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 280 times

Subject: Re: Json serialization support
Posted by mirek on Sun, 05 Feb 2012 17:42:57 GMT
View Forum Message <> Reply to Message

Yes, I was thinking about adding this too..

I was slow to do so because I am playing with idea that this perhaps could be somehow 'generalized', e.g. you should be able to produce / consume XML this way as well (some sort of limited XML without attributes) or anything that resembles JSON array/object/string/number model.

In any case, thanks, this will speed things up. Going to check your code now

Mirek

Subject: Re: Json serialization support
Posted by mirek on Sun, 05 Feb 2012 19:44:10 GMT
View Forum Message <> Reply to Message

I think your code might be a bit overengineered, duplicating the JSON parser and constructing a new tree. I think it might be better and more general to simply use Value as tree:

```
#include <Core/Core.h>

using namespace Upp;

struct JsonIO {
 Value& node;
 bool   loading;

 bool IsLoading() const { return loading; }

 template <class T>
```

```
 JsonIO& operator()(const char *key, T& value);

 JsonIO(Value& node, bool loading) : node(node), loading(loading) {}
 JsonIO(const Value& node) : node(const_cast<Value&>(node)), loading(true) {}
};

template <class T>
void Jsonize(JsonIO io, T& var)
{
 var.Jsonize(io);
}

template <class T>
JsonIO& JsonIO::operator()(const char *key, T& value)
{
 if(IsLoading()) {
  const Value& v = node[key];
  if(!v.IsVoid())
   Jsonize(JsonIO(const_cast<Value&>(node[key]), true), value);
 }
 else {
  Value x;
  Jsonize(JsonIO(x, false), value);
  ValueMap m; // bottleneck here
  if(IsValueMap(node))
   m = node;
  m.Add(key, x);
  node = m;
 }
 return *this;
}

template <class T>
void JsonizeValue(JsonIO io, T& var)
{ // bad code
 if(io.IsLoading())
  var = io.node;
 else
  io.node = var;
}

template<> inline void Jsonize(JsonIO io, int& var)      { JsonizeValue(io, var); } // Check for
correct types
template<> inline void Jsonize(JsonIO io, double& var)   { JsonizeValue(io, var); }
template<> inline void Jsonize(JsonIO io, bool& var)     { JsonizeValue(io, var); }
template<> inline void Jsonize(JsonIO io, String& var)   { JsonizeValue(io, var); }

template <class T, class V>
```

```
void JsonizeArray(JsonIO io, T& array)
{
 if(io.IsLoading()) {
  array.Clear();
  for(int i = 0; i < io.node.GetCount(); i++)
   Jsonize(JsonIO(io.node[i]), array.Add());
 }
 else {
  ValueArray va;
  for(int i = 0; i < array.GetCount(); i++) {
   Value x;
   Jsonize(JsonIO(x, false), array[i]);
   va.Add(x);
  }
  io.node = va;
 }
}

template <class T>
void Jsonize(JsonIO io, Vector<T>& var)
{
 JsonizeArray<Vector<T>, T>(io, var);
}

template <class T>
void Jsonize(JsonIO io, Array<T>& var)
{
 JsonizeArray<Array<T>, T>(io, var);
}

struct Test {
 int a, b;

 void Jsonize(JsonIO io) {
  io("a", a)
    ("b", b);
 }

 void Serialize(Stream& s) {
  s % a % b;
 }
};

template <class T>
String StoreAsJson(const T& var)
{
 Value x;
 Jsonize(JsonIO(x, false), const_cast<T&>(var));
```

```
 return AsJSON(x);
}

template <class T>
void LoadFromJson(T& var, const char *json)
{
 Value x;
  x = ParseJSON(json);
 Jsonize(JsonIO(x, true), var);
}

CONSOLE_APP_MAIN
{
 Array<Test> test;
 for(int i = 0; i < 10; i++) {
  Test t;
  t.a = 1 + i;
  t.b = 23 + i;
  test.Add(t);
 }

 String json = StoreAsJson(test);
 DUMP(json);
 Array<Test> test2;
 LoadFromJson(test2, json);
 DUMP(StoreAsJson(test2));

 const int N = 10000;
 for(int i = 0; i < N; i++) {
  Array<Test> test2;
  RTIMING("Jsonize");
  LoadFromJson(test2, StoreAsJson(test));
 }
 for(int i = 0; i < N; i++) {
  Array<Test> test2;
  RTIMING("Serialize");
  LoadFromString(test2, StoreAsString(test));
 }

 Test tt;
 tt.a = 1; tt.b = 2;
 for(int i = 0; i < 10 * N; i++) {
  RTIMING("Jsonize one");
  Test tt2;
  LoadFromJson(tt2, StoreAsJson(tt));
 }
 for(int i = 0; i < 10 * N; i++) {
  RTIMING("Serialize one");
```

```
  Test tt2;
  LoadFromString(tt2, StoreAsString(tt));
 }
}
```

Note about benchmarking: In your code, there seems to be 7.8 ratio json / binary. This is a little bit illusory because binary serialization is this case is slowed down by doing some processing per serialization call (magic numbers, versions etc..). Means, if you serialize more data, like Vector here, binary serialization becomes relatively faster. I am getting about 6 ration for "jsonize one"/"serialize one" (which is equivalent of your code), but this grows to about 25 for benchmarking with Array of 10 elements... I guess it would be the same for your code too.

Subject: Re: Json serialization support
Posted by Mindtraveller on Sun, 05 Feb 2012 21:10:12 GMT
View Forum Message <> Reply to Message

Thank you, Mirek!
I agree with your corrections (code looks much better).
Hope to see Jsonize in U++ SVN soon.
I think JSON better fits human-friendly storing of different things than XML. So I look forward to change storing of data from XML towards JSON.
It's more effective, more intuitive and less overweighted.

Subject: Re: Json serialization support
Posted by Mindtraveller on Fri, 10 Feb 2012 15:57:20 GMT
View Forum Message <> Reply to Message

Mirek, is it possible to add Jsonize to U++ Core?

Subject: Re: Json serialization support
Posted by mirek on Fri, 10 Feb 2012 16:03:08 GMT
View Forum Message <> Reply to Message

Mindtraveller wrote on Fri, 10 February 2012 10:57Mirek, is it possible to add Jsonize to U++ Core?

I am not at what I would consider "production quality" yet.

There are issues to be solved, like storing int64 numbers (as text) etc...

If you need this now, perhaps you could fetch it from sandbox ('structurize'), interface is not going to change.
