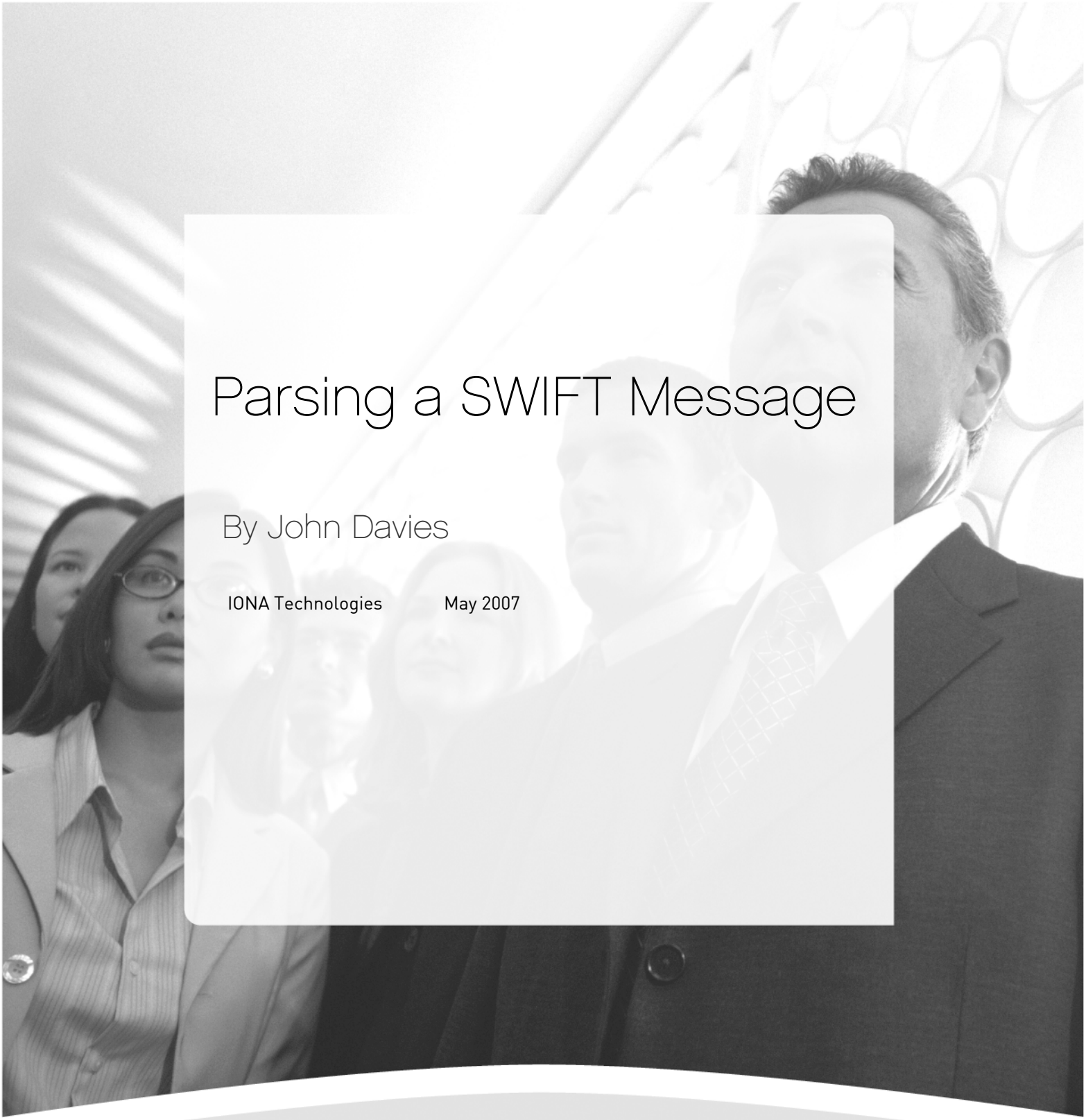

Parsing a SWIFT Message

By John Davies

IONA Technologies May 2007

IONA®

Making Software Work Together™

Where to Begin

As a die-hard programmer of nearly 25 years experience I look at almost everything as a programming challenge. Of the 15 years I've spent as a consultant at least 12 of them have been in wholesale banking, so if someone was to approach me with what looks like a rather old-fashioned text based message and ask if I could parse and validate it my default answer would be a confident "yes, I'll have it done by the end of the week", knowing that in general I could knock it up in an afternoon and then spend the next 4 days perfecting it.

Wrong!

This rather old-fashioned text message is a SWIFT MT103 message. Take a look:

```
{1:F01MIDLGB22AXXX0548034693}{2:I103BKTRUS33XBRDN3}{3:{108:MT103}}{4:
:20:8861198-0706
:23B:CRED
:32A:000612USD5443,99
:33B:USD5443,99
:50K:GIAN ANGELO IMPORTS
NAPLES
:52A:BCITITMM500
:53A:BCITUS33
:54A:IRVTUS3N
:57A:BNPAFRPPGRE
:59:/20041010050500001M02606
KILLY S.A.
GRENOBLE
:70:/RFB/INVOICE 559661
:71A:SHA
-}
```

OK, it's not exactly a tag/value *ini* or *properties* file but it does seem to have a pattern, surely it will only take a day or so? I think not, let us take a closer look.

