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.
