file PropertyConverter.h

PropertyConverter.h

```
1 /*************************************************************\
2 Original Author: Georg Fritzsche
3 Created: 2009-11-07
4 License: Dual license model; choose one of two:
5  New BSD License
6  http://www.opensource.org/licenses/bsd-license.php
7  - or -
8  GNU Lesser General Public License, version 2.1
9  http://www.gnu.org/licenses/lgpl-2.1.html
10
11 Copyright 2009 Georg Fritzsche, Firebreath development team
12 \*************************************************************/
13
14 #pragma once
15 #ifndef PROPERTY_CONVERTER_H
16 #define PROPERTY_CONVERTER_H
17
18 #include <boost/function.hpp>
19 #include <boost/bind.hpp>
20 #include <boost/function_types/is_member_function_pointer.hpp>
21 #include <boost/function_types/result_type.hpp>
22 #include <boost/function_types/parameter_types.hpp>
23 #include <boost/mpl/equal.hpp>
24 #include <boost/mpl/front.hpp>
25 #include "APITypes.h"
26 #include "ConverterUtils.h"
27
28 namespace FB
29 {
43  template<class C, typename F1, typename F2>
44  inline PropertyFunctors
45  make_property(C* instance, F1 getter, F2 setter);
46
59  template<class C, typename F>
60  inline PropertyFunctors
61  make_property(C* instance, F getter);
62
63  namespace detail { namespace properties
64  {
65  template<class C, bool IsConst = false>
66  struct select_get_property_functor {
67  template<typename T>
68  static inline
69  FB::GetPropFunctor f(C* instance, T (C::*getter)())
70  {
71  return boost::bind(boost::mem_fn(getter), instance);
72  }
73  };
74
75  template<class C>
76  struct select_get_property_functor<C, /* IsConst= */ true> {
77  template<typename T>
78  static inline
79  FB::GetPropFunctor f(C* instance, T (C::*getter)() const)
80  {
81  return boost::bind(boost::mem_fn(getter), instance);
82  }
83  };
84
85  template<class C, typename F>
86  struct getter
87  {
88  enum { const_qualified =
89  boost::function_types::is_member_function_pointer
90  <F, boost::function_types::const_qualified>::value };
91
92  typedef select_get_property_functor<C, const_qualified> result;
93  };
94
95  template<class C, bool IsConst = false>
96  struct select_set_property_functor {
97  template<typename T>
```

```
98  static inline
99  FB::SetPropFunctor f(C* instance, void (C::*setter)(T))
100  {
101  typedef typename FB::detail::plain_type<T>::type Ty;
102  typedef FB::detail::converter<Ty, FB::variant> converter;
103  return
104  boost::bind(setter, instance,
105  boost::bind(&converter::convert, _1));
106  }
107  };
108
109  template<class C>
110  struct select_set_property_functor<C, /* IsConst= */ true> {
111  template<typename T>
112  static inline
113  void f(C* instance, void (C::*setter)(T) const, const FB::variant& v)
114  {
115  typedef typename FB::detail::plain_type<T>::type Ty;
116  typedef FB::detail::converter<Ty, FB::variant> converter;
117  return
118  boost::bind(setter, instance,
119  boost::bind(&converter::convert, _1));
120  }
121  };
122
123  template<class C, typename F>
124  struct setter
125  {
126
127  enum { const_qualified =
128  boost::function_types::is_member_function_pointer
129  <F, boost::function_types::const_qualified>::value };
130
131  typedef select_set_property_functor<C, const_qualified> result;
132  };
133
134  inline void dummySetter(const FB::variant&)
135  {
136
137  }
138  } }
139
140  // make read/write property functor
141
142  template<class C, typename F1, typename F2>
143  inline PropertyFunctors
144  make_property(C* instance, F1 f1, F2 f2)
145  {
146  return PropertyFunctors(
147  FB::detail::properties::getter<C, F1>::result::f(instance, f1),
148  FB::detail::properties::setter<C, F2>::result::f(instance, f2));
149  }
150
151  // make read-only property
152
153  template<class C, typename F>
154  inline PropertyFunctors
155  make_property(C* instance, F f)
156  {
157  return PropertyFunctors(
158  FB::detail::properties::getter<C, F>::result::f(instance, f),
159  boost::bind(FB::detail::properties::dummySetter, _1));
160  }
161 }
162
163 #endif // PROPERTY_CONVERTER_H
164
```

FB::GetPropFunctor
boost::function< FB::variant()> GetPropFunctor
Defines an alias representing a property getter functor used by FB::JSAPIAuto.
Definition: APITypes.h:307
FB::make_property
PropertyFunctors make_property(C *instance, F1 getter, F2 setter)
Generate read-write property functors for use with registerProperty() of FB::JSAPIAuto.
Definition: PropertyConverter.h:144
FB::SetPropFunctor
boost::function< void(const FB::variant &)> SetPropFunctor
Defines an alias representing a property setter functor used by FB::JSAPIAuto.
Definition: APITypes.h:309
FB::variant

Accepts any datatype, used in all interactions with javascript. Provides tools for getting back out t...
Definition: variant.h:198

FB::PropertyFunctors

used by FB::JSAPIAuto to store property implementation details, created by FB::make_property().
Definition: APITypes.h:311
