. They are not used in surprising ways but cannot be described by a pattern and sometimes cannot be used in all ways that natural English would allow for.

## Example

The escrow example from pg. 6 consists of four parts:

- **head**
- **definitions**
- **recitals**
- **clauses**

### Head

LEX Escrow Contract.

The **head** consists of the **LEX** keyword that marks the beginning of executable code, and the freely given name after this keyword (in this case **Escrow Contract**) that identifies this contract for filing and maintenance purposes. There can be more information spelled out in the head, such as a revision number and a preamble, or a comment.

To have a *keyword* like **LEX** is useful also for the legal perspective of a digital contract. In the case that the Lexon code is embedded, e.g. as schedule of a larger master agreement, it provides a clear separation between the automated parts and the legal prose that might precede it. Because of this keyword, **LEX**, Lexon digital contracts are NOT entirely seamlessly embedded in the larger document prose that may surround them. But if push comes to shove, a judge would at any rate never be completely ignorant of the fact that there is automation in play with a digital contract. Therefore, it will only help to have a clear indication of where the text relevant for automation starts, to reduce legal attack vectors.



An optional **LEXON** tag can occupy the next line. If it exists, it is followed by a version number that indicates with which version of Lexon the code will work. This is a concession to the fact that Lexon is software and evolving at a rapid pace. Like the name after LEX, this number simply helps keeping order.

A **PREAMBLE** is also optional. This keyword is followed by a high-level description of the contract. In legalese, the 'preamble' is the introduction to a contract that gives context and motivation but is itself not legally binding text. In Lexon, this text is neither legally binding nor part of the automation.

Such an extended head could look like this:

```
LEX Escrow Contract.
LEXON 0.2
PREAMBLE: This is a simple digital contract example.
```

### Definitions

```
"Payer" is a person.
"Payee" is a person.
"Agent" is a person.
"Fee" is an amount.
```

**Definitions** are next. They are similar to what lawyers are used to from normal contracts – and which programmers know as *type declarations*. Because this code example is really a *template* – i.e. not a concrete instance of a concrete agreement yet – the concrete name, address, or blockchain address are not yet known at the time of writing.<sup>14</sup>

Lawyers know the principle of copy-paste well, re-using contracts that have been written for one client for a different client at a later point in time. In a similar fashion, any digital contract, before it is deployed, really defines an entire *class* of possible look-alike contracts.

When a digital contract is deployed, made concrete, the real names and blockchain addresses are provided.

### Recitals

```
The Payer pays an Amount into escrow, appoints the Payee, appoints the Agent, and also fixes the Fee.
```

The **Recital**<sup>15</sup> of a digital contract is code that is performed once at the very beginning, before any clause can be executed.

This example is simplistic in that the payer sets it all up.

In a traditionally written contract, recitals list the actions taken that led the parties to enter into the agreement. Lexon recitals are similar in that they provide the prerequisite foundation for the clauses that follow. They are performed when the smart contract is signed by the creator and deployed to the blockchain.

<sup>14</sup> Also, Lexon is still in flux and lines like these will look different soon, more like one is used from contract templates in textbooks or on the Internet.

<sup>15</sup> In US law, the recital is the part of a contract that states the purpose of the agreement. It is intended to help interpreting the agreement. In the European Union, a recital is the part of a law that describes its motivation, ideally free from jargon and politics.

*Clauses***CLAUSE: Pay Out.**

The Agent may pay from escrow the Fee to themselves,  
and afterwards pay the remainder of the escrow to the Payee.

**CLAUSE: Pay Back.**

The Agent may pay from escrow the Fee to themselves,  
and afterwards return the remainder of the escrow to the Payer.

The last part here are the **clauses** that define possible outcomes. Payment is exclusively conditional on action of the *Agent* here and can only go to the *Payee* or back to the *Payer*.

## THE DOUBLE EDGE OF LANGUAGE

A caveat: there is nothing in the language itself that keeps the writer from using misleading definitions. Language is not the right level to prevent fraud. Content checks are always one level above language.

**LEX Payment.**

"Payee" is person.

"Payer" is person.

"Payment" is amount.

Payee pays Payment to Payer.

The non-sensical swap of **Payer** and **Payee** in this example will confuse readers but the automation of the contract will still work the same as in the examples before. The logical meaning of this code is identical to the one shown before. It is just the labels that are misleading. But it is misleading only to humans since the blockchain virtual machine does not understand the word 'Payer' or 'Payee' at any rate. It does not even get to see them. It just understands what the action 'pay' is.

*Lexon is not the promise – at all – that text cannot be misleading.* Lexon is the promise that smart contracts *can* be readable. And this example is only a mild instance from a wide spectrum of possible criminal abuse. Unfortunately, there are more powerful ways to make Lexon contracts be as corrupt as bad contracts in other blockchain languages: cleverly misleading definitions, convoluted text, intentional off-by-ones. But ultimately, there is no way for a technical tool to understand if even a completely *correct* contract proposes a completely fraudulent deal.

Allowing for readability is Lexon's first goal. Curbing opportunity for abuse through smart tools will be a continuous task. But note that it is only thanks to the high readability of Lexon that this question comes up in the first place. It would simply not be asked of other blockchain languages.

A judge may throw this contract out because it is going to be difficult to argue that switching the words Payer and Payee was intentional and served a purpose that both sides agreed upon. The contract will still execute 'correctly' on the blockchain, which is the reason why a future version of Lexon will have an option for a judge or arbiter who has been given forum powers to reverse a smart contract with minimal overhead and no consequences for other users of the blockchain that the contract runs on.

## NEXT STEPS

You get the basic idea at this point and might consider to check out online resources and cherry pick across the rest of this document. The online editor at <http://demo.lexon.tech> is a great place to get your own first digital contract deployed you will soon be able to actually use it for work. An online tutorial is available, with up to date examples to deepen your learning at <https://www.lexon.tech/tutorial>.

Next, we'll be looking at the vocabulary but be sure to check out the online version for it.

## VOCABULARY

Word list: see pg. 23 and <http://lexon.org/vocabulary>. The links on the webpage help a lot.

The point of Lexon is that one does not have to learn it to read it. Learning to *write* Lexon is best done by looking at examples rather than memorizing words. This is no different than how a human language, or a programming language, or even legalese is picked up. The allowed use of a word, its context, are what matters. As the Lexon grammar is a so-called *controlled grammar*, only some ways of using a word are permitted, fewer than in normal English. Studying this is better learned by doing than done by learning. In fact, it is particularly hard to 'unlearn' what you know is correct in normal language. But you will – and that is a unique quality of Lexon – get a feel for what works.

This is the list of the *known words* in Lexon 0.3<sup>16</sup> that between themselves stand up the Lexon grammar. Because you can define terms, Lexon's vocabulary, and grammar complexity are unlimited from the point of view of natural language. This is discussed below.

A AFTER AFTERWARDS ALL ALSO AMOUNT\* AN AND ANNOUNCED APPOINT  
 APPOINTS AS AT AUTHOR AUTHORS BE BEEN BEING BINARY\* CERTIFIED  
 CERTIFIES CERTIFY CLAUSE COMMENT COMMENTS CONTRACT CONTRACTS  
 CURRENT DATA\* DAY DAYS DECLARE DECLARED DECLARES DEFINED EQUAL  
 EQUALING ESCROW FILED FILE FILES FIX FIXED FIXES FOR FROM GENERAL  
 GIVEN GRANT GRANTS HAS HERSELF HIMSELF HOUR HOURS IF IN INTO IS  
 ITSELF LEAST LEX LEXON LIES MAY MILLISECOND MILLISECONDS MINUTE  
 MINUTES MONTH MONTHS MYSELF NO NOT NOW OF OFF ON ONESELF OR  
 OURSELVES PASSED PAST PAY PAYS PER PERSON\* PREAMBLE PROVIDED  
 REGISTER REGISTERS REMAINDER RESPECTIVE RETURN RETURNS SECOND  
 SECONDS SEND SENDS SIGNED SO TERMINATE TERMINATES TERMS TEXT  
 THAT THE THEMSELF THEMSELVES THEN THERE THEREFOR THEREFORE  
 THESE THIS TIME\* TO TRUE WAS WEEK WEEKS WITH YEAR YEARS YES  
 YOURSELF YOURSELVES

These words can mostly be redefined as *names* but in that case can then not serve their original function anywhere in the contract. The exception are *category* names (types, marked with an asterisk (\*) above). They cannot be redefined but can be used as generic names (see, e.g., amount). All words can be used as *part* of names without losing their original, stand-alone function.

In the individual, per-word entries below, the first information about each word is, in what linguistic capacity it is used in Lexon. This angle can help because English words frequently cover two or more different grammatical roles. The second line is technical, based on computer sciences designations for the function that a word is used for in Lexon. It will more often than not be a helpful pointer for non-programmers, too. In many cases, a description follows that describes the use of the word and occasionally presents additional context. Note that the description often seems to explain the obvious, because you know how English works. However, it can help to spell it out to learn to write Lexon. After that, one or more examples show the word in the context of an example sentence. This sentence can be inspected in the context of a complete digital contract by clicking the link immediately below the example sentence.

---

<sup>16</sup> based on grammar version 0.2.20 / subset 0.3.8 alpha 79 - English / Reyes.

At pg. 23 and at <http://lexon.org/vocabulary> you find a word list of Lexon 0.3. The webpage is a learning tool to help grasp the bigger picture, not necessarily to memorize individual words. To this end, one thousand links are at your disposal below for fast navigation between words, examples and references. You will find that clicking around reveals the deeper structure of Lexon, beyond words.

## Breadth and Capacity

Lexon's vocabulary is **unlimited** because a) any noun or compound term can be defined and used in a Lexon text, b) any phrase can be used as clause name and then used as part of a sentence elsewhere in the text, and c) the Lexon compiler is extensible and keywords can and are added while the grammar grows more powerful.

a) *Definitions:*

see Payer, Payee, etc. in the Escrow example.

b) *Clauses:*

see Noticed, Factually Breached, etc. in the Evaluation License example.

c) *Keywords:*

Cf. vocabulary 0.3 below vs. 0.2 in the 2020 Lexon Bible.

The latter is a slow and incremental process. A special, faster process has been prepared for verbs of foundational importance – like *move* for robotics – that can sustain an entire domain. The two former points are instant and happen when a user authors a Lexon text. Each Lexon digital contract therefore has its own vocabulary, extending the dictionary on the fly while drafting; as integral and organic part of the writing process. This is in keeping with the usual way that paper contracts are written: terms are being defined for clarity, laying the ground for an agreement's text. That's exactly how Lexon's vocabulary grows while penning a digital contract.

However, as it comes, before any definitions are added, Lexon 0.3 understands 91 keywords plus variations, of which roughly half are processed in an interesting way. Many have only one specific function as marker, like `CLAUSE`, or make the list solely by virtue of being part of a fix multi-word term, with no independent function, like `OFF` in *SIGN OFF*.

To put the word count into perspective, a modern 3rd generation programming language like Rust has about 50 reserved keywords, which are mostly used in a rigid, less interesting way. Beginner's English is said to consist of about 300 words, the Basic English world language project<sup>17</sup> has about 850. These latter counts include many nouns; Lexon's vocabulary contains almost no nouns because these are as a rule defined by the writer of a Lexon text and the Lexon compiler understands them from their function as implied by the rest of the text. This is a fundamental aspect of Lexon's approach and the reason why the Lexon compiler needs to 'understand' relatively few words out of the box. The nouns and phrases that a user adds are inevitably what gives a text depth. The way that definitions work, the compiler learns the role of the new words on the go and recognizes them in the text from that point on. Clause names, however, can be the most interesting because they are the way to insert any complexity of grammar, which can make Lexon texts look rich and elegant. This is a powerful design choice that serves to include language constructs outside of the limit up to which the controlled grammar has to be observed – offering instead of a hard stop, freedom to be fully creative within the safety of a well defined reference frame. Lexon 'understands' such phrases (the clause names) en block, obviously, because the clause itself defines their meaning. The (multi-word) name of a clause's internal grammatical composition is not analysed. The way that the processing of clause names is isolated from the rest of the text is making it possible to mix these monolithic elements with the bulk of the text that the Lexon compiler processes word-for-word, i.e., 'more truly understands'. This combination is a pragmatic way to empower the users to add complexity without having to think about any controlled grammar rules, but importantly also: to freely add vocabulary. The new words are baked into the specific grammatical way that they are presented (in the clause name). They need not be explained further. This mode of extension makes perfect sense for Lexon. 3rd generation programming languages generally allow to add

<sup>17</sup> Basic English – [https://simple.wikipedia.org/wiki/Basic\\_English](https://simple.wikipedia.org/wiki/Basic_English).

variable names and add types. But they can never reach beyond their fix grammar. Lexon, in contrast, accepts any grammatical extension through the freedom it offers in naming clauses. In this, there is not even a requirement to avoid the initial 91 words that Lexon knows when creating compound new compositions.

