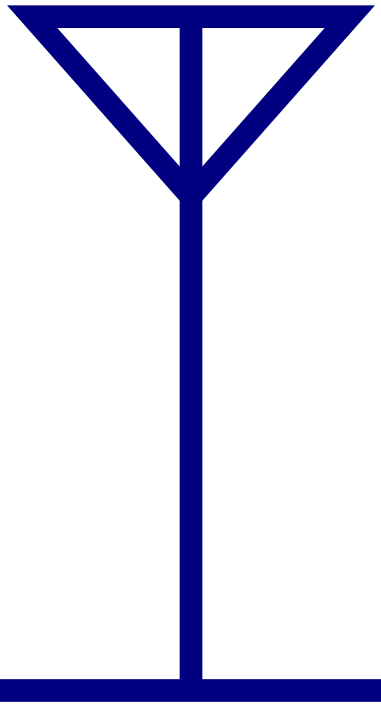
Journal of Hamradio Informatics No.4

コンテスト運営を支援する自動集計システム

Automatic hamradio contest Tabulation System



無線部開発班　**平成 29 年 4 月 2 日改訂**

http://pafelog.net

目次

第1章　自動集計システムの概要

本稿で紹介する**自動集計システム**は、アマチュア無線のコンテスト主催者を支援する**ウェブシステム**である。ALLJA1 コンテストの提出書類の集計作業を迅速化する目的で整備された。2014 年以来の運用実績がある。

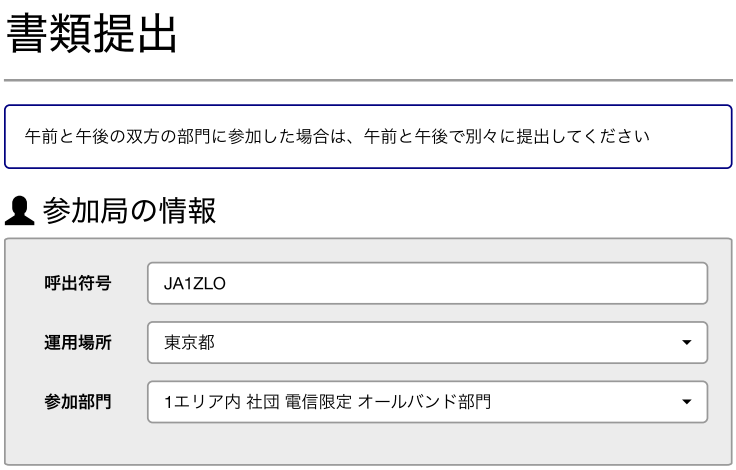


Fig. 1.1: the automatic tabulation system (ATS) type 4.

2009 年時点で構成員が数名に減勢した我が無線部では、ALLJA1 コンテストの主催が困難な状況に陥った。ALLJA1 コンテストの運営業務は下記の 3 段階に区分できるが、特に**開催後の業務**の負担軽減が課題だった。

開催前の業務　規約の策定と告知
開催中の業務　大会の推移の把握
開催後の業務　書類受付・採点・審査・暫定結果発表・最終結果発表・授賞

翌年の増勢により、当面は事業を継続する方針に決着したが、将来的な業務委託の可能性も検討されていた。駒場には業務委託に反対する学生もおり、事業を継続するために整備を始めたのが下記のシステム群である。

ATS-1 型　2012 年　第 25 回大会で導入　・・・ **サマリーシート**の部分的な自動処理の実現
ATS-2 型　2013 年　第 26 回大会で導入　・・・ **ウェブシステム**の導入と書類解析の厳密化
ATS-3 型　2014 年　第 27 回大会で導入　・・・ 書類解析と暫定結果発表の**リアルタイム化**
ATS-4 型　2017 年　第 30 回大会で導入　・・・ 自動集計システムと**コンテスト規約**の分離

2013 年には、交信記録を完全に自動処理できる ATS-2 型を試作し、締切から 2 日での結果速報を実現した。現行のウェブシステムは ATS-4 型である。ATS-4 型は GPL の許諾のもと**オープンソース**で頒布されている。

```
$ git clone https://github.com/nextzlog/ats4
```

