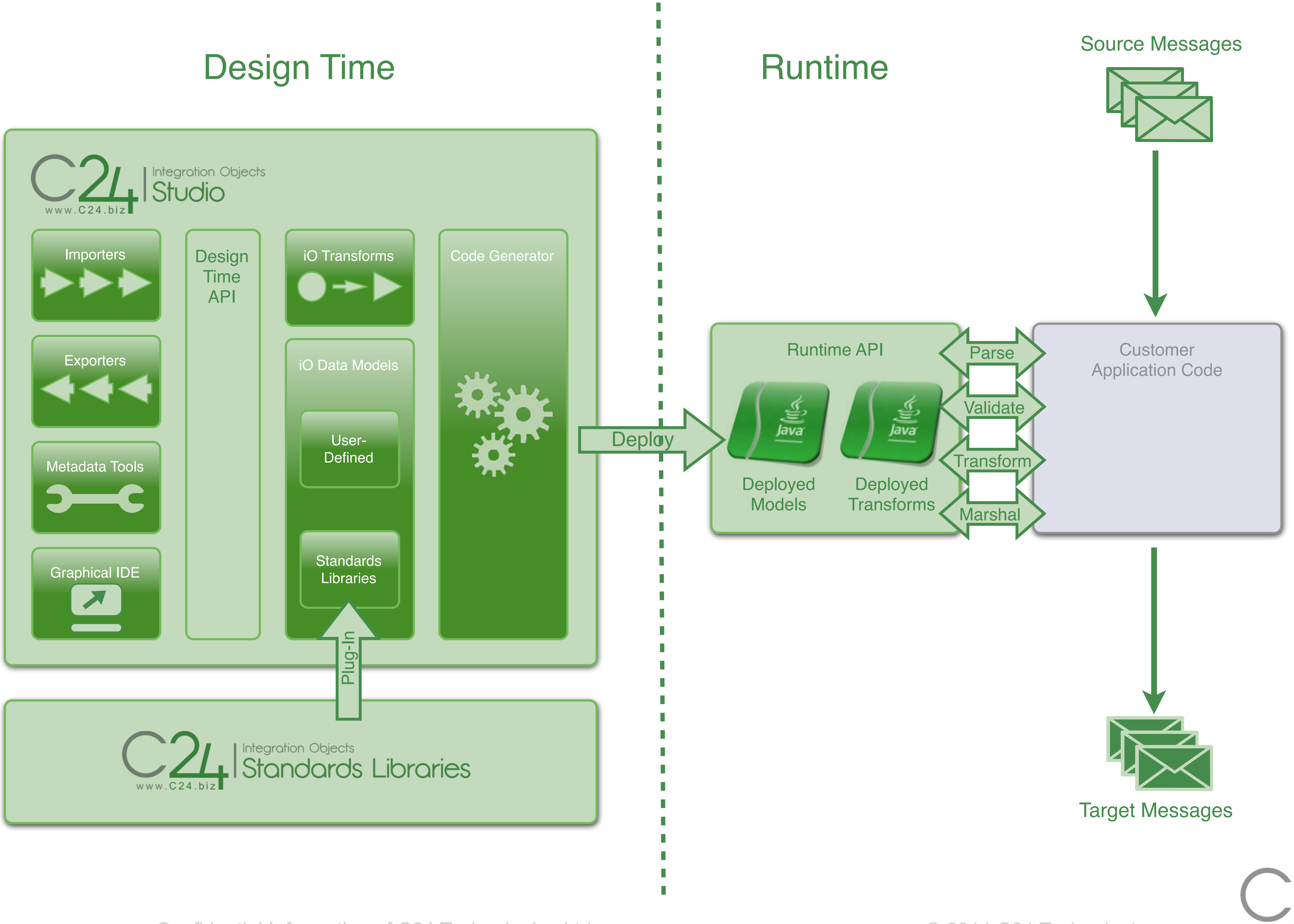
Getting more out of (and specifically IN TO) Oracle Coherence

