Available online www.jsaer.com

Journal of Scientific and Engineering Research, 2023, 10(10):115-118

Research Article

ISSN: 2394-2630
CODEN(USA): JSERBR

File Based Extract Configuration Management in Global Payment Applications

Siva Sathyanarayana Movva

Email: sivasathya@gmail.com

Abstract Global Payments or Transactions happen in the form of messages for cash transfers, securities, metals, and various other needs generating millions of data daily and totaling up to a billion in a year. Along with the central applications handling the msg transfer end to end numerous other applications are needed to serve the purpose of reconciliation, business intelligence, tracking of payments and other customer specific requirements. The messages comprise of different fields and here we explore the ways of handling the extract configuration of these fields and the advantages in introducing the file-based configuration management.

Keywords *payments, transactions, messages, fields, configuration, extract*

1. Introduction

There are numerous applications apart from the core ones handling the global cash payments and other transactions. All these applications are built for achieving various goals business intelligence, providing data for billing & traffic analysis, backup and create recovery data for disaster recoveries and offline inquiries, validation of intercontinental and intra-continental traffic, perform validation and reconciliation of traffic, on-demand message extraction & statistics, message compliance and data modelling, create condensed version of sales force traffic, data analysis of message usage & different parts of the message, near real time day & time tracking of message trajectory, report of KYC requirements to prevent money laundering, back-up of undelivered messages for disaster recovery, extraction of detailed message information for customer view and other customer specific requirements.

Earlier Way of Handling Extract Configuration:

In all the previous implementations, the extract configuration was embedded in the code itself. Hence any change in the configuration leads to a change in the code there by leading to new release cycle and associated time and monetary costs.

Example of Prototype for Extract Configuration inside the application Algo:

```
extractalgo.cpp: define type EXTRACT_MSGTYPES[] = { AAA, ABA, ACA, DDD, BBB, CCC, … }
extractalgo.cpp:
        if (MT == XXX) then
       case EXTRACT_FLD_AA:
       case EXTRACT_FLD_BB:
       case EXTRACT_FLD_CC:
```

A configuration change can also arise when one of the mentioned tags AA, BB or CC can always be ceased to be extracted or a new tag DD got to be extracted from then on. Any such change can be implemented only by doing a change in the algorithm and there by dealing with the software release cycles of coding, testing, pre-production and final deployment.

Issues with the older way of config transition:

When extract configuration is embedded in the code, any change in the config the code has to be changed and for the code change to get applied entire release cycle has to undergo and that adds up to tremendous project time and cost.

Hence a new way of handling this config change is designed and can be termed this as “File Based Extract Configuration Management” and the name coined from its design and implementation. The first file with configuration details at the time of initial design of the project/product/module can be named say as ExtractConfigFile.V1 and subsequent changes in configuration is implemented with introduction of new configuration files say as ExtractConfigFile.V2, ExtractConfigFile.V3 and so on incrementing the N value in ExtractConfigurationFile.VN.

ExtractConfigurationFile.V1 ExtractConfigurationFile.V2 ExtractConfigurationFile.V3 … ExtractConfigurationFile.VN

Recovery Time Derivation of Configuration File Name and Configuration File Path:

The software implementation is designed such that the configuration file is loaded at the time of application bring-up as already discussed above. The relevant configuration file for the current RUN is designated by the N value which is stored as an integer or real number in a Globally accessible Param. In the software implementation before the context of loading the configuration file, the configuration file path is formed by adding up the sub-strings of File System path of the Operating System on which the application runs, the sub-string "/ExtractConfigurationFile.V " and the current N value loaded to a local from the Globally Accessible Param. Hence as the design warrants due care got to be taken so that any changes in the message configurations to be extracted have to be captured in a latest extract configuration file with an incremental N value and the N value got to be loaded into the Globally accessible Param.

