메타데이터

1. 개요
2. Data를 정의하는 data
3. 종류는 많지만 목적과 형태가 대동소이하기 때문에 이것저것 볼 필요는 없고 문자열 기반의 XML과 native기반의 JSON(http://json.org/json-ko.html)만 참조해도 충분
4. 잘 만들어진 메타데이터는 어플 장르, 자료구조, 자료형에 독립적이므로 사골까지 우려먹을 수 있음
5. 목적
6. 메타데이터의 공통적 특징 파악
7. 문자열 기반과 native 기반의 차이점 파악
8. 구조 설계
9. 샘플을 통한 응용법 확인
10. 메타데이터의 공통적 특징 파악
11. Tree, 혹은 graph 형태의 node 구조를 가질 것
12. Key-value pair 형태를 가질 것
13. 일반적인 자료형(정수, 실수, 스위치, 문자열 등)을 단항, 혹은 배열 형태로 격납 가능 할 것
14. Text, binary, 기타 포맷간 누수되는 데이터 없이 상호 변환이 가능 할 것
15. 데이터로 무엇이 들어갈지 모르므로 유니코드 기반 필수
16. 문자열 기반 격납

1> 문자열을 파싱해 key-value 모두 문자열 형태로 저장

2> 구현비용 낮고 가벼움. 사실상 메타데이터의 역할은 파싱으로 토큰화 하는 것이 전부

3> 꺼내 사용하는 측에서 원하는 형태로 변환하는 과정 필요하고 메모리 낭비 큼

1. Native 기반 격납

1> source단에서부터 자료형 선언, 파싱시 native 형태로 변환해 격납

2> 자료형 처리가 추가되므로 구현비용 크지만 메모리를 아낄 수 있고 꺼내 쓰는 비용이 적음

3> 지원 자료형(primitive data type)에 대한 정의 필요

1. 구조 설계

1> 모든 key는 문자열, 배열 인덱싱은 정수 사용

2> 문자열은 큰 따옴표 사이의 값들로 정의

3> 태그 구분은 공문자

3> 모든 선언의 기본 형태는, [종류] - [이름] – [몸체]로 함

4> 노드

* Node “노드 이름” { }
* 종류 선언은 “Node”, 이름은 문자열, 몸체는 “{“, “}” 사이로 함

5> 값

* 익숙한 C style 선언을 따름
* 자료형은 8가지 타입 지원(bool, int, uint, int2, float, float2(vector2) , float3(vector3), string)
* 문자열의 경우 탈출문자(\t, \n, \” 등) 지원
* 몸체는 “=” 부터 “;” 사이로 함

Ex> bool ok = Yes;

Ex> int mp = -1;

Ex> uint mp = 100;

Ex> int2 position = (10, 200);

Ex> float length = 100.123;

Ex> vec2 winSize = (10.5, 20.8);

Ex> vec3 rot = (10.5, 20.8, -1);

Ex> str desc = “제가 좀 잘생겨서 죄송요.\n\t 데헷~”;

* 배열의 경우 “/” seperator로 구분

Ex> uint mp = 100 / 200 / 300 / 400;

Ex> vec2 winPosSize = (-10.5, -20.8) / (110, 200);

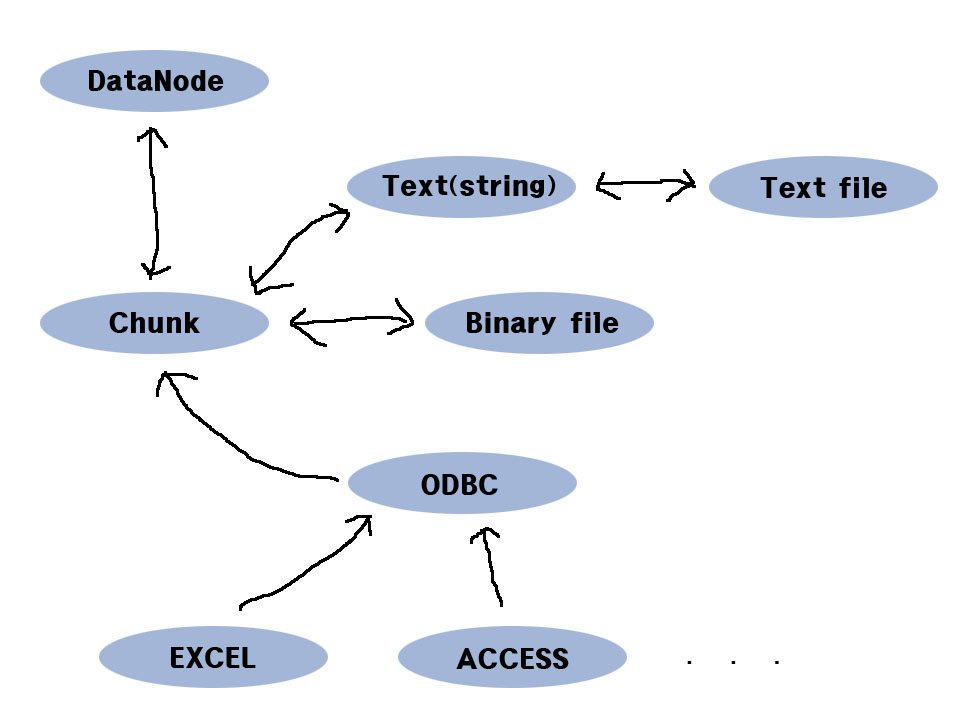
Ex> str tag = “오호” / “으헷” / “으히히^^”;

Ex>

|  |
| --- |
| int IQ = 65;  str nameOfHero = "미모와기품";  float 바보 = 1234.123;  Node "MonsterTable"  {  Node "Orc"  {  str KoreanName = "오꾸";  str EnglishName = "Orc";  int intelligence = 10;  }  Node "Knight"  {  str KoreanName = "얍샵이";  str EnglishName = "Knight";  int intelligence = 15;  uint threeSize = 40 / 30 / 40;  }  Node "You"  {  str KoreanName = "알면서";  str EnglishName = "Legend of beauty";  uint threeSize = 82 / 66 / 93;  Node "특징"  {  str Desc = "그런거 없음" / “묻지마라”;  uint hiddenThreeSize = 79 / 76 / 101;  }  }  } |

6> 포맷

* 최종 격납된 컨테이너를 DataNode라고 할 때, Text, binary(chunk), etc(EXCEL, ACCESS, …) 포맷과의 상호 전환 구조 정의



* Chunk(memory)를 중심으로 다른 포맷과의 호환성 유지
* DataNode는 chunk를 상대로만 in/out
* 타 포맷을 사용할때는 chunk를 상대로 하는 parser만 존재하면 됨
* DataNode에서 읽어들일 때는 포맷이나 확장자 상관 없이 적합한 문법으로 구성되어 있으면 모두 읽을 수 있어야 함

7. 샘플을 통한 응용법 확인