This message content is made up of 4 parts:

```
{1:F01MIDLGB22AXXX0548034693}
{2:I103BKTRUS33XBRDN3}
{3:{108:MT103}}
{4:
:20:8861198-0706
:23B:CRED
...
-}
```

In fact this complete message is made up of 5 parts (for SWIFT literates the 5th part is added as the message goes out). Let us examine the structure of each message part.

Block 1, Basic Header

Not too complex, you will have to search for a "{1:" and then parse four basic fields and one complex field:

```
Application ID, 1 char (A, F or L only, no other characters are valid)
Service ID, 2 chars (01, 03 or 21 only)
LT Address
Bank Code, 4 upper case chars
Country Code, upper case 2 chars but validated against the bank's internal
list
Location Code
Location Code 1, 1 upper char
Location Code 2, 1 upper char
Logical Terminal Code, 1 upper char
Branch code, 3 upper chars
Session Number, 4 numbers (always 4)
Sequence Number, 6 numbers (again always 6)
```

OK, pretty easy, you could knock up something using regular expression (my personal favorite) or perhaps some sort of serialization, but the quickest would probably be hand-cranked, read N chars and put the value into a variable. Do not, however, forget the validation. You cannot just read a character without checking it, however some of the values are going to be needed later to indicate the types of fields we're going to be expecting. For this reason, regexp is going to help and we can use the expression as part of the validation.

Block 2, Application Header Input

This is much the same stuff as Block 1 but made slightly more complex by the addition of an optional complex field at the end of Block 2. Optional meaning that it might or might not be there, both cases are valid. Well almost true, in some cases the field itself is optional but there are some situations where another field's value dictates whether it should be there or not. In the real nightmare case there can be special combinations of several fields that dictate another's existence. This is one of the last gotchas just when you thought you had finally cracked it after a month's work or so.

Block 3, User Header

Block 3 is actually optional and so are all the fields in it, all 5 of them. Fortunately these five fields are all simple ones, well, simple by SWIFT standards, let us take a look.

- Field 103, "{103:" followed by 3 upper case chars
- Field 113, "{113:" followed by 4 characters from SWIFT's "x" char type (a-z, A-Z, /-?:().,'+)
- Field 108, "{108:" followed by up to 16 chars of type "x" (see above)
- Field 119, "{119:" followed by up to 8 uppercase chars or numbers
- Field 115, "{115:" followed by up to 32 "x type" chars

All fields end with a '}' but there can be more than one of these fields.

By this time if you have chosen regular expressions then they are going to start getting very complex. Another consideration with regexp is that there is no real standard. There are a few options; Java's own in J2SE 1.4 is a good and rich implementation, Apache's is not quite so good and slower. There are two others worth looking at, both are quite fast:

```
http://www.karneim.com/jrexx/
http://www.brics.dk/~amoeller/automaton/
```

JRexx is the closest to Sun's version and has a very interesting GUI add-on. Incidentally if you want to use J2SE 1.3 and still use Sun's RegExp we managed to extract it from 1.4 and recompile it under J2SE 1.3.

Block 4, SWIFT Message Body

In this example, we'll look at an *Incoming* MT103 (Single Customer Transfer Text Block), a long way from being the most complex SWIFT message and probably just average in terms of complexity. Incoming by the way is from SWIFT's point of view, it's actually outgoing from the bank or from your point of view.

There are 24 complex fields in Block 4, 6 are mandatory, 3 are optional but can have multiple instances and the other 15 are optional. Let us take a look at just one of them, field 53a, it looks innocent enough in the following example:

```
:53A:BCITUS33
```

This field alone though, just one of 23 others in this block, could take you the good part of a day to code, so why? Let us have a look at the structure. Firstly, there is a choice of 3 types of 53 in an MT103; A, B or D, i.e. 53A, 53B or 53D, this is the 53A definition:

```
Party line1 (optional)
Debit/Credit Indicator
D/C Indicator (D or C)
Party Identifier (34 "x" type chars)
Party Identifier (34 "x" type chars)
D/C Indicator (D or C)
BIC
Bank Code (4 Upper case chars)
Country Code (2 Upper case chars)
Location Code
Location Code 1 (1 Upper case char)
Location Code 2 (1 Upper case char)
Branch code (Optional 3 Upper case chars)
```

So, there are two more definitions like this in field 53, and another 23 fields some of which are equally complex to handle. Something else worth noting if you were thinking of parsing the message line by line is that a few fields are multi-line. Field 59 for example has an optional *Party Identifier* of 34 *x* type characters proceeded by a '/' and terminated by the end of the line (always CR-LF in SWIFT). This is followed (assuming it was there in the first place) by 1 to 4 lines of up to 35 'x' type characters.

```
:59:/20041010050500001M02606
KILLY S.A.
GRENOBLE
```

Block 5, Trailer

The trailer is an optional block with 6 complex and 2 simple fields. It is more of the same stuff.