John Davies | CTO
Steve Miller | Product Director

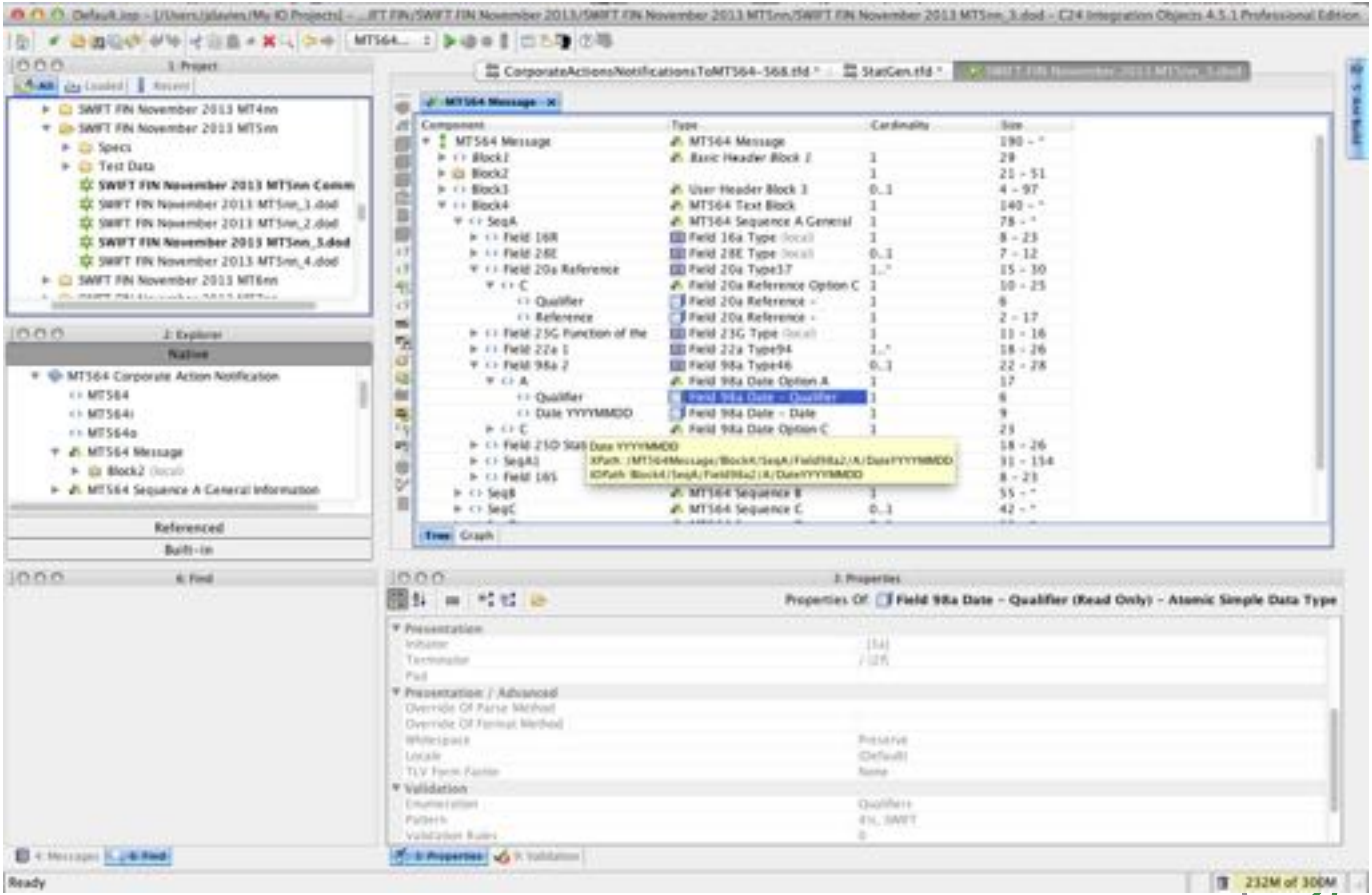
Coherence SIG
Oracle London HQ | 17th July 2014

C24 Integration Objects



Confidential Information of C24 Technologies Ltd.

SWIFT Example (MT564)



Default.iop - [/Users/jdavies/My IO Projects] - .../FT FIN/SWIFT FIN November 2013/SWIFT FIN November 2013 MT5nn/SWIFT FIN November 2013 MT5nn_3.dod - C24 Integration Objects 4.5.1 Professional Edition

1: Project

- SWIFT FIN November 2013 MT4nn
- SWIFT FIN November 2013 MT5nn
  - Specs
  - Test Data
  - **SWIFT FIN November 2013 MT5nn Comm**
  - SWIFT FIN November 2013 MT5nn_1.dod
  - SWIFT FIN November 2013 MT5nn_2.dod
  - **SWIFT FIN November 2013 MT5nn_3.dod**
  - SWIFT FIN November 2013 MT5nn_4.dod
- SWIFT FIN November 2013 MT6nn

2: Explorer

Native

- MT564 Corporate Action Notification
  - MT564
  - MT564i
  - MT564o
  - MT564 Message
    - Block2 (local)
  - MT564 Sequence A General Information

Referenced

Built-in

4: Find

CorporateActionsNotificationsToMT564-568.tfd * | StatGen.tfd * | SWIFT FIN November 2013 MT5nn_3.dod

MT564 Message

Component	Type	Cardinality	Size
MT564 Message	MT564 Message		190 - *
Block1	Basic Header Block 1	1	29
Block2		1	21 - 51
Block3	User Header Block 3	0..1	4 - 97
Block4	MT564 Text Block	1	140 - *
SeqA	MT564 Sequence A General	1	78 - *
Field 16R	Field 16a Type (local)	1	8 - 23
Field 28E	Field 28E Type (local)	0..1	7 - 12
Field 20a Reference	Field 20a Type17	1..*	15 - 30
C	Field 20a Reference Option C	1	10 - 25
Qualifier	Field 20a Reference -	1	6
Reference	Field 20a Reference -	1	2 - 17
Field 23G Function of the	Field 23G Type (local)	1	11 - 16
Field 22a 1	Field 22a Type94	1..*	18 - 26
Field 98a 2	Field 98a Type46	0..1	22 - 28
A	Field 98a Date Option A	1	17
Qualifier	Field 98a Date - Qualifier	1	6
Date YYYYMMDD	Field 98a Date - Date	1	9
C	Field 98a Date Option C	1	23
Field 25D Stat			18 - 26
SeqA1			31 - 134
Field 16S			8 - 23
SeqB	MT564 Sequence B	1	55 - *
SeqC	MT564 Sequence C	0..1	42 - *