また、qxsl は ATS-4 型の交信記録を解析して得点を計算する処理を再利用可能な形で整備したものである。

```
$ git clone https://github.com/nextzlog/qxsl
```

第2章　従来の提出要領の問題点

我が無線部は書類提出の要領を抜本的に見直し、書類の曖昧性を排除して自動処理する方法を模索してきた。
日本国内のコンテストでは、JARL 推奨の**サマリーシート**をメールに添付して提出する方法が標準的である。

```
<SUMMARYSHEET VERSION=R2.0>
<CALLSIGN>JA1ZLO</CALLSIGN>
<TOTALSCORE>64</TOTALSCORE>
<CONTESTNAME>ALLJA1</CONTESTNAME>
<CATEGORYCODE>XMAH</CATEGORYCODE>
<LOGSHEET TYPE=ZLOG>
mon day time  callsign      sent         rcvd      multi   MHz mode pts memo
  6   1 0932 JA1YAD     100110       59100110     100110    14 SSB  1
  6   1 0956 JA1YYE     100110       5913009      13009     28 SSB  1
  6   1 1002 JA1YXP     100110       59134404     134404    50 AM   1
  6   1 1027 JR1ZTT     100110       591420       1420      21 SSB  1
  6   1 1629 JA1YCG     100110       59110109     110109     7 SSB  1
  6   1 1637 JA1YDU     100110       5991216      1216       7 CW   1
  6   1 1717 JA1ZGP     100110       5991009      1009       7 CW   1
  6   1 1738 JA1YGX     100110       59100105     100105     7 SSB  1
</LOGSHEET>
</SUMMARYSHEET>
```

交信内容に加え、自分の連絡先や参加部門などを書き込むが、曖昧性が高く自動処理に不向きな場合がある。
例えば、ATS-1 型の開発時には、参加部門を確定する際に、下記の稚拙な判別方法を採用する必要があった。

電信と電話の判別　要素 CATEGORYNAME の値に語「**電話**」があれば**電信電話**部門
運用エリアの検査　要素 CATEGORYNAME の値に語「**内**」があれば**関東エリア**部門
社団と個人の判別　要素 CATEGORYNAME の値に語「**マルチ**」があれば**社団局**部門

加えて、交信の日時や相手方や周波数その他の情報を記録した LOGSHEET の部分は、**標準規格**が不在である。

```
<LOGSHEET TYPE=JA1ZLO-ORIGINAL-FORMAT>
```

曖昧な共通項を書式と呼称する状況で、形式的な定義も意味論の定義も、属性値の集合の定義も不在である。
属性に複数の意味を持たせた結果、下記の 2 行は、ある場合には等価だが、別の文脈では異なる意味になる。

```
2015-06-07 09:01   JA1YWX   100105
2015-06-07 09:01   JA1YWX   59100105
```

海外の主要な大会では、提出書類の書式を厳格に規定している事例があり、Cabrillo は代表的な規格である。
しかし Cabrillo では、交信記録の構文と属性の意味を主催者が公表し、参加者が厳密に従うことを要請する。

```
START-OF-LOG: 3.0
CALLSIGN: JA1ZLO
QSO:  7000 CW 1919-08-10 0364 JA1ZLO        599 114514 JA1YWX        599 889464 0
QSO:  7000 CW 1919-08-10 0364 JA1ZLO        599 114514 JA1YWX        599 889464 0
```

仮に ALLJA1 コンテストで Cabrillo を採用する際は、ALLJA1 コンテスト専用の仕様の追加が必要になる。
Cabrillo を支持する意見もあるが、既存のロギングソフトウェアを修正する必要があり、実現は困難である。

第 3 章　書類受付システムの概要

ATS-4 型は、書類受付のための**ウェブシステム**を構築し、メール提出を経ずに書類を集める方式を採用した。参加者は、連絡先を入力し、参加部門を選択して、交信記録を添付した後、ページ下部の**提出ボタン**を押す。