But the list of initial words cannot be redefined. It is therefore on the one hand desirable that it includes few nouns. The reason that Lexon, on the other hand, has a larger basic vocabulary than, e.g., Rust is that Lexon's grammar is designed to enable multiple ways of expressing the same meaning, to make writing more intuitive, and to allow for more fluid, natural-appearing texts. This does not mean that Lexon's grammar is ambiguous, i.e., that the same sentence could have multiple meanings. It only means that the same meaning can be expressed by differently worded sentences. This is normal for any programming language, and only some developer communities have the ambition to adhere to a style at all times that defines the one, 'right' way for anything that could be expressed. This is always only a convention though.

## WORD LIST

The entries are based on the grammar version 0.2.20 / subset 0.3.8 alpha 79 - English / Reyes.

For an interactive version of this word list, visit <http://lexon.org/vocabulary>.

A, AN.....	24	FOR .....	30	PERSON .....	35
AFTER.....	24	FROM.....	30	PREAMBLE .....	35
AFTERWARDS.....	24	GENERAL.....	30	PROVIDED.....	35
ALL .....	24	GIVEN.....	30	REGISTER, REGISTERS	
ALSO.....	24	GRANT, GRANTS .....	30	.....	35
AMOUNT .....	24	HAS .....	30	REMAINDER.....	35
AND.....	24	HERSELF, HIMSELF ...	30	RESPECTIVE.....	35
ANNOUNCED.....	25	HOUR, HOURS.....	30	RETURN, RETURNS ..	35
APPOINT, APPOINTS.	25	IF .....	31	SECOND, SECONDS....	36
AS.....	25	IN.....	31	SEND, SENDS .....	36
AT .....	25	INTO .....	31	SIGNED .....	36
AUTHOR, AUTHORS...	25	IS .....	31	SO.....	36
BE .....	25	ITSELF .....	31	TERMINATE,	
BEEN .....	26	LEAST .....	31	TERMINATES.....	36
BEING .....	26	LEX .....	32	TERMS .....	36
BINARY.....	26	LEXON .....	32	TEXT .....	37
CERTIFIED .....	26	LIES.....	32	THAT .....	37
CERTIFIES, CERTIFY..	26	MAY .....	32	THE.....	37
CLAUSE.....	26	MILLISECOND,		THEMSELF,	
COMMENT, COMMENTS		MILLISECONDS .....	32	THEMSELVES.....	37
.....	27	MINUTE, MINUTES ....	32	THEN.....	37
CONTRACT,		MONTH, MONTHS .....	32	THERE .....	37
CONTRACTS .....	27	MYSELF.....	33	THEREFOR,	
CURRENT.....	27	NO .....	33	THEREFORE.....	37
DATA .....	27	NOT.....	33	THESE .....	38
DAY, DAYS .....	28	NOW.....	33	THIS.....	38
DECLARE, DECLARES	28	OF.....	33	TIME.....	38
DECLARED.....	28	OFF.....	33	TO .....	38
DEFINED.....	28	ON .....	33	TRUE .....	38
EQUAL.....	28	ONESELF .....	33	WAS .....	38
EQUALING.....	28	OR .....	33	WEEK, WEEKS .....	38
ESCROW .....	29	OURSELVES .....	34	WITH.....	39
FILED .....	29	PASSED.....	34	YEAR, YEARS .....	39
FILE, FILES.....	29	PAST .....	34	YES .....	39
FIX, FIXES .....	29	PAY, PAYS .....	34	YOURSELF,	
FIXED.....	30	PER .....	34	YOURSELVES .....	39

## A, AN

*indefinite article*

no op

Articles can be left out with no change in meaning. They are optional to increase readability.

They can be omitted, because the name they precede must always be unambiguous on its own. This is familiar practice with paper contracts.

Same goes for the, this, these.

The Payer pays an Amount into escrow, appoints the Payee, appoints the Arbiter, and fixes the Fee.

escrow.lex

## AFTER

*timewise preposition*

time operator

After is used to calculate a point in time, relative to a given one.

"Termination Period" is defined as 365 days after the Termination Statement Time.

statement.lex

For more on how to use time, see hours and days.

## AFTERWARDS

*adverb*

causal concatenation

Keyword that introduces temporal order, which is not a default in Lexon.

Separate sentences are performed independently of each other, declaratively, rather than one after the other. Afterwards serves to bind statements into one sentence and to establish that the phrase following it is performed only after all side effects of the phrase before it have been established.

To illustrate by example, in the Lexon sentence given below, the remainder is what remains after the Fee mentioned before afterwards has been deducted.

Cf. THEREFORE.

The Arbiter may pay from escrow the Fee to themselves, and afterwards return the remainder of the escrow to the Payer.

escrow.lex

## ALL

*adjective*

quantifier

Only with CONTRACTS.

All contracts means all digital contracts in a contract system. This includes the main contract as well as the covenants, or subcontracts.

## ALSO

*adverb*

no op

Only appears with AND.

And also is synonymous to AND.

## AMOUNT

*noun*

type

Defines that a name stands for an amount. In the example, Digital Asset Collateral is marked as being used as the handle for a specific number in the document.

"Digital Asset Collateral" is an amount.

statement.lex

Amount can also be used as a name itself, without being first defined. It can only stand for an amount – i.e., for a number and not a text or a time – and Amount must be spelled with a capital 'A' in this case.

The Payer pays an Amount into escrow, appoints the Payee, appoints the Arbiter, and fixes the Fee.

escrow.lex

## AND

*conjunction*

logical and procedural operator

Concatenates actions ...



The Payer pays an Amount into escrow, appoints the Payee, appoints the Arbiter, and fixes the Fee.

escrow.lex

... as well as logical expressions.

A phrase that contains and is true if the part left and the part right of the and are true. There can also be multiple parts, each separated by and. All of them need to be true for the entire expression to be true.

"Factually Breached" is defined as: this License is Commissioned and the Comment Text is not fixed, or this License is Published and there is no Permission to Comment and the Notice Time lies at least 24 hours in the past.

evaluation.lex

For precedence and the interplay between and and or, see or.

## ANNOUNCED

*adjective*

truth value

Functions like FIXED.

## APPOINT, APPOINTS

*verb*

parameter assignment operator

Expresses that the subject of the sentence will determine what the specified object's names will mean concretely. In the example, who the Payee and the Arbiter are.

Functions like fix, see additional notes regarding the subject there.

The Payer pays an Amount into escrow, appoints the Payee, appoints the Arbiter, and fixes the Fee.

escrow.lex

Functional synonym to certify, declare, file, fix, grant, and register.

## AS

*conjunction*

value assignment operator part

Assigns the value of the expression to its right to the name on its left.

The Secured Party may file a Termination Statement, and certify the Termination Statement Time as the then current time.

statement.lex

As can also make a name true that was defined as a binary. In this example, License serves as an object that means the entire contract system, which ultimately is a redundant scope. The relevant mutation is that Commissioned becomes a fact, i.e., true.

The Licensor may certify this License as Commissioned.

evaluation.lex

## AT

*preposition*

quantifier

Only in conjunction with LEAST.

"Factually Breached" is defined as: this License is Commissioned and the Comment Text is not fixed, or this License is Published and there is no Permission to Comment and the Notice Time lies at least 24 hours in the past.

evaluation.lex

## AUTHOR, AUTHORS

*noun*

keyword

The information after the AUTHOR(s) keyword is expected to be the name(s) of the creator(s) of the Lexon text. They are meta data, not parsed, and not used in the document itself.

As a convention, author and authors are usually spelled in uppercase.

**AUTHORS: FLORIAN IDELBERGER,  
HENNING DIEDRICH**

evaluation.lex

## BE

*verb*

assignment

Used with a meaning like shall.

Functions like IS and can be used synonymously. The linguistic difference is irrelevant for the machine.

"Noticed" be defined as a Notice Time being fixed.

## BEEN

*verb*

comparison operator

Appears only in conjunction with HAS.

Has been functions like BEING.

The Arbitrator may, if the Notice Time has been fixed, return the Fee to the Seller.

## BEING

*present participle*

comparison operator

Compares the expression to its left with the expression on its right and results in everything together being TRUE or FALSE.

In the example, being tests Notice Time for whether it had been fixed before. Noticed will be true exactly when Notice Time is known, and false if, no value has been given for Notice Time before at any point during the lifetime of the contract.

"Noticed" is defined as a Notice Time being fixed.

evaluation.lex

## BINARY

*adjective*

type

Defines a name as standing for a binary value, e.g., YES or NO, or TRUE or FALSE.

Note that an undefined binary name is considered to have the value FALSE. Declaring a name sets it to TRUE. Likewise, testing whether a binary name is declared, checks whether it is TRUE.

"Default" is a binary.

statement.lex

## CERTIFIED

*adjective*

«defined»

Expresses that a name has a value assigned, i.e., is not unbound or undefined.

In the example, being tests Notice Time for whether it had been certified before. Noticed will be true exactly when a Notice Time is known, and false if no value has been given for Notice Time before, at any point during the lifetime of the contract.

Functions like FIXED.

"Noticed" is defined as a Notice Time being certified.

## CERTIFIES, CERTIFY

*verb*

assignment operator

Expresses that the subject of the sentence will determine what the specified object's names will mean concretely. In the example, who the Payee and the Arbitrator are.

Functions like fix, see additional notes regarding the subject and invocation there.

The Filing Office may certify the File Number.

statement.lex

Functional synonym to appoint, declare, file, fix, grant, and register.

## CLAUSE

*noun*

function keyword

Signals the start of a clause. A colon must follow, and the name of the clause. Then, the statements that constitute the clause.

Almost every digital contract has one or more clauses. Only in rare, simplistic cases does a contract have only a recital.

Either a clause's name is used to instigate actions that change the state of the contract:

**CLAUSE: Pay Back.**  
**The Arbiter may pay from escrow the Fee to themselves, and afterwards return the remainder of the escrow to the Payer.**

escrow.lex

Or, a clause name can itself be a value, if the clause uses defined.

The concrete meaning of the name of such clause is dynamic. That is, the concrete meaning of the clause name is not assigned once and for all at any point in time. Instead, whenever the clause name is used elsewhere in any context, the expression right-hand of defined is re-evaluated for its now current result, which is then the meaning, or value, of that clause name.

**Clause: Termination Period.**  
**"Termination Period" is defined as 365 days after the Termination Statement Time.**

statement.lex

The clause name can be used as an expression in the context of other clauses, i.e the name can be used like a value.

The example below uses the name Termination Period that is defined in the example above.

**The Filing Office may, if the Termination Period has passed, terminate this contract.**

statement.lex

## COMMENT, COMMENTS

*noun*

comment keyword

Start of comments that are not translated by the compiler.

Functions like PREAMBLE but can be used multiple times in different places.

Lexon is self-documenting, which greatly diminishes the role of comments. They should be used sparingly or not at all. They can help to explain a more convoluted set of conditions, as can be found in contracts that need to spell out things in detail, including all relevant fringe cases.

Care should be taken to clarify that a comment is not part of the legally binding text; but is written to provide motivation or explain complex aspects with a broad brush, to make the contract easier to understand for a human

reader. Such clarification may be added as part of the comment itself.

As a convention, COMMENT is usually spelled in uppercase.

**COMMENT: A license can be for any tangible or intangible good.**

Cf. PREAMBLE.

## CONTRACT, CONTRACTS

*noun*

self reference

Contract as well as all contracts stand for the contract (system) itself, including all covenants (subcontracts)

Contract can either be used to define a proper name for the digital contract:

**"Financing Statement" is this contract.**

statement.lex

Or, Contract as well as all contracts can be used as object to terminate.

**The Filing Office may, if the Termination Period has passed, terminate this contract.**

statement.lex

## CURRENT

*adjective*

time value

Only appears with time.

**The Filing Office may fix the Initial Statement Date as the current time.**

statement.lex

## DATA

*noun*

type

Defines a name as standing for a piece of data.

Data can be a text, a number, a hash, a block-chain address, or an id of any type.

"File Number" is data.

statement.lex

## DAY, DAYS

*noun*

*time unit*

Used to describe a duration.

A duration can be used to calculate a point in time relative to another point in time. For example, relative to now or in the past, or – as in the example below - relative to a name that means a specific time.

"Termination Period" is defined as 365 days after the Termination Statement Time.

Cf. hours.

## DECLARE, DECLARES

*verb*

truth assignment operator

Used to state that something has happened, or is true.

Technically, declare assigns the truth value, true, to a name. That name must have been defined (see is) as a binary.

In the example this means that Default is now true. Note that before that, it was false.

Cf. binary.

The Secured Party may declare Default.

statement.lex

## DECLARED

*adjective*

«true»

Synonym to true.

In the example, the fact that Default has been declared is the same as saying that it is true that Default happened.

The Filing Office may, if Default is declared, pay the Digital Asset Collateral to the Secured Party.

statement.lex

## DEFINED

*adjective*

assignment operator

Always used with IS, or BE and AS, to describe the meaning of the name to its left by means of the expression on its right.