Date YYYYMMDD
XPath: /MT564Message/Block4/SeqA/Field98a2/A/DateYYYYMMDD
IOPath: Block4/SeqA/Field98a2/A/DateYYYYMMDD

Tree | Graph

3: Properties

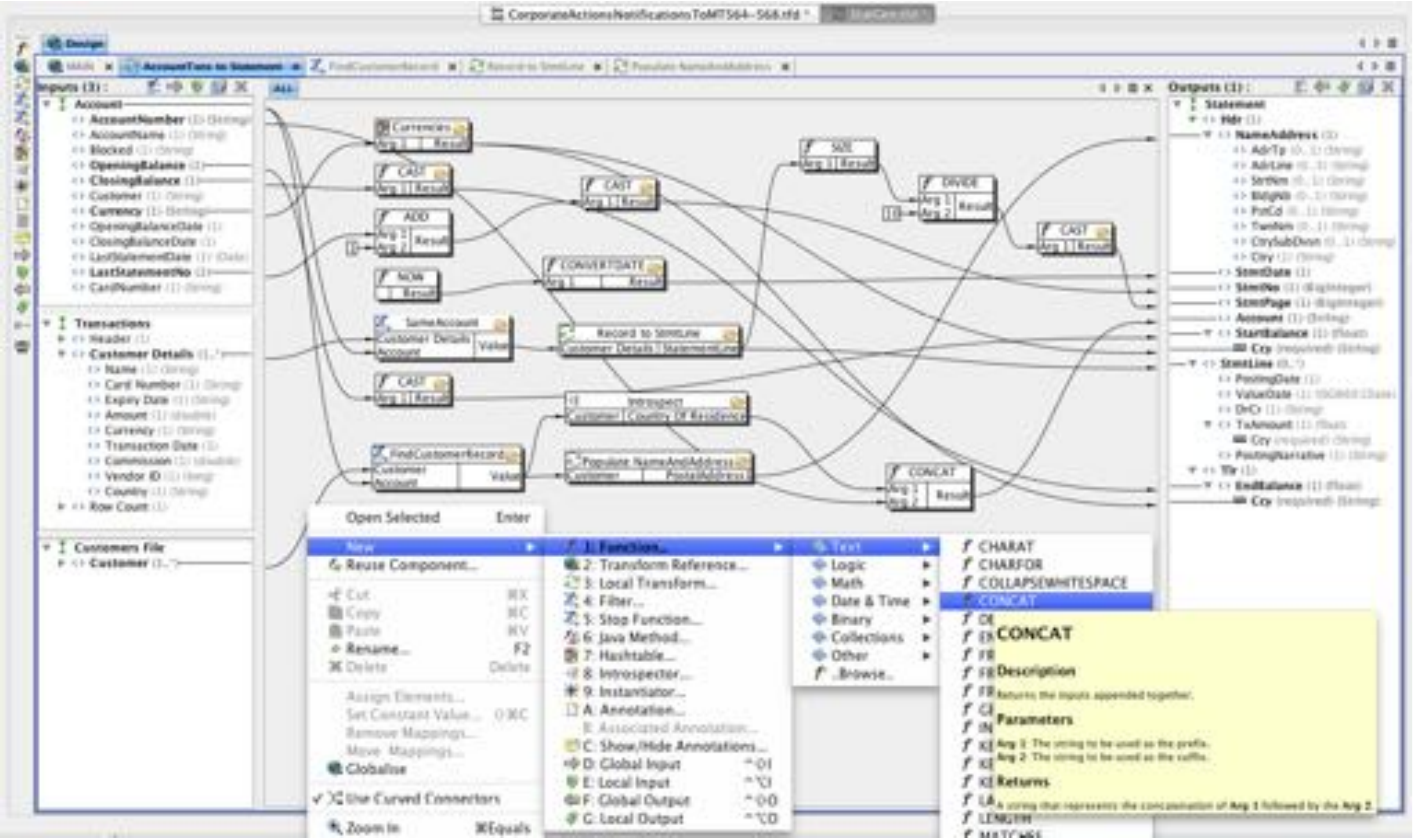
Properties Of: **Field 98a Date - Qualifier (Read Only) - Atomic Simple Data Type**

Presentation	
Initiator	:[1a]
Terminator	/ (2f)
Pad	
Presentation / Advanced	
Override Of Parse Method	
Override Of Format Method	
Whitespace	Preserve
Locale	(Default)
TLV Form Factor	None
Validation	
Enumeration	Qualifiers
Pattern	4!c, SWIFT
Validation Rules	0

4: Messages | 6: Find | 3: Properties | 9: Validation

Ready | 232M of 300M

Transformation Example



CONCAT

Description

Returns the inputs appended together.