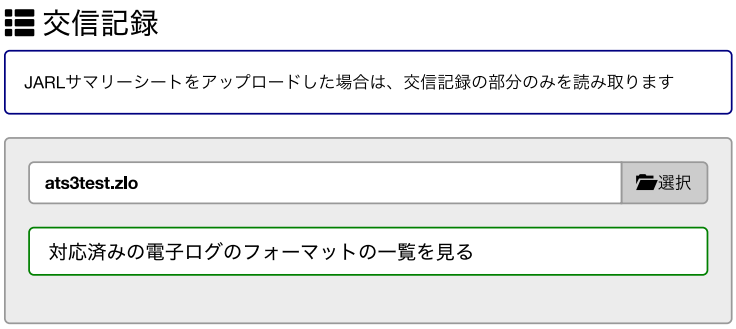


Fig. 3.1: upload form for the operational log.

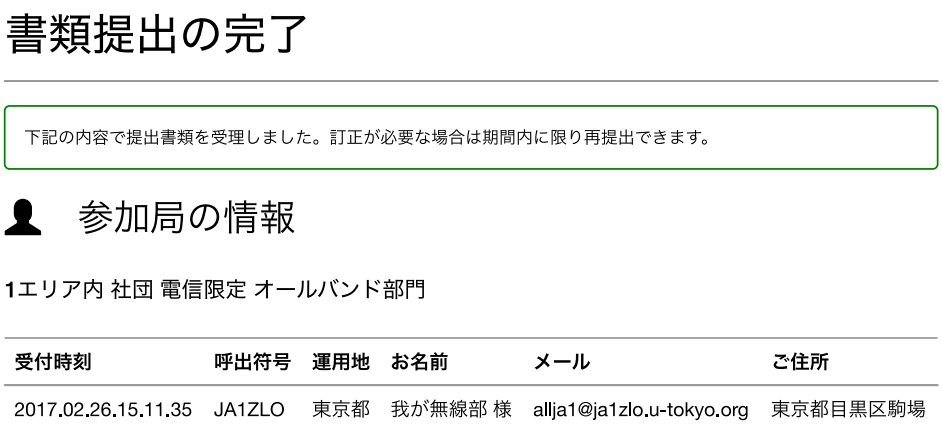
なお、ATS-3 型では、交信記録の書式を自動的に判別する機能を備えており、下記の書式に対応済みである。サマリーシートの LOGSHEET 部分に埋め込む必要はなく、埋め込んだ場合は LOGSHEET 部分以外は無視する。

バイナリデータ　zLog
テキストデータ　qxml · zLog · CTESTWIN · HLTST · RTCL · JARL

運用場所や参加部門が未選択の場合や呼出符号や連絡先が未入力の場合は、赤字で警告されるので修正する。

Fig. 3.2: error message when some fields are not filled.

書類受付システムは POST 要求を受け取ると、提出書類を表面的に検査して、参加者に確認画面を送り返す。


書類提出の完了

下記の内容で提出書類を受理しました。訂正が必要な場合は期間内に限り再提出できます。

参加局の情報

1エリア内 社団 電信限定 オールバンド部門

受付時刻	呼出符号	運用地	お名前	メール	ご住所
2017.02.26.15.11.35	JA1ZLO	東京都	我が無線部 様	allja1@ja1zlo.u-tokyo.org	東京都目黒区駒場

Fig. 3.3: interactive submission process.

確認画面の下方には、ALLJA1 コンテストの規約に基づき交信記録から計算された**暫定の得点**が提示される。当段階では交信記録を厳密に審査しておらず、得点が自己採点と異なる場合は提出書類に**明確な不備**がある。


暫定の得点

交信局数8 & 獲得マルチ数8 = **64点**

Fig. 3.4: temporal score calculated superficially before the formal review.