Implementation prototype for Extract Configuration File Name and Path:

NValueLocal << Global Param Holding N Value;Extract configuration File Path = OS File System Path + "/ExtractConfigurationFile.V " + NValueLocal

Content Structure in the Configuration File:

```
MSGTYPE XXX
    BLOCK A
        TAG CCC
            MANDATORY Y
        TAG DDD
            MANDATORY Y
    BLOCK B
        TAG :DE:
        TAG :FG:
        TAG :GH:
            MULTILINE Y
MSGTYPE ZZZ
    BLOCK A
     TAG CCC
            MANDATORY Y
        TAG DDD
            MANDATORY Y
    BLOCK B
        TAG :DE:
            MANDATORY Y
        TAG :FG:
                MANDATORY Y
            TAG :GHS:
                MANDATORY Y
                MULTILINE Y
            TAG :SR:
                MANDATORY Y
                MULTILINE Y

MSGTYPE WWW
MSGSUBTYPE TTT
    BLOCK A
        TAG AAA
        TAG BBB
        TAG CCC
        TAG DDD
            MANDATORY Y
    BLOCK B
        TAG :AA:
            MANDATORY Y
        TAG :BB:
            MANDATORY Y
        TAG :CCA:
            MANDATORY Y
        TAG :DDB:
        TAG :TOA:
            MULTILINE Y
            MASKREQUIRED Y
            MANDATORY Y
        TAG :TOK:
            MULTILINE Y
            MASKREQUIRED Y
            MANDATORY Y
        TAG :TOF:
            MULTILINE Y
            MASKREQUIRED Y
            MANDATORY Y
```

Working Mechanism with the above configuration file:

Explanation of different names/nouns used in the Sample format file above.

MSGTYPE – There are different types of messages serving different purposes in global transactions. like handling customer to customer cash payments / transactions, Types handling/acting as covers on the done payments, types handling interactions. between bank/individual/institutional customer and the tracking system.

BLOCK -- Blocks are sub-parts in a message of any message Type. Basic Header block, Application Header Block, User Header Block, Text Block, Trailer and Z block information.

TAG -- TAGs are sub parts in a block and each handling different inputs like Name of the Customer, bank, branch, account, even customer address and other details.

MULTILINE – Indicates if a Tag/field is single line or multi-line, say a bank name is single line but Customer Address is definitely multi-line.

MANDATORY -- Indicates tags or fields to be mandatorily present and mandatorily there by extracted too.

SUB TYPE – This field is for different sub-types in a particular message type.

The provision of above fields/tags and the explanation provided for each of them itself describes the strategy behind the design of the file, where for each message type there is an entry and in it there are entries for sub-types, the fields which are mandatory, the ones which are spread across multiple lines versus single lines.

Conclusion

To Quantify the Cost Analysis from earlier methodology to current methodology in a year, let us take the number of config changes per year on an average taking last 3 years into consideration. In the last 3 years there were 11 config files released and hence in a year at least 3 config files per year, which in a way avoided 3 code release cycles. Over all 150 man hours and equivalent cost if it saved because of this new adoption.

References

[1]. Berg, Formal Methods of Program Verification and Specification, Prentice Hall, 1982.
[2]. B. W. Boehm, "A Spiral Model of Software Development and Enhancement", *IEEE Computer*, pp. 61-72, May 1988.
[3]. G. Booch, Object-Oriented Design with Applications, Benjamin Cummings, 1991.
[4]. P. Coad and E. Yourdon, Object-Oriented Analysis, Prentice Hall, 1990.
[5]. B. W. Kernighan and D. M. Ritchie, The C Programming Language, Prentice Hall, 1991.
[6]. S. R. Ladd, Turbo C++ Techniques and Applications, M T Books, 1990.
[7]. S. B. Lippman, C+ + Primer, Addison Wesley, 1991.
[8]. P. J. Lukas, The C+ + Programmers Handbook, Prentice Hall, 1992.
[9]. B. Meyer, Object-Oriented Software Construction, Prentice Hall, 1988.
[10]. R. S. Pressman, Software Engineering - A Practitioners Approach, McGraw-Hill, 1987.
[11]. R. R. Seban, *A Temporal Logic for Proofs of Correctness of Distributed Protocols*, March 1993.
[12]. R. R. Seban, *An Introduction to Object-Oriented Design with C++*, December 1992.
[13]. I. Sommerville, Software Engineering, Addison Wesley, 1992.
[14]. B. Stroustrup, The C++ Programming Language, Addison Wesley, 1989.
[15]. R. H. Thayer, "System and Software Requirements Engineering", *IEEE Tutorial*, 1990.