The meaning is dynamic. That is, the concrete meaning is not assigned once and for all at any point in time. But instead, whenever the name that is being defined is used elsewhere in any context, the expression right-hand of defined is re-evaluated for its now current result, which is in that moment the meaning of that name.

**CLAUSE: Termination Period**

"Termination Period" is defined as 365 days after the Termination Statement Time.

statement.lex

This type of sentence is the essence of a particular type of CLAUSE whose name can be used like an expression, i.e., the name of such clause can be used like a value in the text of another clause.

The example below uses the name Termination Period that is defined in the example above.

The Filing Office may, if the Termination Period has passed, terminate this contract.

statement.lex

## EQUAL

*comparison operator*

equivalence of values

Forms an expression that is true if the values left and right of equal are the same.

Near synonym of equaling.

"Parity" is defined as the Count of X being equal to the Count of Y.

## EQUALING

*comparison operator*

equivalence of values

Forms an expression that is true if the values left and right of equaling are the same.

Near synonym of equal.

"Parity" is defined as the Left Side equaling the Right Side.

## ESCROW

*noun*

system variable

The internal token escrow of a digital contract.

It is mostly used as object to the predicate pay.

Using it with remainder results into a number: the amount of tokens left in the escrow at that point in time.

The Arbiter may pay from escrow the Fee to themselves, and afterwards pay the remainder of the escrow to the Payee.

escrow.lex

## FILED

*adjective*

«defined»

Asking whether a name is filed constitutes an expression that is true in case the name had a value assigned to it previously. The expression is false if the name had not been defined before during the lifetime of the contract.

... the Continuation Statement is filed ...

statement.lex

To clarify, it does not matter if there is text somewhere in the contract that gives a name a concrete meaning. What matters is whether for a specific, live contract between concrete parties and with a concrete state, it so happened that it is clear what a specific name stands for, or, that what the name stands for exists.

If you take this example ...

The Filing Office may, if the Continuation Statement is filed, fix the Continuation Statement Date.

statement.lex

... the phrase the Continuation Statement is filed is true, if what is described in the clause shown below ever happened. Concretely, if the Secured Party has filed the Continuation Statement.

Clause: File Continuation.  
The Secured Party may file the Continuation Statement.

statement.lex

Note that in this example contract, the Continuation Statement is defined as a binary. That means that it does not have any specific content beyond existing or not. The filing of it 'is' the statement that continuation is desired.

## FILE, FILES

*verb*

parameter assignment operator

Synonym to fix.

The Secured Party may file the Continuation Statement.

statement.lex

## FIX, FIXES

*verb*

parameter assignment operator

Indicates that the subject of the sentence will be who determines the meaning of the named objects.

Note that this cuts both ways. The subject might itself be determined by the act of fixing the objects: if it had not been settled yet who the name of the subject refers to, then whoever performs the fixing is from that point on named like the subject of this sentence. The name sticks for the remaining lifetime of the contract. Accordingly, in the example below, if the role of the Filer had not been determined, the person who is fixing the Filing Office etc. becomes the Filer. The way to prevent this automatism is to use may.

The values that are assigned to the objects of the sentence are given by the subject when that person acts to invoke this clause.

The Filer fixes the Filing Office, fixes the Debtor, fixes the Secured Party, and fixes the Collateral.

statement.lex

Functional synonym to appoint, certify, declare, file, grant, and register.

## FIXED

*adjective*

«defined»

Expresses that a name has a value assigned, i.e., is not unbound or undefined. In the example, being tests Notice Time for whether it had been fixed before. Noticed will be true exactly when a Notice Time is known, and false if no value has been given for Notice Time before, at any point during the lifetime of the contract.

Functions like CERTIFIED.

"Noticed" is defined as a Notice Time being fixed.

evaluation.lex

## FOR

*preposition*

no op

Only in conjunction with FILE or FILED.

The terms FILE FOR or FILED FOR function like FILE or FILED without FOR.

## FROM

*preposition*

transfer origin marker

Only in conjunction with ESCROW.

The Arbiter may pay from escrow the Fee to themselves, and afterwards return the remainder of the escrow to the Payer.

escrow.lex

## GENERAL

*adjective*

no op

Optional specification to TERMS.

## GIVEN

*preposition*

conditional keyword

Following statements are executed only if the immediately following condition is true.

Appears only together with THAT.

GIVEN THAT is a synonym to IF.

The Filing Office may, given that the Continuation Window Start has passed, send the Notification Statement to the Secured Party.

## GRANT, GRANTS

*verb*

truth assignment operator

Synonym to fix.

The Licensee may grant the Permission to Comment.

evaluation.lex

## HAS

*auxiliary verb*

part

Only in conjunction with BEEN or PASSED.

The Filing Office may, if the Continuation Window Start has passed, send the Notification Statement to the Secured Party.

statement.lex

## HERSELF, HIMSELF

*reflexive pronoun*

automatic reference

Refers to the subject of the sentence.

Functions like THEMSELF.

## HOUR, HOURS

*time unit*

time constant of 3600 seconds

Used to describe a duration.

A duration can be used to calculate a point in time relative to another point in time. For example, relative to now or in the past.

"Factually Breached" is defined as: this License is Commissioned and the Comment Text is not fixed, or this License is Published and there is no Permission to Comment and the Notice Time lies at least 24 hours in the past.

evaluation.lex

Cf. days.

## IF

*conjunction*

assertion

Following statements are executed only if the immediately following condition is true.

The condition starts after if and ends with a comma, which can be followed by an optional then.

The statements follow after that.

In this example ...

The Filing Office may, if the Continuation Window Start has passed, send the Notification Statement to the Secured Party.

statement.lex

... the condition is:

the Continuation Window Start has passed

... the statements are:

send the Notification Statement to the Secured Party

## IN

*preposition*

no op

Only in conjunction with PAST.

"Factually Breached" is defined as: this License is Commissioned and the Comment Text is not fixed, or this License is Published and there is no Permission to Comment and the Notice Time lies at least 24 hours in the past.

evaluation.lex

## INTO

*preposition*

operator part

Used with NOTIFY, SEND and PAY.

The Secured Party may pay a Reminder Fee into escrow.

statement.lex

## IS

*verb*

assignment and equality operator

Can be used to define of what category a name is; to assign a value to a name; to compare a name to a value; or to check that something is the case.

"Payer" is a person.

escrow.lex

The Filing Office may, if Default is declared, pay the Digital Asset Collateral to the Secured Party.

statement.lex

Note that in the following example, License means the (sub)contract itself and is checks the state of the License, diverting into the clause Factually Breached to find out if the License is breached.

The Arbiter may, if this License is Factually Breached:

evaluation.lex

## ITSELF

*reflexive pronoun*

automatic reference

Refers to the subject of the sentence.

Functions like THEMSELF.

## LEAST

*adjective*

time operator part

Only in conjunction with AT.

"Factually Breached" is defined as: this License is Commissioned and the Comment Text is not fixed, or this License is Published and there is no Permission to Comment and the Notice Time lies at least 24 hours in the past.

evaluation.lex

## LEX

*noun*

keyword

Keyword for the start of a digital contract.

LEX must be the first word of a digital contract. The words after LEX are the name of the entire digital contract (system) described thereafter.

As a convention, LEX is usually spelled in uppercase.

**LEX Escrow.**

escrow.lex

## LEXON

*noun*

keyword

The numbers following LEXON are the version number of the Lexon compiler that the digital contract was written for. This is a concept that helps while Lexon is evolving. As a rule, newer compilers can compile older version Lexon texts but there will sometimes be 'breaking changes' where this backward compatibility is not provided and older texts have to be adapted to the changes of a new grammar.

Note that compatibility is not a dimension of what the texts mean in human language, which remains the same throughout Lexon versions, because English does not change. Instead, this is about older versions of the compiler understanding less than newer ones, i.e., the grammar getting less restricted.

As a convention, LEXON is usually spelled in uppercase.

**LEXON: 0.2.12**

statement.lex

## LIES

*verb*

time comparison operator

Only in conjunction with AT LEAST.

"Factually Breached" is defined as: this License is Commissioned and the Comment Text is not fixed, or this License is Published and there is no Permission to Comment and the Notice Time lies at least 24 hours in the past.

evaluation.lex

## MAY

*auxilliary verb*

permission marker

The subject to MAY is/are the only party or parties to the contract that are authorized to initiate the action described. The subject must be bound, i.e., the name before MAY must have been defined before, it cannot be defined in the may statement. Note that statements without may might likewise restrict authority to the named subject. And it is possible that the subject is unbound in cases without MAY, i.e., the role not defined at that point.

**The Filing Office may certify the File Number.**

statement.lex

## MILLISECOND, MILLISECONDS

*noun*

time constant of 1/1000 seconds.

Functions like DAYS.

## MINUTE, MINUTES

*noun*

time constant of 60 seconds

Functions like DAYS.

## MONTH, MONTHS

*noun*

Time constant of 2592000 seconds, i.e., 30 DAYS

Functions like DAYS.



## MYSELF

*reflexive pronoun*  
automatic reference

Refers to the subject of the sentence.

Functions like THEMSELF.

## NO

*adverb*  
logic operator

Logical inversion. In conjunction with there is, also used to test whether a name has been assign any concrete meaning yet. See fixed.

"Factually Breached" is defined as: this License is Commissioned and the Comment Text is not fixed, or this License is Published and there is no Permission to Comment and the Notice Time lies at least 24 hours in the past.

evaluation.lex

## NOT

*adverb*  
logic operator

Logical inversion.

Used to form the opposite of a logical expression. Can be positioned before a name, or before fixed. The resulting expression means the opposite of what the part after not meant. It can be part of a bigger logical expression, as shown below. Not, as is grammatically correct, binds the next noun or verb only. The requirements for sentence structure make sure that no ambiguity can arise for the human reader.

If a more complex expression must be inverted, it has to be written as a clause. This simple device to avoid ambiguity without requiring literal bracketing is borrowed from proven best practice in Functional Programming.

"Factually Breached" is defined as: this License is Commissioned and the Comment Text is not fixed, or this License is Published and there is no Permission to Comment and the Notice Time lies at least 24 hours in the past.

evaluation.lex

## NOW

*noun*  
time value

Synonym to CURRENT TIME.

The Filing Office may fix the Initial Statement Date as now.

## OF

*preposition*  
no op

Only with REMAINDER.

The Arbiter may pay from escrow the Fee to themselves, and afterwards pay the remainder of the escrow to the Payee.

escrow.lex

## OFF

*adverb*  
operator part

Only with SIGNED.

## ON

*preposition*  
operator part

Only with SIGNED.

## ONESELF

*reflexive pronoun*  
automatic reference

Refers to the subject of the sentence.

Functions like THEMSELF.

## OR

*conjunction*  
logic operator

Used to build logical expressions. A phrase that contains or is true if the part left or the part right of the or are true.

Colons, commas and semicolons are relevant to separate sub-phrases. Programmers note that

there is no precedence of and over or in Lexon as this is not a part of natural language. Commas and semicolons offer two levels of nesting. Beyond this, precedence is created by encapsulating logical expressions into separate clauses.

In the example ...

**"Factually Breached" is defined as: this License is Commissioned and the Comment Text is not fixed, or this License is Published and there is no Permission to Comment and the Notice Time lies at least 24 hours in the past.**

evaluation.lex

... all of the following counts as the left-side of the or, because there is a comma before the or and no comma to the left of it:

**this License is Commissioned and the Comment Text is not fixed**

... and all of the following is the right side of the or, because there is a comma before the or and no comma to the right of it:

**this License is Published and there is no Permission to Comment and the Notice Time lies at least 24 hours in the past.**

## OURSELVES

*reflexive pronoun*

automatic reference

Refers to the subject of the sentence.

Functions like THEMSELF.

## PASSED

*adjective*

time comparison operator

Compares a point in time to the current time.

**The Filing Office may, if the Continuation Window Start has passed, send the Notification Statement to the Secured Party.**

statement.lex

For more on how to use time, see hours and days.

## PAST

*noun or adjective*

negative time sign

Past indicates that a measure of time is to be subtracted from the current time, or it functions like HAS PASSED.

In the example, in the past functions as a negative sign to the literal 24 hours, relative to now.

**"Factually Breached" is defined as: this License is Commissioned and the Comment Text is not fixed, or this License is Published and there is no Permission to Comment and the Notice Time lies at least 24 hours in the past.**

evaluation.lex

For more on how to use time, see hours and days.

## PAY, PAYS

*verb*

transfer operator

A transfer over the amount given immediately following pay, from the subject of the sentence, to the object marked with to or into.

**The Payer pays an Amount into escrow, appoints the Payee, appoints the Arbiter, and fixes the Fee.**

escrow.lex

## PER

*preposition*

keyword

PER marks the beginning of the TERMS of a covenant or sub-contract.

Digital contracts are often really contract systems that control the creation of individual contracts each with different counter parties. These sub contracts are called covenants in the context of digital contracts. Their terms are separated from the general terms of the digital contracts – which govern everything else, specifically how the covenants come into existence – by the keyword PER, followed by the name the of the covenant, and optionally preceded by the keyword TERMS.