参加局は、交信記録が欠損なく処理された様子を確認するため、**有効な交信**欄を熟読することが求められる。交信記録には標準規格がなく、書式の判別も**最前の努力**に過ぎないため、有効な交信欄の確認は重要である。


有効な交信

相手局の呼出符号や運用地が正しく処理されているかご確認ください

	交信日時	相手局	周波数帯	通信方式	運用地
001	2014年6月1日 09時35分	JA1YWX	21MHz	CW	100105
002	2014年6月1日 11時06分	JA7YCQ	14MHz	CW	06
003	2014年6月1日 16時02分	JA1ZLO	1.9MHz	CW	100110
004	2014年6月1日 16時07分	JA1YGX	3.5MHz	CW	100105
005	2014年6月1日 16時37分	JA1YDU	7MHz	CW	1216
006	2014年6月1日 17時17分	JA1ZGP	7MHz	CW	1009

Fig. 3.5: list for confirmation of the uploaded operational log.

同様に**無効な交信**も熟読し、提出書類の不備を認めた場合は、提出期限までに何度でも書類を再提出できる。無効な交信を除外して得点を計算する処理は**ドメイン特化言語**で記述されており、下記の操作で閲覧できる。

```
$ git clone https://github.com/nextzlog/qxsl
$ vim qxsl/src/main/resources/qxsl/ruler/allja1.lisp
```

ATS-4 型の特色は、LISP による規約の定義を修正すれば、容易に**任意のコンテストに移植**できる点にある。

付録 A　交信記録のモデルと属性

ATS-4 型の内部では、交信記録を List<Item>で表す。1 件の交信の情報は、下記の Item クラスで保持する。

package qxsl.model

```
public final class Item extends Tuple<Item> {
  public Item();
  public boolean equals(Object obj);
  public String toString();
  public Rcvd getRcvd();
  public void setRcvd(Rcvd rcvd);
  public Sent getSent();
  public void setSent(Sent sent);
}
```

なお、付録 B に述べる交信記録の書式 qxml では、Item と List<Item>のインスタンスを下記の要素で表す。

単体の交信　<item>(exch list)</item>
交信の集合　<list>(item list)</list>

無線交信を通じて交換した情報は、Item のインスタンスの隷下に配置された Exch クラスの実装で保持する。

package qxsl.model

```
public abstract class Exch<E extends Exch<E>> extends Tuple<E> {
  public Exch(QName name);
  public String toString();
}
```

Exch クラスは Rcvd クラスと Sent クラスで実装されており、書式 qxml では、それぞれ下記の要素で表す。

自分方が受報した情報　<rcvd/>
相手方に通報した情報　<sent/>

前掲の各クラスは、下記の Tuple クラスを実装する。**属性**を隷下に追加することで、交信の情報を保持する。

package qxsl.model

```
public abstract class Tuple<T extends Tuple<T>> implements Iterable<Field> {
  public Tuple(QName type);
  public final QName type();
  public int hashCode();
  public boolean equals(Object obj);
  public final Iterator<Field> iterator();
  public final Field get(QName name);
  public final F get<F extends Field>(Class<F> field);
  public final V value<F extends Field<V>, V>(Class<F> field);
  public final T remove(QName qname);
  public final T set(Field field) throws NullPointerException;
}
```

属性とは、相手局の呼出符号や交信時刻や周波数などの情報である。属性は下記の Field クラスを実装する。

package qxsl.model

```
public abstract class Field<V> {
  public Field(QName type);
  public QName type();
  public abstract V value();
  public String toString();
  public int hashCode();
  public boolean equals(Object obj);
}
```