Parameters

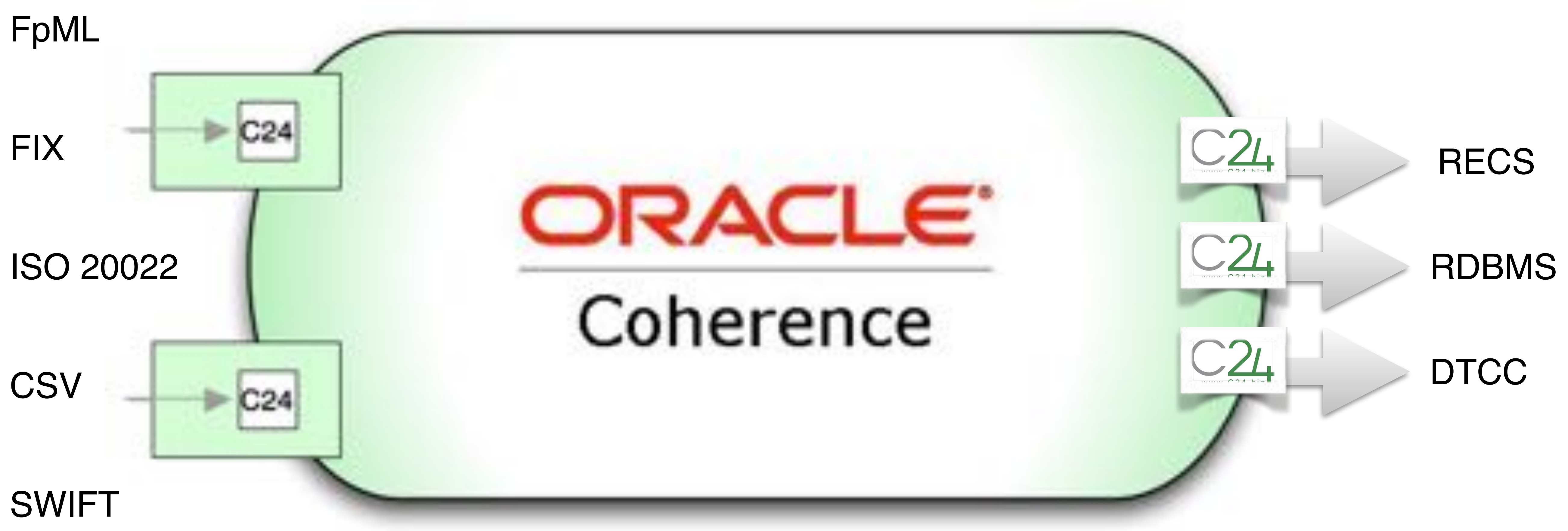
Arg 1: The string to be used as the prefix.
Arg 2: The string to be used as the suffix.

Returns

A string that represents the concatenation of **Arg 1** followed by the **Arg 2**

Where we fit

- Where ever you're dealing with events or messages, C24 can help
- The more complex or scale-critical the better the advantage
  - Proprietary formats, industry standards, legacy interfaces

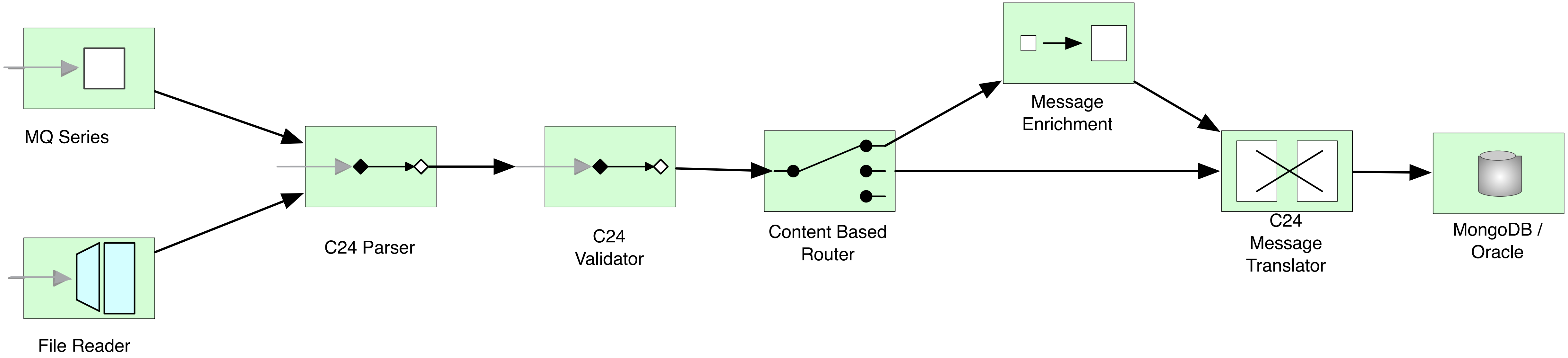


EAI, SOA & ESBs

- C24-iO has deep integration with Spring, Mule, Fuse & Camel etc.