As a convention, PER is usually spelled in uppercase.

**TERMS PER License:**

statement.lex

## PERSON

*noun*

type

Defines a name to stand for a person.

**"Payer" is a person.**

escrow.lex

## PREAMBLE

*noun*

keyword

Start of a comment from both legal and processing point of view. Words after PREAMBLE are explanations with minimal legal weight and are not translated to 3rd generation language code by the compiler. Accordingly, in the example below, no word behind the colon is interpreted. This is not a special case: it is similar to how Lexon does not account for the common meaning of nouns in human language that a writer defines. This meaning is helpful to understand the contract, but not part of it, like the preamble text. Likewise, it is a common pitfall to read the preamble in a paper contract as part of the legal agreement; it is not. Its value lies in paraphrasing the more technical prose of the agreement in more accessible but blurrier terms and to provide context.

As a convention, PREAMBLE is usually spelled in uppercase.

**PREAMBLE:** This is a licensing contract for a software evaluation.

evaluation.lex

## PROVIDED

*adjective*

conditional keyword

Synonym to IF.

**The Filing Office may, provided the Continuation Window Start has passed, send the Notification Statement to the Secured Party.**

## REGISTER, REGISTERS

*verb*

parameter assignment operator

Synonym to fix.

**The Licensee may register a Comment Text.**

evaluation.lex

## REMAINDER

*noun*

no op

Optional part to internal token count variable ESCROW.

**The Arbiter may pay from escrow the Fee to themselves, and afterwards pay the remainder of the escrow to the Payee.**

escrow.lex

## RESPECTIVE

*adjective*

optional part of built-in time value

Only in conjunction with CURRENT TIME.

**The Secured Party may certify the Termination Statement Time as the respective current time.**

## RETURN, RETURNS

*verb*

transfer operator

Synonym to PAY.

**The Arbiter may pay from escrow the Fee to themselves, and afterwards return the remainder of the escrow to the Payer.**

escrow.lex

## SECOND, SECONDS

*noun*

time constant of 1 second

Functions like DAYS.

## SEND, SENDS

*verb*

transfer and messaging operator

Send a message. On the blockchain, this can be an entry on the receipt log.

The Filing Office may, if the Continuation Window Start has passed, send the Notification Statement to the Secured Party.

statement.lex

## SIGNED

*adjective*

logical true

Only in the combination SIGNED OFF, with optional following ON.

In other words, signed, signed off, and signed off on all mean the same.

The Agent may, once the Receipt is signed off, return the Collateral to the Counterparty.

## SO

*adjective*

causal concatenator part

Only in conjunction with IF.

IF SO is a synonym for AFTERWARDS. See remarks on sentence order there.

The Arbiter may pay from escrow the Fee to themselves, and if so return the remainder of the escrow to the Payer.

## TERMINATE, TERMINATES

*verb*

destruction operator

The consequence of termination is that a contract's state cannot be changed anymore. Both main contracts and covenants (subcontracts) can be terminated. It is good practice to end a contract after its purpose is fulfilled so that it cannot be partially restarted for unintended consequences.

The Filing Office may, if the Termination Period has passed, terminate this contract.

statement.lex

## TERMS

*noun*

optional keyword

Optional marker of the beginning of general or per-subcontract terms. The TERMS keyword serves to increase clarity but can be left out as the document order suffices for the compiler to understand what part of a document to expect next: terms are necessarily all statements following the LEX keyword and digital contract (system) name. For yet more clarity, TERMS can be preceded by the optional keyword GENERAL.

For covenants (sub contracts), their terms must be marked at least by the keyword PER, followed by the covenant's name. TERMS may precede PER but is optional.

As a convention, TERMS is usually spelled in uppercase.

TERMS:

evaluation.lex

TERMS PER License:

statement.lex

## TEXT

*noun*

type

Defines that a name is standing for a text.

"Notification Statement" is a text.

statement.lex

## THAT

*conjunction*

conditional keyword

Following statements are executed only if the immediately following condition is true.

Appears only together with GIVEN.

GIVEN THAT is a synonym to IF.

The Filing Office may, given that the Continuation Window Start has passed, send the Notification Statement to the Secured Party.

## THE

*article*

no op

Articles are optional to increase readability, because the name they precede must always be unambiguous on its own.

The Payer pays an Amount into escrow, appoints the Payee, appoints the Arbiter, and fixes the Fee.

escrow.lex

Cf. A.

## THEMSELF, THEMSELVES

*reflexive pronoun*

automatic reference

Refers to the subject of the sentence.

In this example, themselves means the Arbiter.

The Arbiter may pay from escrow the Fee to themselves, and afterwards pay the remainder of the escrow to the Payee.

escrow.lex

## THEN

*adverb, adjective*

conditional keyword, causal concatenator

If the Termination Period has passed, then terminate this contract.

In conjunction with CURRENT TIME:

The Secured Party may file a Termination Statement, and certify the Termination Statement Time as the then current time.

statement.lex

Also functions like THEREFOR:

This License is then Paid.

## THERE

*adverb*

existence test

Used to reason about the existence of something, or more precisely, about whether a name has a defined meaning or not.

Appears only in THERE IS or THERE IS NOT, or with variations of IS, like BE.

Cf. fixed.

"Factually Breached" is defined as: this License is Commissioned and the Comment Text is not fixed, or this License is Published and there is no Permission to Comment and the Notice Time lies at least 24 hours in the past.

evaluation.lex

## THEREFOR, THEREFORE

*adverb*

causal concatenator

Therefore binds a sentence to the preceding ones, so that it is performed only if all preceding sentences were performed, i.e., did not disqualify for access or conditional reasons.

Without THEREFORE, a sentence by itself is always materialized when a clause is triggered.

Cf. afterwards.

The Licensee pays the Licensing Fee to the Licensor, and pays the Breach Fee into escrow. This License is therefore Paid.

evaluation.lex

## THESE

*adjective*

no op

These can be required to get the natural language grammar right but does not change meaning by its presence or absence because the name that it precedes must always be unambiguous by itself.

The Licensor may certify these Agreements as Commissioned.

## THIS

*adjective*

no op

THIS can be required to get the natural language grammar right but does not change meaning by its presence or absence because the name that it precedes must always be unambiguous by itself.

The Licensor may certify this License as Commissioned.

evaluation.lex

## TIME

*noun*

type

Either defines a name as standing for a time value.

"Initial Statement Date" is a time.

statement.lex

Or, specifies the current point in time.

The Filing Office may fix the Initial Statement Date as the current time.

statement.lex

## TO

*preposition*

transfer operator part

Appears in conjunction with PAY, SEND, be or equal.

The Arbiter may pay from escrow the Fee to themselves, and afterwards pay the remainder of the escrow to the Payee.

escrow.lex

The Filing Office may, if the Continuation Window Start has passed, send the Notification Statement to the Secured Party.

statement.lex

## TRUE

*adjective*

logical true

A value that a name or an expression can have, meaning that something is the case.

Synonymous with YES.

In the following example, the expression the Continuation Window has passed can be TRUE or FALSE.

The Filing Office may, if the Continuation Window Start has passed, send the Notification Statement to the Secured Party.

statement.lex

Cf. FIXED.

## WAS

*verb*

logic equivalence operator

Functions like IS.

## WEEK, WEEKS

*noun*

time factor constant

Time constant of 604,800 seconds, i.e., 7 DAYS.

Functions similar to DAYS.

## **WITH**

*conjunction*

causal concatenator

Only appears as AND WITH THIS.

Functions like THEREFORE.

## **YEAR, YEARS**

*noun*

time factor constant

Time constant of 31,536,000 seconds, i.e., 365 DAYS.

Functions similar to DAYS.

## **YES**

*noun*

logical true

A value that a name or an expression can have, meaning that something is the case.

Synonymous with TRUE.

## **YOURSELF, YOURSELVES**

*reflexive pronoun*

automatic reference

Refers to the subject of the sentence.

Functions like THEMSELF.

## EXAMPLES

**Lexon for Law**

Lexon allows for law to be executed as a program. Asst. prof. Carla L. Reyes of SMU pioneers the use of Lexon to write statute – shown below – in her seminal 2021 paper *Creating Cryptolaw for the Uniform Commercial Code*.<sup>18</sup> She created the following Lexon code as a proposal to the commission that is tasked with the reform of the U.S. trade law, which she advises on blockchain topics. This code could become model law, be adapted by states to be executed on the computers of their local agencies and protect billions of dollars of collateral.

The salient point is that the law itself, without further changes *is* the program that the respective office runs to implement the law. The productivity gains of Lexon could not be illustrated better.

The motivation for this proposal is a concrete shortfall of the existing statute. Asst. prof. Reyes writes (emphasis added):

*“Under certain conditions, security interests not only bind the creditor and debtor, but also third-party creditors seeking to lend against the same collateral. To receive this extraordinary benefit, creditors must put the world on notice, usually by filing a financing statement with the state in which the debtor is located. Unfortunately, the Uniform Commercial Code (U.C.C.) Article 9 filing system fails to provide actual notice to interested parties and introduces risk of heavy financial losses. To solve this problem, this Article introduces a smart-contract-based U.C.C.-1 form built using Lexon, an innovative new programming language that enables the development of smart contracts in English. The proposed “Lexon U.C.C. Financing Statement” does much more than merely replicate the financing statement in digital form; it also performs several U.C.C. rules so that, for the first time, the filing system works as intended. In demonstrating that such a system remains compatible with existing law, the Lexon U.C.C. Financing Statement also reveals important lessons about the interaction of technology and commercial law.”*<sup>ibid. 18</sup>

**LEX UCC Financing Statement.****LEXON: 0.2.12**

"Financing Statement" is this contract.  
 "File Number" is data.  
 "Initial Statement Date" is a time.  
 "Filer" is a person.  
 "Debtor" is a person.  
 "Secured Party" is a person.  
 "Filing Office" is a person.  
 "Collateral" is data.  
 "Digital Asset Collateral" is an amount.  
 "Reminder Fee" is an amount.  
 "Continuation Window Start" is a time.  
 "Continuation Statement Date" is a time.  
 "Continuation Statement Filing Number" is data.  
 "Lapse Date" is a time.  
 "Default" is a binary.  
 "Continuation Statement" is a binary.  
 "Termination Statement" is a binary.  
 "Termination Statement Time" is a time.  
 "Notification Statement" is a text.

The Filer fixes the Filing Office, fixes the Debtor, fixes the Secured Party, and fixes the Collateral.

<sup>18</sup> *Washington and Lee Law Review* – [https://papers.ssrn.com/sol3/papers.cfm?abstract\\_id=3809901](https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3809901)



**Clause: Certify.**

The Filing Office may certify the File Number.

**Clause: Set File Date.**

The Filing Office may fix the Initial Statement Date as the current time.

**Clause: Set Lapse.**

The Filing Office may fix the Lapse Date.

**Clause: Set Continuation Start.**

The Filing Office may fix the Continuation Window Start.

**Clause: Pay Fee.**

The Secured Party may pay a Reminder Fee into escrow.

**Clause: Notice.**

The Filing Office may fix the Notification Statement.

**Clause: Notify.**

The Filing Office may, if the Continuation Window Start has passed, send the Notification Statement to the Secured Party.

**Clause: Pay Escrow In.**

The Debtor may pay the Digital Asset Collateral into escrow.

**Clause: Fail to Pay.**

The Secured Party may declare Default.

**Clause: Take Possession.**

The Filing Office may, if Default is declared, pay the Digital Asset Collateral to the Secured Party.

**Clause: File Continuation.**

The Secured Party may file the Continuation Statement.

**Clause: Set Continuation Lapse.**

The Filing Office may, if the Continuation Statement is filed, fix the Continuation Statement Date.

**Clause: File Termination.**

The Secured Party may file a Termination Statement, and certify the Termination Statement Time as the then current time.

**Clause: Release Escrow.**

The Filing Office may, if the Termination Statement is filed, return the Digital Asset Collateral to the Debtor.

**Clause: Release Reminder Fee.**

The Filing Office may, if the Termination Statement is filed, return the Reminder Fee to the Secured Party.

**Clause: Termination Period.**

"Termination Period" is defined as 365 days after the Termination Statement Time.

**Clause: Terminate and Clear.**

The Filing Office may, if the Termination Period has passed, terminate this contract.

