
Subject: Json serialization support

Posted by [Mindtraveller](#) on Sat, 04 Feb 2012 20:15:58 GMT

[View Forum Message](#) <> [Reply to Message](#)

Since Mirek announced "native" JSON support in U++, I tried to use it for serializing my structures and found it to be relatively hard to implement. For example, loading not-very-complex JSON led me to following ugly code:

```
for (int projectI=0; projectI<versions.GetCount(); ++projectI)
{
    if (!IsValueMap(versions[projectI]))
        continue;
```

```
    ValueMap curProject(versions[projectI]);
```

```
    ValueMap curProjectGlobal;
```

```
    if (IsValueMap(curProject[KEY_PROJECT_GLOBAL]))
        curProjectGlobal = curProject[KEY_PROJECT_GLOBAL];
    if (!IsDate(curProjectGlobal[KEY_PROJECT_GLOBAL_DATE]))
        curProjectGlobal.Set(KEY_PROJECT_GLOBAL_DATE, GetSysDate());
    if (!IsNumber(curProjectGlobal[KEY_PROJECT_GLOBAL_COUNTER]))
        curProjectGlobal.Set(KEY_PROJECT_GLOBAL_COUNTER, 0);
    curProject.Set(KEY_PROJECT_GLOBAL, curProjectGlobal);
```

```
    ValueMap curProjectVersion;
```

```
    if (IsValueMap(curProject[KEY_PROJECT_VERSION]))
        curProjectVersion = curProject[KEY_PROJECT_VERSION];
    if (!IsDate(curProjectVersion[KEY_PROJECT_VERSION_DATE]))
        curProjectVersion.Set(KEY_PROJECT_VERSION_DATE, GetSysDate());
    if (!IsNumber(curProjectVersion[KEY_PROJECT_VERSION_COUNTER]))
        curProjectVersion.Set(KEY_PROJECT_VERSION_COUNTER, 0);
    ValueMap curProjectVersionCurrent;
    if (IsValueMap(curProjectVersion[KEY_PROJECT_VERSION_CURRENT]))
        curProjectVersionCurrent = curProjectVersion[KEY_PROJECT_VERSION_CURRENT];
    curProjectVersion.Set(KEY_PROJECT_VERSION_CURRENT, curProjectVersionCurrent);
```

```
    curProject.Set(KEY_PROJECT_VERSION, curProjectVersion);
```

```
    versions.SetAt(projectI, curProject);
}
```

Maybe ValueMap and ValueArray classes are good for other tasks, anyway.

So I thought about little extending JSON support in U++ making it the same as serialization with Stream and Xml - as IMO it was a brilliant solution to make it pure and clear inside a single member function (I mean Serialize and Xmlize).

After a day of work I've made a number of helper classes based on CParser (and inspired with Mirek's JSON parsing functions).

Finally, I've come to what I wanted:

```
struct TestStruct2
{
    int c;
    bool d;
    double e;
    Time t;

    void Jsonize(JsonIO &json)
    {
        json
            ("c", c)
            ("d", d)
            ("e", e)
            ("t", t)
        ;
    }
};

struct TestStruct
{
    String u;
    String a;
    int b;
    TestStruct2 c;
    VectorMap<String, Vector<int> > map1;

    void Jsonize(JsonIO &json)
    {
        json
            ("a",a)
            ("b",b)
            ("c",c)
            ("u",u)
            ("map1",map1)
        ;
    }
};
```

As you can see, JSON serialization is used the common way. Also it natively supports VectorMap/ArrayMap and Vector/Array serialization.

The code was not widely tested and not all the types are supported, but this is the beginning.

Also the efficiency of different serialization types was tested.

JSON/XML/BINARY timing:

release: 125/265/16

debug: 842/1857/125

Maybe, not that bad for 1-day code which was not optimized at all.

So here are sources with Jsonize package. If it is compiled as a main package (not as dependency), the test binary is made.

Any comments, critics and suggestions are welcome.

File Attachments

1) [Jsonize.zip](#), downloaded 304 times