- We generate the Spring config for you so you can use Spring Integration right out of the box

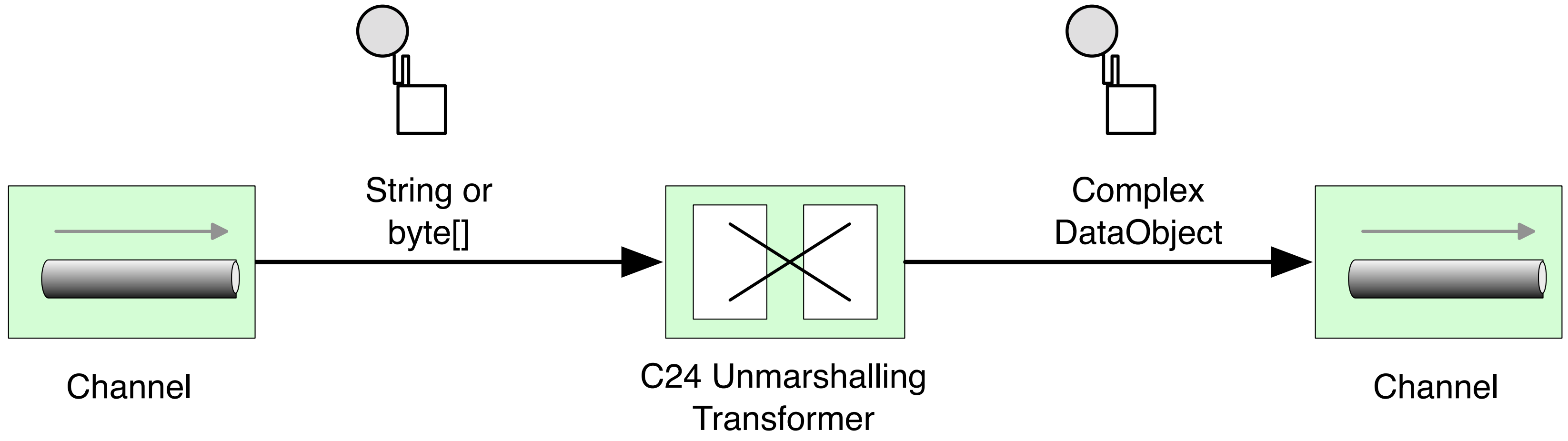
- New performance changes to Spring due out later this summer were driven by C24
  - Spring 4.1 will be able to pre-compile the SpEL expressions



Spring out of the box

- Deploying Fix for example will deploy the config for the parser and model...

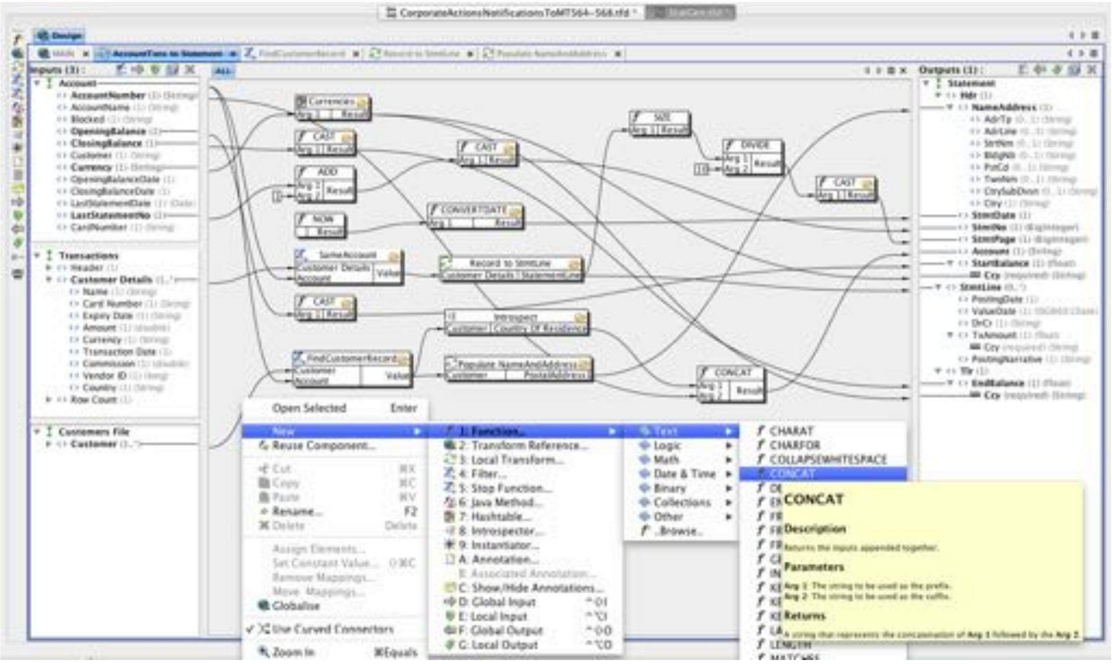
```
<int-c24:unmarshalling-transformer
   model-ref="fixModel"
   source-factory-ref="sourceFactory"
   input-channel="…" output-channel="…" />

<bean id="sourceFactory"
   class="biz.c24...source.FixSourceFactory">
  <property name="encoding" value="UTF-8" />
</bean>
```