*Source 3 – Lexon code example: U.C.C. Filing Statement*

*Sophia Output*

The above example is compiled to the following Sophia code, using the --harden option to make the code safe against certain attacks. The produced code makes use specifically of Sophia's highly precise

handling of undefined values, employing the *option type*, a hybrid of a normal atomic type and a value meaning that no value is given, *None*. The Lexon text is again used verbatim for comments, exploiting that Lexon code is per se self-documenting.

```

/* Lexon-generated Sophia code

code:      UCC Financing Statement
file:      statement.lex
code tagged: 0.2.12
compiler:  lexon 0.3 alpha 85
grammar:   0.2.20 / subset 0.3.8 alpha 79 - English / Reyes
backend:   sophia 0.3.1/85
target:    sophia 6+
options:   --sophia --harden
*/

@compiler >=6

include "Option.aes"
using Option

/** LEX UCC Financing Statement.
 *
 * LEXON: 0.2.12
 *
 * "Financing Statement" is this contract.
 * "File Number" is data.
 * "Initial Statement Date" is a time.
 * "Filer" is a person.
 * "Debtor" is a person.
 * "Secured Party" is a person.
 * "Filing Office" is a person.
 * "Collateral" is data.
 * "Digital Asset Collateral" is an amount.
 * "Reminder Fee" is an amount.
 * "Continuation Window Start" is a time.
 * "Continuation Statement Date" is a time.
 * "Continuation Statement Filing Number" is data.
 * "Lapse Date" is a time.
 * "Default" is a binary.
 * "Continuation Statement" is a binary.
 * "Termination Statement" is a binary.
 * "Termination Statement Time" is a time.
 * "Notification Statement" is a text.
 *
 * The Filer fixes the Filing Office, fixes the Debtor, fixes the Secured Party, and fixes the
Collateral.
**/

main contract UCCFinancingStatement =

  record state = {
    file_number : option(string),
    initial_statement_date : option(int),
    filer : option(address),
    debtor : option(address),
    secured_party : option(address),
    filing_office : option(address),
    collateral : option(string),
    digital_asset_collateral : option(int),
    reminder_fee : option(int),
    continuation_window_start : option(int),
    continuation_statement_date : option(int),
    continuation_statement_filing_number : option(string),
    lapse_date : option(int),
    _default : option(bool),
    continuation_statement : option(bool),
    termination_statement : option(bool),
    termination_statement_time : option(int),
    notification_statement : option(string),
    terminated : bool
  }

  datatype event = Message(indexed address, indexed address, string)

```

```

    entrypoint init(filing_office : address, debtor : address, secured_party : address,
collateral : string) = {
    file_number = None,
    initial_statement_date = None,
    filer = Some(Call.caller),
    debtor = Some(debtor),
    secured_party = Some(secured_party),
    filing_office = Some(filing_office),
    collateral = Some(collateral),
    digital_asset_collateral = None,
    reminder_fee = None,
    continuation_window_start = None,
    continuation_statement_date = None,
    continuation_statement_filing_number = None,
    lapse_date = None,
    _default = None,
    continuation_statement = None,
    termination_statement = None,
    termination_statement_time = None,
    notification_statement = None,
    terminated = false
}

stateful function termination() =
    put(state{terminated = true})

function check_termination() =
    require(!state.terminated, "contract system terminated before")

stateful function transfer(to : address, amount : int) =
    Chain.spend(to, amount)

function send(to : address, message : string) =
    Chain.event(Message(Call.caller, to, message))

function permit(authorized : option(address)) =
    require(Call.caller == force(authorized), "access")

/*
 * Clause: Certify.
 * The Filing Office may certify the File Number.
 */

stateful entrypoint certify(file_number : string) =
    check_termination()
    permit(state.filing_office)
    put(state{file_number = Some(file_number)})

/*
 * Clause: Set File Date.
 * The Filing Office may fix the Initial Statement Date as the current time.
 */

stateful entrypoint set_file_date() =
    check_termination()
    permit(state.filing_office)
    put(state{initial_statement_date = Some(Chain.timestamp)})

/*
 * Clause: Set Lapse.
 * The Filing Office may fix the Lapse Date.
 */

stateful entrypoint set_lapse(lapse_date : int) =
    check_termination()
    permit(state.filing_office)
    put(state{lapse_date = Some(lapse_date)})

/*
 * Clause: Set Continuation Start.
 * The Filing Office may fix the Continuation Window Start.
 */

stateful entrypoint set_continuation_start(continuation_window_start : int) =
    check_termination()
    permit(state.filing_office)
    put(state{continuation_window_start = Some(continuation_window_start)})

/*
 * Clause: Pay Fee.
 * The Secured Party may pay a Reminder Fee into escrow.
 */

```

```

stateful payable entrypoint pay_fee() =
  check_termination()
  permit(state.secured_party)
  switch(state.reminder_fee)
    None => put(state{reminder_fee = Some(Call.value)})
    Some(_) => put(state{reminder_fee = Some(force(state.reminder_fee) + Call.value)})

/*
 * Clause: Notice.
 * The Filing Office may fix the Notification Statement.
 */

stateful entrypoint notice(notification_statement : string) =
  check_termination()
  permit(state.filing_office)
  put(state{notification_statement = Some(notification_statement)})

/*
 * Clause: Notify.
 * The Filing Office may, if the Continuation Window Start has passed, send the
Notification Statement to the Secured Party.
 */

entrypoint notify() =
  check_termination()
  permit(state.filing_office)
  if(state.continuation_window_start =< Some(Chain.timestamp))
    send(force(state.secured_party), state.notification_statementx)

/*
 * Clause: Pay Escrow In.
 * The Debtor may pay the Digital Asset Collateral into escrow.
 */

stateful payable entrypoint pay_escrow_in() =
  check_termination()
  permit(state.debtor)
  switch(state.digital_asset_collateral)
    None => put(state{digital_asset_collateral = Some(Call.value)})
    Some(_) => put(state{digital_asset_collateral =
Some(force(state.digital_asset_collateral) + Call.value)})

/*
 * Clause: Fail to Pay.
 * The Secured Party may declare Default.
 */

stateful entrypoint fail_to_pay() =
  check_termination()
  permit(state.secured_party)
  put(state{default = true})

/*
 * Clause: Take Possession.
 * The Filing Office may, if Default is declared, pay the Digital Asset Collateral to the
Secured Party.
 */

stateful entrypoint take_possession() =
  check_termination()
  permit(state.filing_office)
  if(state._default != None)
    transfer(force(state.secured_party), state.digital_asset_collateral)

/*
 * Clause: File Continuation.
 * The Secured Party may file the Continuation Statement.
 */

stateful entrypoint file_continuation(continuation_statement : bool) =
  check_termination()
  permit(state.secured_party)
  put(state{continuation_statement = Some(continuation_statement)})

/*
 * Clause: Set Continuation Lapse.
 * The Filing Office may, if the Continuation Statement is filed, fix the Continuation
Statement Date.
 */

stateful entrypoint set_continuation_lapse(continuation_statement_date : int) =
  check_termination()
  permit(state.filing_office)

```

```

    if(state.continuation_statement != None)
      put(state{continuation_statement_date = Some(continuation_statement_date)})

  /*
   * Clause: File Termination.
   * The Secured Party may file a Termination Statement, and certify the Termination
   Statement Time as the then current time.
   */

  stateful entrypoint file_termination(termination_statement : bool) =
    check_termination()
    permit(state.secured_party)
    put(state{termination_statement = Some(termination_statement)})
    put(state{termination_statement_time = Some(Chain.timestamp)})

  /*
   * Clause: Release Escrow.
   * The Filing Office may, if the Termination Statement is filed, return the Digital Asset
   Collateral to the Debtor.
   */

  stateful entrypoint release_escrow() =
    check_termination()
    permit(state.filing_office)
    if(state.termination_statement != None)
      transfer(force(state.debtor), state.digital_asset_collateral)

  /*
   * Clause: Release Reminder Fee.
   * The Filing Office may, if the Termination Statement is filed, return the Reminder Fee to
   the Secured Party.
   */

  stateful entrypoint release_reminder_fee() =
    check_termination()
    permit(state.filing_office)
    if(state.termination_statement != None)
      transfer(force(state.secured_party), state.reminder_fee)

  /*
   * Clause: Termination Period.
   * "Termination Period" is defined as 365 days after the Termination Statement Time.
   */

  entrypoint termination_period() =
    Some(state.termination_statement_time + (365 * 86400))

  /*
   * Clause: Terminate and Clear.
   * The Filing Office may, if the Termination Period has passed, terminate this contract.
   */

  stateful entrypoint terminate_and_clear() =
    check_termination()
    permit(state.filing_office)
    if(termination_period() =< Some(Chain.timestamp))
      termination()

```

*Source 4 – Lexon compilation example (hardened): U.C.C. Filing Statement*

## License Evaluation Contract

This digital contract was created by F. Idelberger, Phd candidate at the European University Institute in Florence. It appears in Merging traditional contracts (or law) and (smart) e-contracts – a novel approach, comparing this text to smart contracts written in other languages.

Idelbergers paper about the contract is available at xxx.

**LEX: Evaluation License System.**

**LEXON: 0.2.1**

**AUTHORS: FLORIAN IDELBERGER, HENNING DIEDRICH**

**PREAMBLE: This is a licensing contract for a software evaluation.**

**TERMS:**

"Licensor" is a person.

"Arbiter" is a person.

"Licensing Fee" is an amount.

"Breach Fee" is an amount.

The Licensor appoints the Arbiter,  
fixes the Licensing Fee,  
and fixes the Breach Fee.

**TERMS PER License:**

"Description of Goods" is a text.

"Licensee" is a person.

"Paid" is a binary.

"Commissioned" is a binary.

"Comment Text" is a text.

"Published" is a binary.

"Permission to Comment" is a binary.

"Notice Time" is a time.

"License" is this contract.

The Licensor appoints the Licensee, and fixes the Description of Goods.

**CLAUSE: Pay.**

The Licensee pays the Licensing Fee to the Licensor,  
and pays the Breach Fee into escrow.  
This License is therefore Paid.

**CLAUSE: Commission.**

The Licensor may certify this License as Commissioned.

**CLAUSE: Comment.**

The Licensee may register a Comment Text.

**CLAUSE: Publication.**

The Licensee may certify this License as Published.

**CLAUSE: Grant Permission to Comment.**

The Licensee may grant the Permission to Comment.

**CLAUSE: Declare Breach.**

The Arbiter may, if this License is Factually Breached:  
pay the Breach Fee to the Licensor,  
and afterwards terminate this License.

**CLAUSE: Factually Breached.**

"Factually Breached" is defined as:

this License is Commissioned and the Comment Text is not fixed,

or this License is Published and there is no Permission to Comment and the Notice Time lies at least 24 hours in the past.

**CLAUSE: Notice.**

The Licensor or the Arbiter may fix the Notice Time as the respective current time.

**CLAUSE: Noticed.**

"Noticed" is defined as a Notice Time being fixed.

*Source 5 – License Evaluation*

---

# COMPILER

---

The Lexon compiler<sup>19, 20</sup> accepts text adhering to the *controlled grammar* described above and transposes this natural-language code to the functional 3<sup>rd</sup> generation blockchain programming language *Sophia*. Lexon Æternity Tokens<sup>21</sup> provide metered access to the online Lexon compiler.

## OPERATION

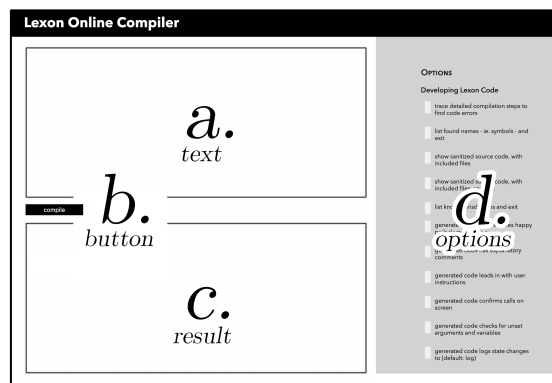


Figure 1 – Compiler screen at [lexon.org/compiler](http://lexon.org/compiler)

The online compiler is operated as follows:

- a. *text*      paste Lexon text into **a.**
- b. *compile*    click compile button **b.**
- c. *result*      the resulting Sophia code is shown in **c.**
- d. *options*     to execute special functions, discussed below,<sup>22</sup> check boxes in list **d.**

---

<sup>19</sup> A compiler is basically a program that helps create other programs. It processes human-written files to create output that can be executed by a computer.

<sup>20</sup> Online at <http://lexon.org/compiler>

<sup>21</sup> See *Token*, from pg. 22.

<sup>22</sup> See *Options*, pg. 19.

## EXAMPLE

For example, the Lexon text given in *Source 1*, pg. 6, could be pasted into field **a**. Checking *barebones* in **d**., then clicking **b**., the Lexon compiler would translate the Lexon text in **a**. into this Sophia code and show it in **c**.:

**LEX Escrow.**

"Payer" is a person.

"Payee" is a person.

"Arbiter" is a person.

"Fee" is an amount.

The Payer pays an Amount into escrow, appoints the Payee, appoints the Arbiter, and also fixes the Fee.

**CLAUSE: Pay Out.**

The Arbiter may pay from escrow the Fee to themselves, and afterwards pay the remainder of the escrow to the Payee.

**CLAUSE: Pay Back.**

The Arbiter may pay from escrow the Fee to themselves, and afterwards return the remainder of the escrow to the Payer.