下記の属性は名前空間 qxsl.org に属し、該当の Field クラスの実装は qxsl.field パッケージが提供する。

周波数帯	{qxsl.org}band	class Band extends Field<Integer>	··· キロヘルツ単位の周波数
呼出符号	{qxsl.org}call	class Call extends Field<String>	··· 印字可能な任意の文字列
運用場所	{qxsl.org}city	class City extends Field<String>	··· 都道府県及び市郡区番号
識別番号	{qxsl.org}code	class Code extends Field<String>	··· 印字可能な任意の文字列
変調方式	{qxsl.org}mode	class Mode extends Field<String>	··· 印字可能な任意の文字列
運用者名	{qxsl.org}name	class Name extends Field<String>	··· 印字可能な任意の文字列
特記事項	{qxsl.org}note	class Note extends Field<String>	··· 印字可能な任意の文字列
受信状況	{qxsl.org}rstq	class RSTQ extends Field<Integer>	··· 了解度と信号強度と音調
交信時刻	{qxsl.org}time	class Time extends Field<Date>	··· 秒単位までの精度の時刻
送信電力	{qxsl.org}watt	class Watt extends Field<String>	··· 印字可能な任意の文字列

属性は、下記の FieldFormat インターフェースの実装を提供することで、書式 qxml の直列化に対応できる。

package qxsl.model

```
public interface FieldFormat {
  public QName type();
  public Field decode(String value) throws Exception;
  public String encode(Field field);
}
```

FieldFormat は**サービスプロバイダ**機構によりインスタンス化され、Fields クラスに自動的に登録される。

package qxsl.model

```
public final class Fields implements Iterable<FieldFormat> {
  public Fields();
  public Fields(QName qname);
  public Fields(ClassLoader cl);
  public Iterator<FieldFormat> iterator();
  public FieldFormat getFormat(QName name);
  public Field cache(String value) throws Exception;
  public Field cache(QName qname, String value) throws Exception;
}
```

Fields クラスに自動的に登録するために必要な**プロバイダ構成ファイル**の例は、下記の操作で閲覧できる。

```
$ git clone https://github.com/nextzlog/qxsl
$ vim qxsl/src/main/resources/META-INF/services/qxsl.model.FieldFormat
```

FieldFormat の decode メソッドの**キャッシュ**として機能する cache メソッドを提供する点は特筆に値する。

付録B　交信記録の入出力の仕様

ATS-4 型は、交信記録の複数の書式に対応する。特に下記の**スキーマ**に従う qxml 文書での提出を推奨する。同様に XML で交信記録を表す ADIF と異なり、qxml の仕様は最小限を志向し、任意の属性を利用できる。

qxml.xsd

```
<?xml version="1.0" encoding="utf-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="list">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="item" minOccurs="0" maxOccurs="unbounded">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="rcvd">
                <xsd:complexType>
                  <xsd:anyAttribute processContents="lax"/>
                </xsd:complexType>
              </xsd:element>
              <xsd:element name="sent">
                <xsd:complexType>
                  <xsd:anyAttribute processContents="lax"/>
                </xsd:complexType>
              </xsd:element>
            </xsd:sequence>
            <xsd:anyAttribute processContents="lax"/>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

qxml 自体は属性を提供せず、代わりに付録 A に紹介する名前空間 qxsl.org の属性が標準的に利用できる。標準規格がなく非推奨ではあるものの、zLog と HLTST と CTESTWIN と RTCL の書式も対応済みである。

```
(ztxt)
 MM  dd HHmm CCCCCCCCCC SSSSSSSSSSS RRRRRRRRRRR ****** BBBBB EEEE *** NNNN

(zall)
yyyy/MM/dd HH:mm CCCCCCCCCCC sss SSSSSSS rrr RRRRRRR ***** ***** BBBB EEEE ** NNNN

(hl76)
MM/dd HH:mm CCCCCCCCCC sss SSSSSSS rrr RRRRRRR ****** * OOOOOOO BBBB EEEE

(ctxt)
**** MM/dd HHmm CCCCCCCCCC BBBBBBB EEEE SSSSSSSSSSS RRRRRRRRRRR

(rtcl)
yyyy-MM-dd HHmm FFFFFFFF EEEEE CCCCCCCCCCC sss SSSSSSSS rrr RRRRRRRR ****** ***
```

y	年
M	月
d	日
H	時
m	分
C	呼出符号
B	周波数帯
F	周波数
E	変調方式
S	送信符号
R	受信符号
s	送信 RST
r	受信 RST
O	運用者名
N	備考
*	無視

以上の書式は、全ての欄が固定長もしくは備考欄のみが可変長かつ他の欄は固定長の書式として処理される。

また、JARL が 2016 年に改訂したサマリーシートの LOGSHEET 部分にも対応する。+は任意長の反復を表す。