I am not a betting man but I would put serious money on the fact that even the brightest of programmers could not write a reliable SWIFT parser for any given message type in under a week. Take an average programmer though and you're looking at several weeks to get it right.

Fine, one of ours might just come close but then they have been writing SWIFT messages for years, even they would need a good spec. And I know for a fact that they would not take me on.

Validation

There is not much I need to say here, just read some of the conditions defined by SWIFT, they speak for themselves.

- C1: If field 33B is present and the currency code is different from the currency code in field 32A, field 36 must be present, otherwise field 36 is not allowed (Error code(s): D75)
- C2: If the country codes of the Sender's and the Receiver's BICs are within the following list:AD, AT, BE, BV, CH, DE, DK, ES, FI, FR, GB, GF, GI, GP, GR, IE, IS, IT, LI, LU, MC, MQ, NL, NO, PM, PT, RE, SE, SJ, SM, TF and VA, then field 33B is mandatory, otherwise field 33B is optional (Error code(s): D49)
- C3: If field 23B contains the code SPRI, field 23E may contain only the codes SDVA, TELB,PHOB, INTC (Error code(s): E01). If field 23B contains one of the codes SSTD or SPAY, field 23E must not be used (Error code(s): E02).
- C10: If field 23B contains the code SPRI, field 56a must not be present (Error code(s): E16).If field 23B contains one of the codes SSTD or SPAY, field 56a may be used with either option A or option C. If option C is used, it must contain a clearing code (Error code(s):E17)
- C13: If any field 23E contains the code CHQB, subfield 1 (Account) in field 59a Beneficiary Customer is not allowed (Error code(s): E18)
- C18: If field 57a is not present, no field 23E may contain TELE or PHON (Error code(s):E45)

There are 19 conditions in an MT103, slightly more than the average but by no means an exception and definitely not the most complex.

Testing

There are 19 conditions so you're going to need at least 19 unique test messages to test your parser. JUnit is a perfect choice but take a read of the conditions above again, you have to now design a test message that not only respects the complex message structure but also breaks only the rule you aim to test. Again we're talking probably over half a day per message, maybe slightly faster once you get into the swing of it but it's going to take you several days if not a week just to write the tests.

OK, it's hard work. What is the alternative?

The classic response: buy it. Writing a parser for a typical SWIFT message is going to take you about 2 weeks for the very basics, even with a good programmer. If he/she quotes less then show them this document first and ask again. If I was working for someone in a bank and they had no choice but to code it in-house, I would schedule about a month per message (pure code no testing) or up to two man-years for something more generic. Even if you just wanted a few fields out of an incoming message and did not need to validate it you will end up spending dozens of days over the following months patching all the options you forgot. Imagine the cost to the business!

These rules took us a number of days to code and get right and we have rather nice tools for writing and testing them. If you have just knocked up a parser then it has now got to support these conditions. Bear in mind that the message itself and the rules will change each year based on normal SWIFT FIN MT Standards Release's, that the current and historical standards may be in production use so need to be maintained, and that the base standards may be specialized or extended based on market or industry best practice conventions, so your solution has got to be quite flexible. And this example is based on a single MT103 - all you need to do is to extend it to the other 240+ messages.

In keeping with eXtreme Programming you should really write the tests before you write the code. Fortunately most of these tests are pure data so you can get a business analyst to knock up the 30+ tests for you while you struggle with the parsing. Perhaps you could have a race.

IONA's Artix Data Services produces pure Java code (supported on 1.3 upwards). You get source code that parses, validates and provides full APIs. We are confident that it will take you months to do it yourself, and that comes with the additional ongoing maintenance obligations. We would suggest your development funds can be more efficiently focused on more productive tasks.

You can download the Artix Data Services SWIFT FIN MT103 based reference implementation with source code usage examples. This is an excellent place to start for a technical review. Evaluations of any additional message types are available upon request.

Artix Data Services is one component in the IONA Artix advanced SOA infrastructure suite, designed to streamline, modernize and lower the operating costs of complex and heterogeneous IT environments. IONA has a proven track record of delivering mission-critical infrastructure, and has built many of the earliest and largest SOAs for Global 2000 customers including Credit Suisse, BellSouth, Raymond James & Associates, Marconi, and Deutsche Post (DHL).

IONA Technologies PLC
The IONA Building
Shelbourne Road
Dublin 4 Ireland
Phone +353 1 637 2000
Fax +353 1 637 2888
Support: support@iona.com
WWW: www.iona.com

IONA Technologies Inc.
200 West Street
Waltham MA 02451
USA
Phone +1 781 902 8000
Fax +1 781 902 8001
Training: training@iona.com

IONA Technologies Japan Ltd
Akasaka Sanchome Building 7/F
3-21-16 Akasaka
Minato-ku Tokyo Japan
Phone +813 3560 5611
Fax +813 3560 5612
Sales: sales@iona.com

Making Software Work Together™