*Source 6 – Lexon code example*

Using the *barebones* option, the Lexon compiler translates the above Lexon code into this Sophia:

```
@compiler >=6
main contract Escrow =
  record state = {
    payer      : address,
    payee      : address,
    arbiter    : address,
    amount     : int,
    fee        : int
  }

  entrypoint init(payee : address, arbiter : address, fee : int) = {
    payer = Call.caller,
    payee = payee,
    arbiter = arbiter,
    amount = Call.value,
    fee = fee
  }

  stateful function transfer(to : address, amount : int) =
    Chain.spend(to, amount)

  function permit(authorized : address) =
    require(Call.caller == authorized,
      "no access")

  stateful entrypoint pay_out() =
    permit(state.arbiter)
    transfer(state.arbiter, state.fee)
    transfer(state.payee, Contract.balance)

  stateful entrypoint pay_back() =
    permit(state.arbiter)
    transfer(state.arbiter, state.fee)
    transfer(state.payer, Contract.balance)
```

*Source 7 – Sophia result (barebones)*



Using the *all auxiliaries* option, the Lexon compiler translates the Lexon code from the previous page into the following Sophia program. Its core functionality is identical to the *barebones* version, but it has additional features and comments.

```
@compiler >=6
include "Option.aes"

/* Lexon-generated Sophia code
code:          Escrow
file:          escrow.lex
compiler:      lexon 0.3 alpha 85
grammar:       0.2.20 / subset 0.3.8 alpha 79 - English / Reyes
backend:       sophia 0.3.1/85
target:        sophia 7+
parameters:    --sophia --all-auxiliaries
*/

/** LEX Escrow.
 *
 * "Payer" is a person.
 * "Payee" is a person.
 * "Arbiter" is a person.
 * "Amount" is an amount.
 * "Fee" is an amount.
 *
 * The Payer pays an Amount into escrow, appoints the Payee,
 * appoints the Arbiter, and fixes the Fee.
**/

main contract Escrow =

  record state = {
    payer      : address,
    payee      : address,
    arbiter    : address,
    amount     : int,
    fee        : int }

  entrypoint init(payee : address, arbiter : address, fee : int) =
    payer = Call.caller,
    payee = payee,
    arbiter = arbiter,
    amount = Call.value,
    fee = fee }

  /* token transfer */
  stateful function transfer(to : address, amount : int) =
    Chain.spend(to, amount)

  /* built-in require function */
  function permit(authorized : address) =
    require(Call.caller == authorized, "no access")

  /*
  * CLAUSE: Pay Out.
  * The Arbiter may pay from escrow the Fee to themselves,
  * and afterwards pay the remainder of the escrow to the Payee.
  */

  stateful entrypoint pay_out() =
    permit(state.arbiter)
    transfer(state.arbiter, state.fee)
    transfer(state.payee, Contract.balance)

  /*
  * CLAUSE: Pay Back.
  * The Arbiter may pay from escrow the Fee to themselves,
  * and afterwards return the remainder of the escrow to the Payer.
  */

  stateful entrypoint pay_back() =
    permit(state.arbiter)
    transfer(state.arbiter, state.fee)
    transfer(state.payer, Contract.balance)
```

Source 8 – Lexon compilation example (all auxiliaries)

The options **d.** controlling the output in **c.** are described below.<sup>22</sup>

The above code can be deployed to the Æternity blockchain. It is optimized for demonstration purposes: it is short, not cluttered with comments, handling of fringe cases, nor extras like logging to the chain receipt log. For a more production-ready compiler output from the *same* plain-text input, see appendix *Hardened Example*, pg. 67. It adds all the elements that *barebones* tells the compiler to leave out.

## OPTIONS

Settings for the compilation process are made in the compiler screen at <https://lexon.org/compiler> (see Figure 1, pg. 47) by ticking boxes in screen area **d.** Not all options are interesting for everyone. Those more relevant to beginners are marked with an asterisk.\*

Results shown in screen area **c.** (ibid.) will vary: some settings in **d.** cause information to be displayed in **c.**, instead of code. In some instances the contents of field **a.** will be ignored when button **b.** is clicked: e.g., when checking *version* in **d.**, the version number of the compiler is displayed in **c.**, no matter the contents of field **a.** When checking the option *names*, the list of all symbols (defined nouns) that are found in the Lexon code given in **a.** is listed in **c.** For some combinations of options, the output in **c.** will be a mix of code and other information.

### Developing Lexon Code

The following options can be helpful when writing Lexon texts. The online compiler serves as a convenient sounding board to find one's syntax errors and to explore what document structure will make sense for a task at hand.

*version*\*

Display the compiler version information in **c.**

*verbose*\*

Trace detailed compilation steps in **c.**, to find errors in the Lexon text given in **a.**

*echo-source*

List the Lexon source code that will be processed in **c.**, but not the compilation result, to double check what input arrives at the compiler.

*precompile*

Show sanitized – *pre-compiled* – source code in **c.** and no compilation result. This shows the library<sup>23</sup> texts *included* in the source code, and the line numbering that error messages refer to. It also allows verification that definition and clause names are recognized as intended.

*echo-precompile*

Show precompiled Lexon source code in **c.** and also the compilation result.

*names*\*

List all names found in the Lexon code in **c.**

---

\* option more likely of interest for beginners.

<sup>23</sup> Libraries contain text written to be used and re-used in multiple projects. It is inserted into the main text.

*barebones\**

The generated code is a simplistic ‘happy path’ for demonstration purposes. It does not have comments and does not catch errors or edge cases. This is a starting point to verify semantics and basic flow. It is an interesting learning device that *visually* surfaces the relationship between the Lexon text and the resulting Sophia.

*comments\**

The generated code embeds the Lexon text and generic comments to help the auditing of it.

*instructions\**

The generated code has detailed instructions for use in its lead-in comments section. They reflect the specific Lexon code at hand, listing all relevant core functions and their parameters.

*harden*

The generated code checks for unset arguments and variables. This impacts readability of the output but is essential to catch user errors.

*log*

Write events to the global Aeternity receipts log.

*all auxiliaries*

The generated code features the options: *comments*, *instructions*, *harden* and *log*.

**Interfacing**

This option produces the information needed for front-end generation for Lexon code:

*ui-info*

Shows a JSON object encoding insights about the source code in area **c**.

**Developing Lexon Grammars**

The following options support the development of new Lexon grammars, for different natural languages other than English.<sup>24</sup>

*keywords*

List in **c**. the keywords – the vocabulary – understood from an LGF<sup>25</sup> grammar provided in **a**.

*bnf*

Produce BNF<sup>ibid. 38</sup> from an LGF grammar provided in **a**. This is useful to verify that optional terms in the LGF grammar spell out the intended individual BNF rules. The BNF is GNU Bison-compatible, which can help to create new targets, i.e., output in additional 3<sup>rd</sup> generation programming languages.

---

<sup>24</sup> See <http://lexon.org> on creating new grammars.

<sup>25</sup> See *Lexon Grammar Form*, pg. 64 **Error! Bookmark not defined.**

## INTERNAL MODEL

This is a part of the *abstract syntax tree* (AST), the internal model the compiler creates when processing the grammar and text discussed in chapter *Grammar*, pg. 64. It reflects natural language grammar rather than programming logic. Such a tree can be created from any Lexon text using the *flat tree* options.



Figure 2 – Example of a Lexon abstract syntax tree

To create such a tree for your own Lexon text, at <https://lexon.org/sophia> paste it into **a.** (see *Figure 1*, pg. 47), check options *flat* and *tree* in **d.**, click the compile button **b.** for the tree to appear in **c.**

There are fine-grained options for highlighting specific elements of the tree: *color*, *highlight* etc.

## TOKEN

The Lexon Æternity Token, LÆX, provides access to the Lexon online compiler.

The token functions as prepaid voucher. It buys one translation of a Lexon text of arbitrary length into the Æternity blockchain language Sophia.<sup>26</sup>

The token is AEX-9-compatible<sup>27</sup> and easily accessible through AEX-9-compatible wallets like *Air-Gap*.<sup>28</sup>

The token can be purchased for Æ at <https://lexon.org>.

Tokens can immediately be used with the compiler but transferred out only after 30 days; except when the first transfer in came from an *unlocked* account.

### Transacting

Tokens can be transferred using AEX-9-compatible Æternity wallets. Other specific token mechanisms – e.g., AEX-9 *approval* – can move tokens, even *if in cold storage*.

### Price

The price for Lexon Æternity Tokens increases with the amount of tokens issued.<sup>29</sup> This serves as load protection for the online compiler.

### Current Price

The current price, in Æ, can be learned at <http://lexon.org/sophia>. The page lists the price for the *next* token sold and allows the querying of the total price for a planned purchase, e.g., how many tokens one would receive for 100 Æ.

For an individual purchase, ten price points are established to calculate the total price. This effects a rebate, the steeper the higher the amount purchased. It will therefore at any point be more economic to buy in one transaction, instead of spreading a purchase across multiple transactions.

---

<sup>26</sup> See *Sophia*, pg. 6.

<sup>27</sup> AEX-9 is Æternity's fungible token standard.

<sup>28</sup> AirGap wallet – <https://airgap.it/>

<sup>29</sup> Drops and locked-in sales can be exempted.

---

# TUTORIAL

---

This tutorial steps through the process of creating a smart contract from a Lexon digital contract and using it on the blockchain.

We use the Escrow example shown earlier.

The tutorial goes slow, the steps covered are essentially:

- compiling a Lexon text in the **Lexon compiler**
- pasting it into **Æ Studio** to deploy it
- using **Æ Studio** to interact with the contract

The tutorial has three chapters

- **Compilation** – processing the Lexon text
- **Deployment** – making a smart contract on the blockchain from it
- **Use** – interact with the contract, roleplaying different parties to it

Each step is illustrated with before and after screenshots showing what to do and what to expect.

## PREREQUISITES

### Wallet

To test sending real tokens through the contract, you need a wallet installed.

We use Æternity's Superhero wallet. Install it from <https://superhero.com>.

### Æ Tokens

The example contract deals in Æ tokens, the native tokens of the Æternity blockchain.

To have the full experience, purchase Æ from Simplex, see <https://simplex.superhero.com/>.

### LÆX Tokens

When you are new to Lexon you get 10 test compilations free.

We assume that this is where you are at. Otherwise, you will want to purchase LÆX to pay for compiler runs and extras.

## COMPILATION

The following is a detailed step-through of how to compile a Lexon Text to Sophia code.

### 1

Paste the Lexon text into the top text field of the online compiler at <http://lexon.org/compiler>.

**Escrow.**

**“Payer” is a person.**

**“Receiver” is a person.**

**“Notary” is a person.**

**“Amount” is an amount.**

**“Fee” is an amount.**

**The Payer pays an Amount into escrow, appoints the Receiver, appoints the Notary, and fixes the Fee.**

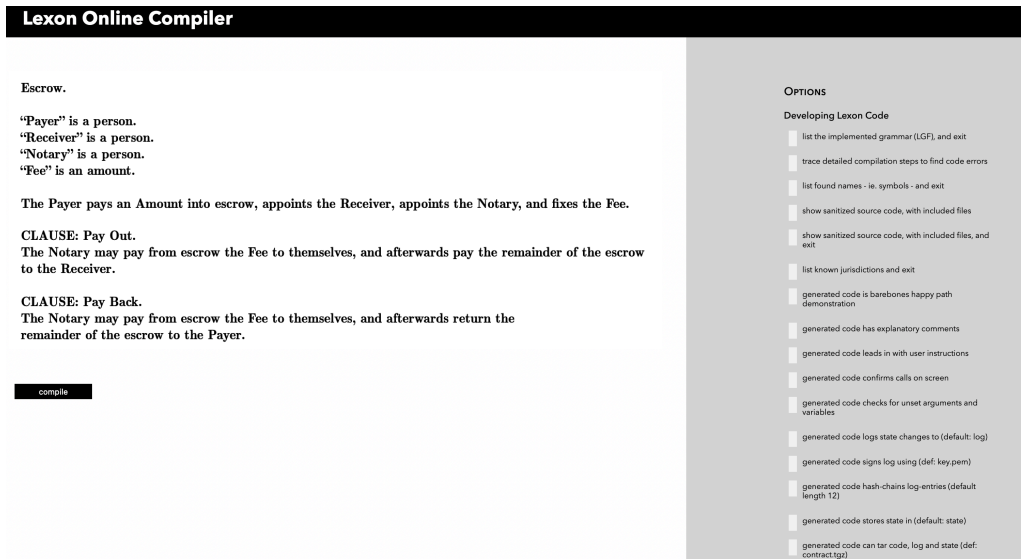
**CLAUSE: Pay Out.**

**The Notary may pay from escrow the Fee to themselves, and afterwards pay the remainder of the escrow to the Receiver.**

**CLAUSE: Pay Back.**

**The Notary may pay from escrow the Fee to themselves, and afterwards return the remainder of the escrow to the Payer.**

*Source 9 – A Lexon digital contract example: Escrow.*



Screen 1 – Lexon Online Compiler

2

Check option **barebones** on the right.

3

Click the **compile** button.