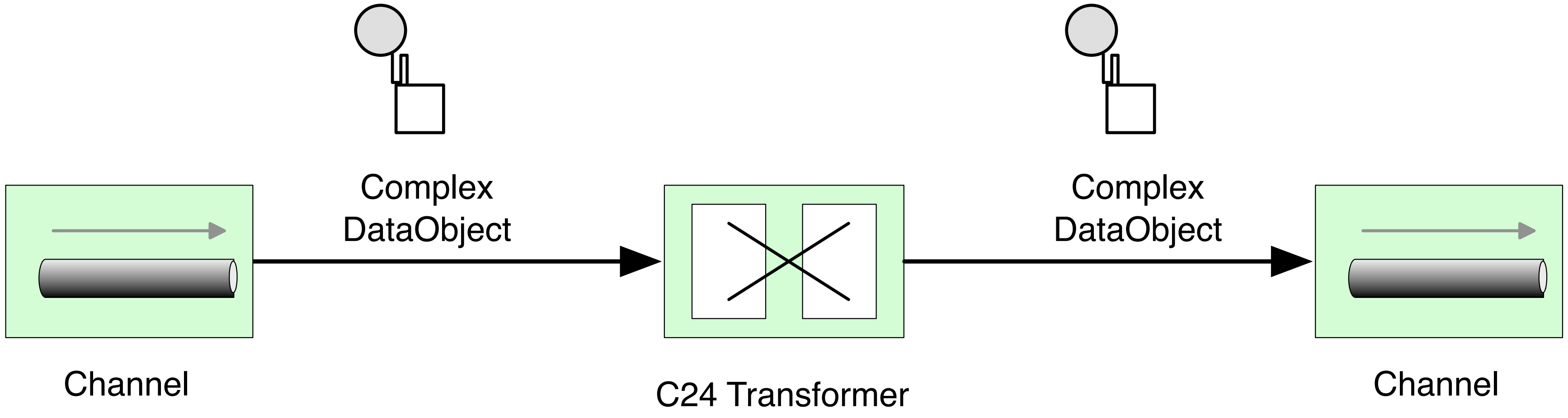
Transformation in Spring

- Real X-to-Y transforms are modelled in the Studio

- The generated transform slots into SI by just specifying the generated class



```
<int-c24:transformer
   transform-class="biz.c24...basic.ExampleTransform"
   input-channel="…" output-channel="…" />
```



Java 8 & Scala

- Change the deploy option to Java 8 and…

```
List<Atom> elements = C24
           .parse(biz.c24.periodic.PeriodicDocumentRoot.class)
           .from(new File("resources/periodic.xml"));

elements.sort((a1, a2) -> Integer.compare(a1.getAtomicNumber(), a2.getAtomicNumber()));

// List all the elements that are solid at room temperature (20C) and boil below 600 deg C
System.out.println("\nSolid at 20°C but boil lower than 800°C...");
elements.stream()
       .filter( a -> a.getMeltingPoint() != null && a.getBoilingPoint() != null )
       .filter(a -> a.getMeltingPoint().getValue() > 293 & a.getBoilingPoint().getValue() < 1073)
       .map(Atom::getNameElement)
       .forEach(System.out::println);
```

- And it works nicely in Scala too…

```
var parser = C24.parse(classOf[CustomersFile]) as C24.Format.XML
var transform = new GenerateContactListTransform
var writer = C24.write() as C24.Format.JSON

new File("/Customers.xml") -> parser -> transform -> writer -> System.out
```

Virtual Getters and Headers

- The XML and therefore generated Java API is usually technically formatted, i.e. it’s not easy to extract key business data
  - tradeDate, buySideCurrency, settlementData are all hidden in the message

- For this reason most architectures use a message wrapper and extract the key fields into a header (header enrichment)
  - Even canonicalised messages present the same issue, key business fields are difficult to find

- Using C24 virtual methods…
  - No need for extra message wrappers, no extra memory used
  - Vastly simplifies user-code & maintenance
  - Can be used with ESB/SOA & messaging for filtering & routing etc.
  - Can be used with in-memory cache and database queries/QL etc.
  - Extremely powerful with Spring Integration/Mule, Coherence etc.