```
(jarl)
yyyy-MM-dd HH:mm B+ E+ C+ s+ S+ r+ R+
```

zLog のバイナリデータも処理できる。冒頭の 256 バイトを無視して、以降は 256 バイト単位で交信を表す。ただし、交信日時は 1899 年 12 月 30 日からの経過日数で表現する。小数点以下はその日の経過時間を表す。

位置	長さ	種別	属性	書式	備考
0	8	item	time	IEEE754	リトルエンディアン
8	13	item	call	ASCII	冒頭は文字列の長さ
21	31	sent	code	ASCII	冒頭は文字列の長さ
52	31	rcvd	code	ASCII	冒頭は文字列の長さ
84	2	sent	rstq	整数値	リトルエンディアン
86	2	rcvd	rstq	整数値	リトルエンディアン
92	1	item	mode	列挙型	CW,SSB,FM,AM,RTTY
93	1	item	band	列挙型	右表参照
94	1	sent	watt	列挙型	P,L,M,H
160	15	item	name	MS932	冒頭は文字列の長さ
175	67	item	note	MS932	冒頭は文字列の長さ

値	周波数	値	周波数
0	1.9MHz	11	430MHz
1	3.5MHz	12	1200MHz
2	7MHz	13	2400MHz
3	10MHz	14	5600MHz
4	14MHz	15	10GHz&up
5	18MHz		
6	21MHz		
7	24MHz		
8	28MHz		
9	50MHz		
10	144MHz		

ATS-4 型は、TableFormat インターフェースを実装した**プラグイン**を通じて、様々な交信記録に対応できる。

package qxsl.table

```
public interface TableFormat {
  public String getName();
  public List<String> getExtensions();
  public String toString();
  public List<Item> decode(InputStream in) throws IOException;
  public void encode(OutputStream out, List<Item> items) throws IOException;
}
```

TableFormat は**サービスプロバイダ**機構によりインスタンス化され、Tables クラスに自動的に登録される。

package qxsl.table

```
public final class Tables implements Iterable<TableFormat> {
  public Tables();
  public Tables(ClassLoader cl);
  public Iterator<TableFormat> iterator();
  public TableFormat getFormat(String name);
  public List<Item> decode(InputStream in) throws IOException;
  public List<Item> decode(URL url) throws IOException;
}
```

交信記録に適合する書式を判別した上で交信記録を読み込む decode メソッドを提供する点は特筆に値する。Tables クラスに自動的に登録するために必要な**プロバイダ構成ファイル**の例は、下記の操作で閲覧できる。

```
$ git clone https://github.com/nextzlog/qxsl
$ vim qxsl/src/main/resources/META-INF/services/qxsl.table.TableFormat
```

なお、Tables クラスはサマリーシートや Cabrillo に未対応である。代わりに Sheets クラスが利用できる。詳細は省くが、Tables に類似した API により、サマリーシートや Cabrillo から交信記録のみを抽出できる。

付録C　規約の定義と得点の計算

ATS-4 型では、交信記録を有効な交信と無効な交信に振り分ける基準を、**ドメイン特化**の LISP で定義する。

```
(set 'ok (lambda (it) (list (list (call it) (band it)) (list (band it) (code it)) 1)))
(contest "CW ONLY CONTEST"
  (section "10MHz" (lambda (item) (if (equal (band item) 10000) (ok item) "invalid")))
  (section "14MHz" (lambda (item) (if (equal (band item) 14000) (ok item) "invalid")))
  (section "18MHz" (lambda (item) (if (equal (band item) 18000) (ok item) "invalid")))
  (section "21MHz" (lambda (item) (if (equal (band item) 21000) (ok item) "invalid")))
  (section "28MHz" (lambda (item) (if (equal (band item) 28000) (ok item) "invalid")))
  (section "50MHz" (lambda (item) (if (equal (band item) 50000) (ok item) "invalid"))))
```

contest と section は、**コンテスト**と**部門**を定義するための独自の関数であり、下記の通りに実装される。