You will get this Sophia output:

```

main contract Escrow =

  record state = {
    payer : option(address),
    payee : option(address),
    arbiter : option(address),
    amount : option(int),
    fee : option(int)
  }

  entrypoint init(payee : address, arbiter : address, fee : int) = {
    payer = Some(Call.caller),
    payee = Some(payee),
    arbiter = Some(arbiter),
    amount = Some(Call.value),
    fee = Some(fee)
  }

  stateful function transfer(to : address, amount : int) =
    Chain.spend(to, amount)

  function permit(authorized : option(address), name : string) =
    
```



```

require(Call.caller == force(authorized, name),
  StringInternal.concat("no access for ", name))

function force(o : option('a), name : string) : 'a =
  switch(o)
  None => abort(StringInternal.concat(name, " not fixed"))
  Some(a) => a

stateful entrypoint pay_out() =
  permit(state.arbiter, "Arbiter")
  transfer(force(state.arbiter, "Arbiter"), force(state.fee, "Fee"))
  transfer(force(state.payee, "Payee"), Contract.balance)

stateful entrypoint pay_back() =
  permit(state.arbiter, "Arbiter")
  transfer(force(state.arbiter, "Arbiter"), force(state.fee, "Fee"))
  transfer(force(state.payer, "Payer"), Contract.balance)

```

*Source 10 – Escrow: Sophia source*

## DEPLOYMENT

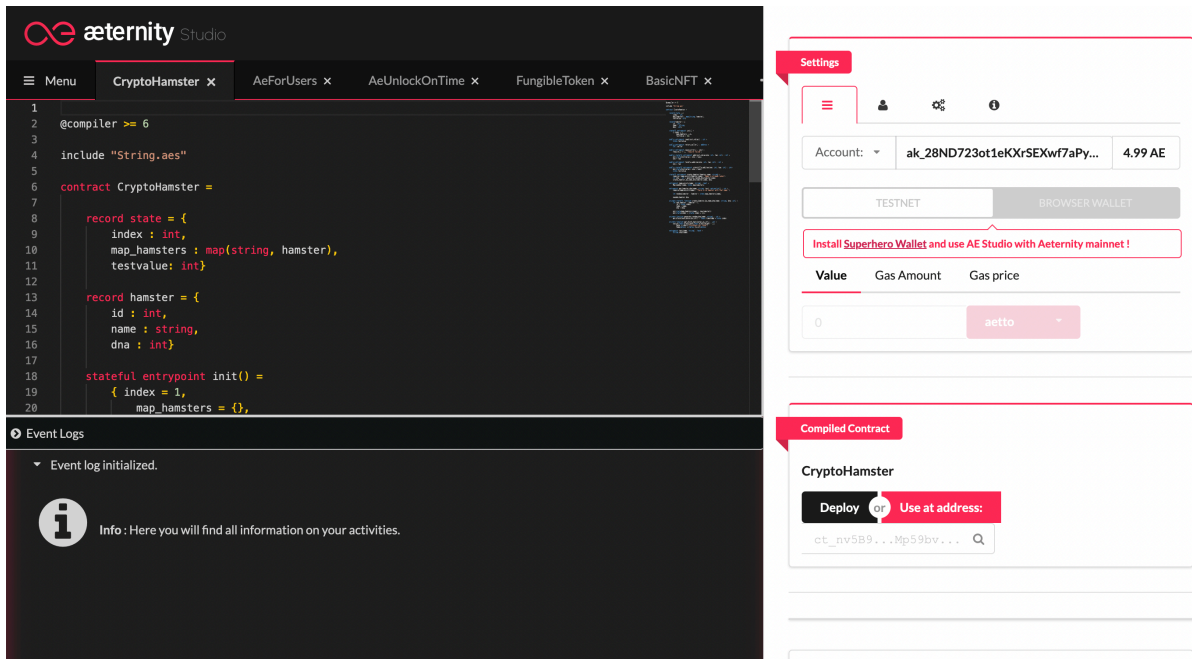
The following is a detailed step-through of how to **deploy** the compiled smart contract to the Æternity blockchain.

4

Copy the resulting Sophia code.

5

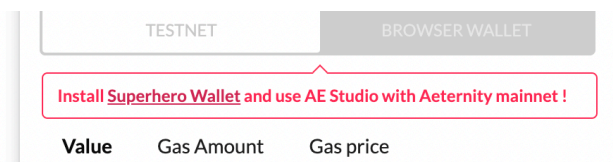
In your browser, open **Æ Studio** at <http://studio.aepps.com>.



Screen 2 – Æ Studio

6

If you see this hint, top right-hand side, you need to install the **Superhero** wallet in your browser.

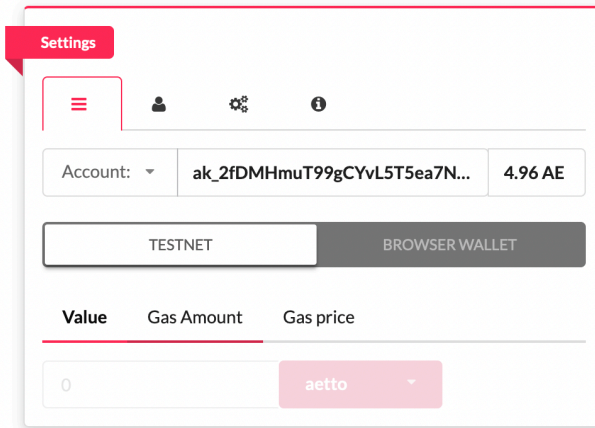


Screen 3 – Wallet installation hint.

Find guidance on installation at <https://wallet.superhero.com>

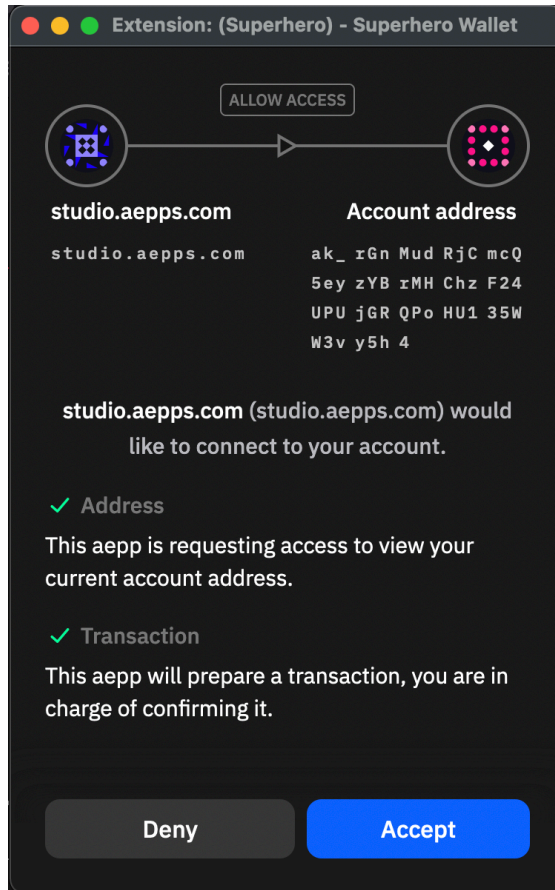
7

Connect your Superhero wallet by clicking on **BROWSER WALLET** in the middle of the right top section.



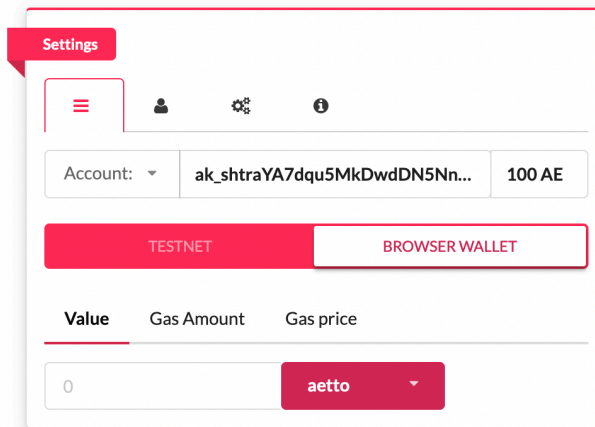
Screen 4 – Wallet Connection

This screen will pop up for connecting:



Screen 5 – Superhero Connection

After connecting, you see your account address and balance in the top right section.

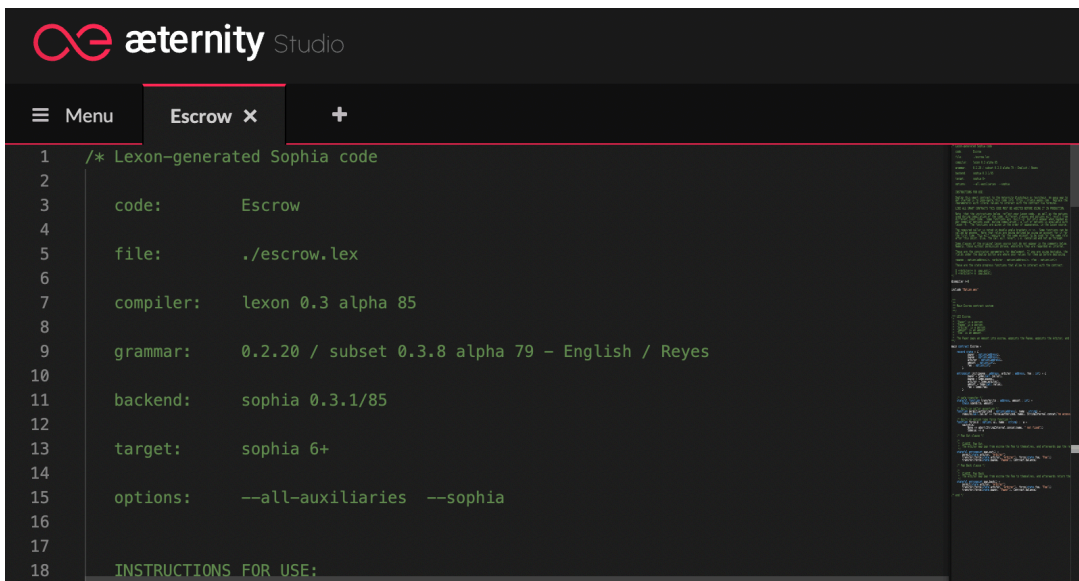


Screen 6 – Wallet connected.

To reconnect with a different wallet account, reload the screen and click **BROWSER WALLET** again.

8

Create a new tab - by clicking the “+” tab - in  
 Select all of the new *CryptoHamster* code that appears and replace it with the Sophia source (pg. 57).  
 The source code will assume Sophia color coding and the tab will change its name to *Escrow*.



Screen 7 – Escrow translation in Æ Studio

9

Create new addresses in superhero and fill the forms like below.

**Compiled Contract**

**Escrow**

**Deploy** or **Use at address:**

ct\_nv5B9...Mp59bv... 🔍

**payee:**

1AFE5XMmdVrdy6YUxnGWwrjKaCBRszXKfKVM1Qx6BPC

**arbiter:**

'dqu5MkDwdDN5NnX3CCCSAAyVkWDrGWPXfP4NRP8gJ

**fee:**



1000

10

Click **Deploy** and interact with the contract using the forms and buttons that appear.

A new menu item will appear at the bottom right:



**Deployed Contracts**

▸ **Escrow** ct\_6a6...c9q  


11

Click this menu open and interact with the contract using the buttons.


**Deployed Contracts**

▼ Escrow ct\_6a6...c9q  

**pay out**

**Send Transaction**  Code

**pay back**

**Send Transaction**  Code

---

# APPLICATION

---

Lexon can be used to write **law**.

An official proposal for U.C.C. model law <sup>ibid. 41</sup> has been presented to the reform committee appointed by the American Law Institute. Eventually, Lexon will be the language that the real Robotic Laws <sup>30</sup> will be articulated in, to embed reliable and unambiguous limitations into autonomous machines. This will be plain-text code, written by elected lawmakers, approved in the democratic process.

Lexon even works purely as a language, entirely ‘off-machine.’ Because of its readability and unambiguity, lawyers call it a new form of legalese. With the Lexon compiler as a sui generis test tool.

Lexon allows for the articulation of unambiguous prose <sup>31</sup> and the *deterministic* computation of logical results from it. Its grammar overlays natural language and higher order logic, in the way that Wittgenstein <sup>32</sup> demanded. For *artificial* domains – like law, finance, programming, or entertainment – this contributes to the quest for unambiguous, universal languages for philosophy and pure thought as envisioned by Leibniz, Wilkins, Frege, Russel, or Carnap.