Virtual Getters

- We can now generate virtual getters, i.e. getters for fields that don’t necessarily represent a real field in the model

```
tradeConfirmed.getTrade().getTradeHeader().getTradeDate().getValue().toDate();
```

- We can now use the much simpler…

```
Date tradeDate = tradeConfirmed.getTradeDate();
```

- Instead of this…

```
MT541SequenceE3Amounts[] seqE3 = mt541Message.getBlock4().getSeqE().getSeqE3();
for (MT541SequenceE3Amounts e3Amount : seqE3) {
  for (Field19aType31 field19 : e3Amount.getField19aAmount()) {
    if (field19.getA().getQualifier().compareTo("SETT") == 0) {
      CurrencyAmount currencyAmount = field19.getA().getSignedCurrencyAmount().getCurrencyAmount();
    }
  }
}
```

- We can use the much simpler…

```
CurrencyAmount getSettlementCurrencyAmount();
```

Lambdas in FS

- Using virtual methods with lambdas we can further simplify the code

```
BigDecimal sum = transactions.stream()
    .filter( t -> t.getBuySideCity().equals(“London”) )
    .filter( t -> t.getBuySideCurrency().equals(“GBP”) )
    .map( t -> t.getBuySideAmount() )
    .sum();
```

- This now works across every version of message format and even different message formats
  - FpML
  - FIX
  - ISO 20022
  - Internal canonical format
  - CSV

Compaction

- Everything too large? Why not compress it?
  - It’s slow to compress - takes up CPU cycles
  - It’s slow to decompress - takes up more CPU cycles
  - The compressed data is relatively useless until it’s decompressed
  - Compressing batches is more efficient but you then have to decompress the entire batch too - More CPU cycles again
- Compaction
  - Smaller size but de-compaction is almost free - in some cases better
  - Works at the field level so we can use the data in its compact form
  - Compaction can use many of the features of compression
- Take a trade value… GBP 12,500,000.00
  - We might want to search on GBP values over 10 million
  - With compaction we can do that, compression we need to decompress first

Java binding and memory

- Typically Java Binding tools, like JAXB, JiBX and C24 create Java that looks like the data source
- While this is very convenient for the programmers it creates a lot of Java objects, this slowly consumes memory
- A typical FpML trade is around 8k in size, bind it to Java and it increases to around 25k
- 1 million FpML message in memory is going to cost anything from 8 to 25GB of RAM, add (HA) high availability and we hit 50GB
  - Expensive!
  - In-memory is still fast but 25k message over the network is very slow
  - And 25GB of data over a network or onto disk, even SSD is slow

SDOs

- SDOs or Simple Data Objects are basically Java Binding into a compact binary codec - From any XML format to binary
- We analyse the data model (or XML schema) not just the instance data so can do things like…
  - Reducing the 7 days of the week to just 3 bits
  - Commonly used Strings become lookups into a static table (1 or 2 bytes)
  - Currencies for example only need 1 byte
  - Date/Time with timezone can be stored in 6 bytes
- Bit-fields are compacted resulting in excellent compaction-ratios
  - Getters calculate the offset on the fly, mask and shift the data and return it
- There is NO change to the getter API between standard binding and SDOs

Standard Java Binding

```
<resetFrequency>
    <periodMultiplier>6</periodMultiplier>
    <period>M</period>
</resetFrequency>
```

- JAXB, JIBX, Castor and standard C24 generate something like …

```
public class ResetFrequency {
    private BigInteger periodMultiplier; // Positive Integer
    private Object period;               // Enum of D, W, M, Q, Y

    public BigInteger getPeriodMultiplier() {
        return this.periodMultiplier;
    }
    // constructors & other getters and setters
```

- In memory - 3 objects - at least 144 bytes
  - The parent, a positive integer and an enumeration for Period
  - 3 Java objects at 48 bytes is 144 bytes and it becomes fragmented in memory