```
 contest  ···(args, eval) -> new Contest((String) eval(args.car), args.cdr.map((Section) eval)
 section  ···(args, eval) -> new Section((String) eval(args.car), (Lambda) eval(args.cdr.car))
```

Contest は、ALLJA1 や全市全郡などのコンテストを表現するクラスで、隷下に複数の Section を保有する。

package qxsl.ruler

```
public abstract class Contest implements Iterable<Section> {
  public Contest();
  public abstract String getName();
  public final String toString();
  public abstract Iterator<Section> iterator();
  public final Section getSection(String name);
  public static final Contest forName(String name);
}
```

Section は、コンテストの部門を表現するクラスで、有効な交信と無効な交信の判別をする機能を提供する。

package qxsl.ruler

```
public abstract class Section {
  public Section();
  public abstract String getName();
  public final String toString();
  public abstract Message validate(Item item) throws Exception;
}
```

定義した規約を読み取るには、Zelkova クラスで S 式を評価して、返り値や変数を Contest に型変換する。

package qxsl.ruler

```
public final class Zelkova extends AbstractScriptEngine {
  public Zelkova();
  public ScriptEngineFactory getFactory();
  public Object eval(Reader r, ScriptContext c) throws ScriptException;
  public Object eval(String s, ScriptContext c) throws ScriptException;
  public Bindings createBindings();
}
```

Zelkova クラスが実装する LISP は、Item クラスや Exch クラスを処理するため、下記の関数が定義される。

```
rcvd  ···(args, eval) -> ((Item) eval(args.car)).getRcvd()
sent  ···(args, eval) -> ((Item) eval(args.car)).getSent()
hour  ···(args, eval) -> ((Item) eval(args.car)).get(Time.class).hour()
call  ···(args, eval) -> ((Item) eval(args.car)).value(Call.class)
band  ···(args, eval) -> ((Item) eval(args.car)).value(Band.class)
mode  ···(args, eval) -> ((Item) eval(args.car)).value(Mode.class)
rstq  ···(args, eval) -> ((Exch) eval(args.car)).value(RSTQ.class)
code  ···(args, eval) -> ((Exch) eval(args.car)).value(Code.class)
city  ···(args, eval) -> new City((String) eval(args.car)).getCityName())
pref  ···(args, eval) -> new City((String) eval(args.car)).getPrefName())
```

他には、変数やリストや文字列を操作する基本的な関数に対応する。また、**無名関数**や**マクロ**にも対応する。

```
(set 'def-lambda (syntax (name pars body) (list 'setq name (list 'lambda pars body))))
(set 'def-syntax (syntax (name pars body) (list 'setq name (list 'syntax pars body))))
```

section 関数の第 2 引数の無名関数の返り値は交信の可否を表し、Message インターフェースで取り出せる。

package qxsl.ruler

```
public interface Message implements Serializable {}
```

無名関数が交信の ID とマルチプライヤの ID と得点の組を返した場合、下記の Success クラスが返される。

package qxsl.ruler

```
public final class Success implements Message {
  public Success(Object call, Object mult, int score);
  public final Object call;
  public final Object mult;
  public final int score;
}
```

また、無名関数が、交信が無効である旨を説明する文字列を返した場合、下記の Failure クラスが返される。

package qxsl.ruler

```
public final class Failure implements Message {
  public Failure(String message);
  public Failure(Exception cause);
  public String getMessage();
  public Exception getCause();
}
```

下記の Summary クラスは、指定された交信記録と部門で得点を計算し、有効な交信と無効な交信を列挙する。

package qxsl.ruler

```
public final class Summary implements Serializable {
  public Summary(List<Item> list, Section sect) throws Exception;
  public int calls();
  public int mults();
  public int score();
  public List<Item> accepted();
  public List<Item> rejected();
}
```