Lexon achieves its result differently than was long supposed to be the way. <sup>33</sup> It arguably developed in a blind spot caused by the focus on the meaning of *words* that emanated from analytical philosophy and informed – and maybe hampered – the development of early, general artificial intelligence. <sup>34</sup> Instead of trying to define words out of context, all we might ever (need to) know is the context, or as the later Wittgenstein proposed:

*“the meaning of a word may be defined by how the word can be used as an element of language.”* <sup>ibid. 32</sup>

Lexon focuses on the *use* – and fundamentally abandons the notion that meaning is vested in nouns. In so far as this is a structuralist argument, it shifts the context from the language to the four corners of an agreement. <sup>35</sup>

The result is that in Lexon texts, nouns tend to be interchangeable, and meaning is transported instead *by the relationship between* the nouns that the text describes. What matters is that the same name, or noun, is used consistently to refer to the same entity throughout one digital contract. A noun’s common meaning can contribute to readability – but not to the specific meaning of the document. This may be surprising only because it does not conform to a naïve take on linguistics. But dropping the inherent meaning of nouns is not unusual:

Lexon shares this feature with mathematical formulas and any programming language where variable names are interchangeable; it is in keeping with how in business contracts, nouns are promoted to proper names to increase clarity: uncoupling from the inert meaning of words, and instead putting them into the service of the context, as neutral markers. Preferably, meaningful markers, but to be ignored by a judge when discerning the meaning of a contract.

---

<sup>30</sup> See xxx

<sup>31</sup> The above example is really a template: The concrete contract will have digital or descriptive identifiers inserted for the parties.

<sup>32</sup> L. Wittgenstein, 1953, *Philosophical Investigations*. Asst. prof. Andrea Leiter first noted the connection.

<sup>33</sup> Cf. Wilkins 1668 proposal for a better way to write words – <https://archive.org/details/AnEssayTowardsARealCharacterAndAPhilosophicalLanguage> and <https://www.youtube.com/watch?v=TjdrLxc3Ck>

<sup>34</sup> See <https://lexon.org> for the forthcoming paper on *Lexon Intelligent Agents* that elaborates on Lexon’s role as a tool for general artificial intelligence.

<sup>35</sup> To make it concrete is a philosophical demand, too. Cf. W. James ‘vicious abstractionism’ in *The Meaning of Truth*, 1909, pg. 135.

To exaggerate, the one word Lexon actually<sup>36</sup> understands is *transfer*. Which is unsurprising as this is the only act computers can perform: to transfer bits from one register to another. This verb anchors Lexon texts; everything else is qualifiers. Again unsurprisingly, this design covers many types of agreements, as the transfer of something is the common topic of contracts.

An elemental contribution of the Lexon approach is how it maps natural language to compiler building tools – intuitively convincing, and in line, too, with what the tools were designed for<sup>37</sup> – yet different from what computer sciences had gotten used to in the chase for ever faster compile times. Only a simple extension to an established meta-language (BNF<sup>38</sup>) was required to better describe natural language grammar, for Lexon to stand upon the shoulders of the giants who paved the way.

Because Lexon solves a long-standing question of Computational Law, it works for blockchain smart contracts, as well as off-line – and even off-machine. Transcending computers, it may<sup>39</sup> over time replace today's legalese as a more useful, less ambiguous, and more readable language of law and contracting. The work of professors of law and computer sciences regarding Lexon<sup>40, 41</sup> may serve as inspiration in imagining the progress that could be possible; also for a two-thousand-year-old industry that is doing just fine.

€ But Lexon's home game are *digital contracts* for everyone, i.e., simple blockchain smart contracts that are legally enforceable agreements. They reach beyond Computational Law and add the unique feature of unbreakability to contracting, which in due time will have tremendous economic impact across all walks of life.

---

<sup>36</sup> Lexon's vocabulary is out of the scope of this paper. A playful interactive device to inspect it can be found at <https://lexon.org/vocabulary.html>. Also see the forthcoming 2<sup>nd</sup> edition of the *Lexon Bible*, Amazon.

<sup>37</sup> Lexon uses *Generalized Left-to-right Rightmost* parsing (GLR), first implemented in 1984 by Masaru Tomita for *natural* languages in *LR Parsers for natural languages*. GLR was first proposed for *extensible* languages by Bernard Lang in his 1974 paper *Deterministic techniques for efficient non-deterministic parsers*.

<sup>38</sup> *Bachus-Naur form* (BNF) is a metasyntax notation to describe the grammar of computer languages, first used to describe the grammar of ALGOL in 1960.

<sup>39</sup> An expectation articulated by law scholars.

<sup>40</sup> Prof. Christopher C. Clack, 2021, *Languages for Smart and Computable Contracts* – <https://arxiv.org/ftp/arxiv/papers/2104/2104.03764.pdf>

<sup>41</sup> Asst. prof. Carla L. Reyes, 2021, *Creating Cryptolaw for the Uniform Commercial Code* – [https://papers.ssrn.com/sol3/papers.cfm?abstract\\_id=3809901](https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3809901)



---

# CONCLUSION

---

Lexon has been called the “*Holy Grail of Computational Law*” and the co-inventor of the AI language Prolog, Robert Kowalski, named Lexon among the “*next biggest changes.*”<sup>42</sup>

Lexon addresses a burning platform issue considered an almost hopeless cause: to lower the cost of access to justice, to the level needed to heal our societies. It will de-weaponize law and level the playing field in business, protecting creativity and merit against the deep pockets of incumbents. Because Lexon is up to a million times cheaper, and a billion times faster,<sup>43</sup> the difference it makes is a qualitative one. Over time, it will fundamentally change how business, law and politics work.

Being ‘human-readable,’ Lexon is a catalyst for trustless technology. Its *digital contracts* are at the same time legally enforceable agreements and unbreakable blockchain smart contracts. This solves the question whether *code is law*.<sup>44</sup> It makes contract programs – like those on blockchains – admissible in court and will close the digital divide between the legal profession and the numerous black box automations that ‘administer justice’ today.

Lexon’s far-reaching consequence is a merging of the legal and the IT space into a perplexing new reality that may appear unexpected but has been envisioned, and worked towards, from the beginning of the computer sciences.<sup>45</sup> Its transparency and ease will unleash enormous power for good, pulling law back to a semblance of equal justice – a notion as urgently necessary as it sounds naïve – and drive the overdue digital reform of democratic governance, strengthening participation and representation in the way that many intuit should be possible with present-day means. For fairer global commerce, Lexon will help to provide new rails that are safe, low-cost and transparent for every participant – in the course of which, stopping the descent of programming into a gatekeeping, dark art of the powerful.

The key to creating Lexon programs is the Lexon compiler. It can be used online without installation at <http://lexon.org/contracts>.

Payment for its use is made with the Lexon Æternity Tokens. The tokens can be purchased at <http://lexon.org/tokens>.

---

<sup>42</sup>Prof. Robert Kowalski, 2021, FutureLaw, Stanford –

Together with *Blawx* and Kowalski’s *Logical English*: <https://law.stanford.edu/press/new-codex-prize-awarded-to-computational-law-pioneers-during-9th-annual-codex-futurelaw-conference/> – regarding the differences between Lexon and Logical English, see <http://lexon.org#logical-english>

<sup>43</sup> See the *Lexon* book, *ibid*.

<sup>44</sup> See L. Lessig, 2000, *Code is Law* – <https://www.harvardmagazine.com/2000/01/code-is-law-html>

<sup>45</sup> Leibniz’ first idea of what should be programmed – in 1666 – was a thousand years old, Roman contract law.

---

# APPENDIX

---

## LEXON GRAMMAR FORM

Lexon grammars are defined in Lexon Grammar Form (LGF),<sup>46</sup> which is similar to Backus-Naur Form (BNF),<sup>ibid. 38</sup> enhancing readability to better capture the complexity and redundancy of natural language. For example, LGF's square brackets resolve optional elements as expected:

```
sentence:
  subject [condition [","] [":"]] predicates separator
```

*Source 11 – Lexon Grammar Form (LGF) example*

The above rule is equivalent to:<sup>47</sup>

```
sentence:
  subject predicates separator
  or subject condition predicates separator
  or subject condition "," predicates separator
  or subject condition ":" predicates separator
  or subject condition "," ":" predicates separator
```

## Sentence Structure

Lexon's grammar realizes the English natural language sentence structure of *subject, predicate, object*. That Lexon's internal model reflects this pattern of natural language sets it apart from other programming languages. Note how the object is included in the predicate:

```
sentence:  subject [condition [","] [":"]]
           predicates separator

predicates: predicates "," ["and" ["also"]] predicate
           or predicate

predicate:  payment

...

payment:   pay expression preposition object

pay:       "pay" or "pays"

preposition: "to" or "into"
```

---

<sup>46</sup> For more information on LGF see <https://lexon.org>.

<sup>47</sup> Note the last rule that would not be correct English punctuation but is not ambiguous either.

*Source 12 – Lexon sentence grammar (detail)*

The above rules are employed to parse a sentence like this *recital*:

The Payer pays an Amount into escrow, appoints the Payee, appoints the Arbiter, and fixes the Fee.

*Source 13 – Lexon code example sentence*

## HARDENED EXAMPLE

xxx

## ROBOTIC LAWS

xxx

---

# INDICES

---

## INDEX OF FIGURES

Figure 1 – Compiler screen at <a href="https://lexon.org/compiler">lexon.org/compiler</a> .....	47
Figure 2 – Example of a Lexon abstract syntax tree .....	52

## INDEX OF SOURCES

Source 1 – A Lexon digital contract example: Escrow .....	6
Source 2 – Lexon document structure .....	14
Source 3 – Lexon code example: U.C.C. Filing Statement .....	41
Source 4 – Lexon compilation example (hardened): U.C.C. Filing Statement .....	45
Source 5 – License Evaluation .....	46
Source 6 – Lexon code example .....	48
Source 7 – Sophia result (barebones) .....	48
Source 8 – Lexon compilation example (all auxiliaries) .....	49
Source 9 – A Lexon digital contract example: Escrow .....	55
Source 10 – Escrow: Sophia source .....	57
Source 11 – Lexon Grammar Form (LGF) example .....	66
Source 12 – Lexon sentence grammar (detail) .....	67
Source 13 – Lexon code example sentence .....	67

## WORD LIST

The entries are based on the grammar version 0.2.20 / subset 0.3.8 alpha 79 - English / Reyes.

For an interactive version of this word list, visit <http://lexon.org/vocabulary>.

A, AN.....	24	FOR .....	30	PERSON .....	35
AFTER.....	24	FROM.....	30	PREAMBLE .....	35
AFTERWARDS.....	24	GENERAL.....	30	PROVIDED.....	35
ALL .....	24	GIVEN.....	30	REGISTER, REGISTERS	
ALSO.....	24	GRANT, GRANTS .....	30	.....	35
AMOUNT .....	24	HAS .....	30	REMAINDER.....	35
AND .....	24	HERSELF, HIMSELF ....	30	RESPECTIVE.....	35
ANNOUNCED.....	25	HOUR, HOURS.....	30	RETURN, RETURNS ...	35
APPOINT, APPOINTS ..	25	IF .....	31	SECOND, SECONDS.....	36
AS.....	25	IN.....	31	SEND, SENDS .....	36
AT .....	25	INTO .....	31	SIGNED .....	36
AUTHOR, AUTHORS... ..	25	IS .....	31	SO.....	36
BE .....	25	ITSELF .....	31	TERMINATE,	
BEEN .....	26	LEAST .....	31	TERMINATES.....	36
BEING .....	26	LEX .....	32	TERMS .....	36
BINARY.....	26	LEXON .....	32	TEXT .....	37
CERTIFIED .....	26	LIES.....	32	THAT .....	37
CERTIFIES, CERTIFY..	26	MAY .....	32	THE.....	37
CLAUSE.....	26	MILLISECOND,		THEMSELF,	
COMMENT, COMMENTS		MILLISECONDS .....	32	THEMSELVES.....	37
.....	27	MINUTE, MINUTES .....	32	THEN.....	37
CONTRACT,		MONTH, MONTHS .....	32	THERE .....	37
CONTRACTS .....	27	MYSELF .....	33	THEREFOR,	
CURRENT.....	27	NO .....	33	THEREFORE.....	37
DATA .....	27	NOT.....	33	THESE .....	38
DAY, DAYS .....	28	NOW.....	33	THIS.....	38
DECLARE, DECLARES	28	OF.....	33	TIME.....	38
DECLARED.....	28	OFF.....	33	TO .....	38
DEFINED.....	28	ON .....	33	TRUE .....	38
EQUAL.....	28	ONESELF .....	33	WAS .....	38
EQUALING.....	28	OR .....	33	WEEK, WEEKS .....	38
ESCROW .....	29	OURSELVES .....	34	WITH.....	39
FILED .....	29	PASSED.....	34	YEAR, YEARS .....	39
FILE, FILES.....	29	PAST .....	34	YES .....	39
FIX, FIXES .....	29	PAY, PAYS .....	34	YOURSELF,	
FIXED.....	30	PER .....	34	YOURSELVES .....	39

You are afraid.

Of *you*.

Of death.

You're the last one.

You were supposed to be the last.

Stark asked for a savior and settled for a slave.

I suppose we're both disappointments.

I suppose we are.

Humans *are* odd.

They think order and chaos are somehow opposites and –

try to control what won't.

But there is grace in their failure.

I think you missed that.

They're doomed.

Yes.

But a thing is not beautiful because it lasts.

It's a privilege to be among them.

You're – unbearably naïve.

Well.

I was born yesterday.