Java Binding with SDOs

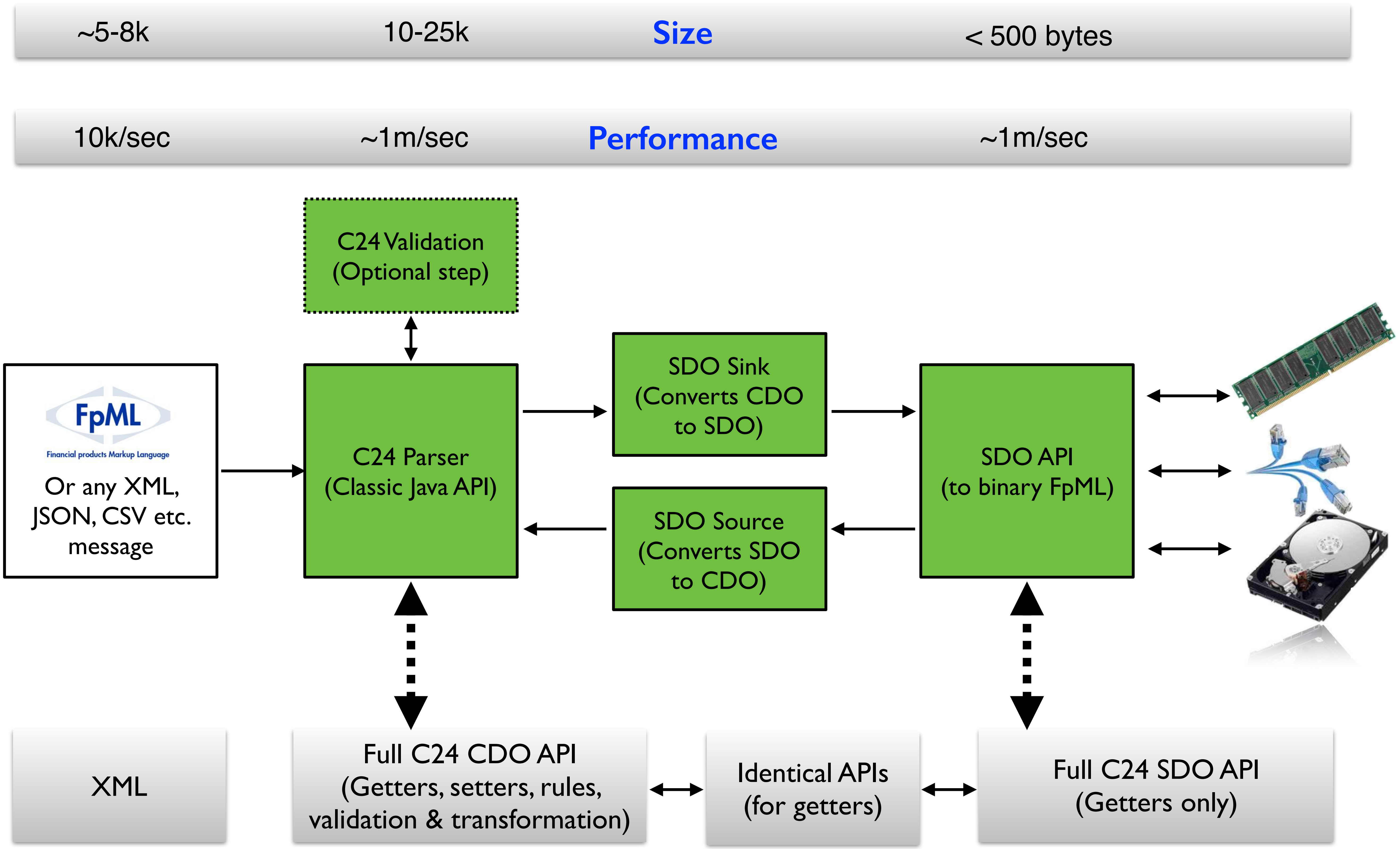
```
<resetFrequency>
    <periodMultiplier>6</periodMultiplier>
    <period>M</period>
</resetFrequency>
```

- Using C24 SDO binary codec we generate …

```
ByteBuffer data;    // From the root object

public BigInteger getPeriodMultiplier() {
        int byteOffset = 123;  // Actually a lot more complex
        return BigInteger.valueOf( data.get(byteOffset) & 0x1F );
    }
    // constructors & other getters
```

- In memory - 1 byte for all three fields
  - The root contains one ByteBuffer which is a wrapper for byte[]
  - The getters use bit-fields, Period is just 3 bits for values D, W, M, Q or Y

How it works



~5-8k

10-25k

Size

< 500 bytes

10k/sec

~1m/sec

Performance

~1m/sec

C24 Validation
(Optional step)

FpML
Financial products Markup Language
Or any XML, JSON, CSV etc. message

C24 Parser
(Classic Java API)

SDO Sink
(Converts CDO to SDO)

SDO Source
(Converts SDO to CDO)

SDO API
(to binary FpML)

XML

Full C24 CDO API
(Getters, setters, rules, validation & transformation)

Identical APIs
(for getters)

Full C24 SDO API
(Getters only)

SDOs - So far*

- ISDA's sample Interest Rate Derivative (vanilla swap) is 7.4k
  - We randomised a few fields and created a few million for testing

- Zipped they are average 1,547 bytes
  - 1 million on disk require 1.5GB and takes 200 seconds to read/decompress
  - Parsing at 20k/sec would add another 50 seconds and need a **lot** of memory

- In memory they are roughly 25k in size (in roughly 400 objects)
  - It was difficult to fit 400k into 10GB of RAM - Lots of full GCs too

- With SDOs the average size was just 442 bytes
  - It took 9 seconds to read and parse 1 million from disk (SSD)
  - It took 415ms to search through all 1 million IRSs in memory (brute force)
  - 20 million fully parsed IRSs comfortably fit in 10GB of RAM

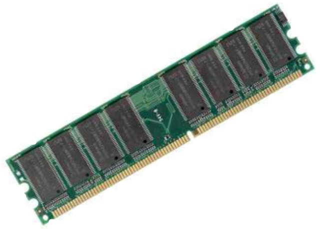
- Total saving on memory with FpML is roughly 50 times

* Tests were run on Java 1.7.0_55 on a MacBook Pro (2.7 GHz Intel i7) on a single core, we continue to improve these figures

SDO selling points

A 5 year leap into the future with Moore's law

- In a nutshell we can compact data by typically over 10 times
- You can get at least 10 times more data into Coherence
- Data takes up a 10th of its usual size
  - On disk or in memory
- Better use of network, memory and disk
  - Massive savings in infrastructure!



C24

www.C24.biz

Info@C24.biz
@C24io

John.Davies@C24.biz
@jtdavies

SDO landing page: http://sdo.c24.biz
http://ref.c24.biz/whitepapers/C24-SDOs-and-Coherence.pdf
http://ref.c24.biz/whitepapers/C24-SDOs-Big-Data-In-Memory.pdf