```
    1: #include "corba.h"
    2: #include "CosEventComm_s.hh"
    3: #include "CosEventChannelAdmin_c.hh"
    4: #include "vport.h"
    5:
    6: USE_STD_NS
    7:
    8: class PushModel : public POA_CosEventComm::PushSupplier, public VISThread
    9: {
   10:   public:
   11:     PushModel(CORBA::ORB_ptr orb,
   12:     CosEventComm::PushConsumer_ptr pushConsumer,
   13:     PortableServer::POA_ptr myPOA) :
   14:     _orb(orb), _pushConsumer(pushConsumer), _myPOA(myPOA), _counter(0), _delay
(1)
   15:     {}
   16:
   17:     void delay(int time) { _delay = time; }
   18:
   19:     void start() {
   20:       // start the thread
   21:       run();
   22:     }
   23:
   24:     void disconnect_push_supplier() {
   25:       cout << "Model::disconnect_push_supplier()" << endl;
   26:       try {
   27:         PortableServer::ObjectId_var objId =
   28:         PortableServer::string_to_ObjectId("PushModel");
   29:
   30:         _myPOA->deactivate_object(objId);
   31:       }
   32:         catch(const CORBA::Exception& e) {
   33:         cout << e << endl;
   34:       }
   35:     }
   36:
   37:     // implement begin() callback
   38:     void begin() {
   39:       while(true) {
   40:         VISPortable::vsleep(_delay);
   41:
   42:         try {
   43:           char buf[81];
   44:           sprintf(buf, "%s%d", "Hello #", ++_counter);
   45:
   46:           CORBA::Any_var message = new CORBA::Any();
   47:           *message <<= buf;
   48:           cout << "Supplier pushing: " << buf << endl;
   49:
   50:           _pushConsumer->push(*message);
   51:         }
   52:         catch(CosEventComm::Disconnected e) {
   53:           cout << "Disconnected #" << _counter << endl;
   54:         }
   55:         catch(CORBA::OBJECT_NOT_EXIST e)
   56:         {
   57:           cout << "Push Consumer has been disconnected" << endl;
   58:           return;
   59:         }
   60:         catch(const CORBA::Exception& e) {
   61:           cout << e << endl;
   62:           disconnect_push_supplier();
   63:           return;
   64:         }
   65:         catch(...) {
   66:           cout << "Unexpected exception" << endl;
   67:           disconnect_push_supplier();
   68:           return;
   69:         }
   70:       }
   71:     }
   72:
   73:   private :
   74:     int                             _delay;
   75:     int                             _counter;
   76:     CORBA::ORB_var                  _orb;
   77:     PortableServer::POA_var         _myPOA;
   78:     CosEventComm::PushConsumer_var  _pushConsumer;
   79: };
   80:
   81: int main(int argc, char* const* argv)
   82: {
   83:   try {
   84:     // Initialize the ORB.
   85:     CORBA::ORB_var orb = CORBA::ORB_init(argc, argv);
   86:
   87:     // get a reference to the root POA
   88:     CORBA::Object_var obj = orb->resolve_initial_references("RootPOA");
   89:     PortableServer::POA_var rootPOA = PortableServer::POA::_narrow(obj);
   90:
   91:     // Create policies for our persistent POA
   92:     CORBA::PolicyList policies;
   93:     policies.length(1);
   94:     policies[(CORBA::ULong)0] =
   95:       rootPOA->create_lifespan_policy(PortableServer::PERSISTENT);
   96:
   97:     PortableServer::POAManager_var poa_manager = rootPOA->the_POAManager();
   98:
   99:     // Create serverPOA with the right policies
  100:     PortableServer::POA_var serverPOA =
  101:       rootPOA->create_POA("event_service_poa", poa_manager, policies);
  102:
  103:     CosEventChannelAdmin::EventChannel_var channel = NULL;
  104:     PushModel* model = NULL;
  105:     CosEventChannelAdmin::ProxyPushConsumer_var pushConsumer = NULL;
  106:
  107:     while(true) {
  108:       try {
  109:       cout << "-> ";
  110:       cout.flush();
  111:
  112:         char cmd;
  113:
  114:         if (cin >> cmd ) {
  115:           if (cmd == 'e') {
  116:             obj = orb->resolve_initial_references("EventService");
  117:             channel = CosEventChannelAdmin::EventChannel::_narrow(obj);
  118:             cout << "Located event channel: " << channel << endl;
  119:             continue;
  120:           }
  121:           else if (cmd == 'p') {
  122:             if (channel == NULL) {
  123:               cout << "Need to locate an [e]vent channel" << endl;
  124:             }
  125:             else {
  126:               pushConsumer = channel->for_suppliers()->obtain_push_consumer();
  127:               cout << "Obtained push consumer: " << pushConsumer << endl;
  128:               continue;
  129:             }
  130:           }
  131:           else if (cmd == 'm') {
  132:             if (pushConsumer == NULL) {
  133:               cout << "Need to obtain a [p]ush consumer" << endl;
  134:             }
  135:             else {
```

```
136:               model = new PushModel(orb, pushConsumer, serverPOA);
137:               CORBA::String_var supplier_name(CORBA::string_dup("PushModel"));
138:               PortableServer::ObjectId_var objId =
139:               PortableServer::string_to_ObjectId(supplier_name);
140:               serverPOA->activate_object_with_id(objId, model);
141:               // Activate the POA Manager
142:               serverPOA->the_POAManager()->activate();
143:               CORBA::Object_var reference = serverPOA->servant_to_reference(mo
del);
144:               cout << "Created model: " << reference << endl;
145:               continue;
146:             }
147:           }
148:           else if (cmd == 's') {
149:             if (model == NULL) {
150:               cout << "Need to create a [m]odel" << endl;
151:             }
152:             else {
153:               int delay;
154:
155:               if (cin >> delay ) {
156:                 if (delay < 0)
157:                   cout << "[s]leep delay must be positive" ;
158:                 else
159:                   model->delay(delay);
160:               }
161:               else {
162:                 cerr << "Invalid argument to [s]leep" << endl;
163:               }
164:             }
165:           }
166:           else if (cmd == 'c' ) {
167:             if (model == NULL) {
168:               cout << "Need to create a [m]odel" << endl;
169:             }
170:             else if (pushConsumer == NULL) {
171:               cout << "Need to obtain a [p]ush consumer" << endl;
172:             }
173:             else {
174:               cout << "Connecting..." << endl;
175:               pushConsumer->connect_push_supplier(model->_this());
176:               model->start();
177:               continue;
178:             }
179:           }
180:           else if (cmd == 'd') {
181:             if (pushConsumer == NULL) {
182:               cout << "Need to obtain a [p]ush consumer" << endl;
183:             }
184:             else {
185:               cout << "Disconnecting..." << endl;
186:               pushConsumer->disconnect_push_consumer();
187:               continue;
188:             }
189:           }
190:           else if (cmd == 'q') {
191:             cout << "Quitting..." << endl;
192:             CORBA::ORB::shutdown(1UL);
193:             break;
194:           }
195:           else
196:           {
197:             cout  << "Commands: e               [e]vent channel"    << endl
198:                   << "          s <# seconds>   set [s]leep delay" << endl
199:                   << "          p               [p]ush consumer"    << endl
200:                   << "          m               [m]odel"            << endl
201:                   << "          c               [c]onnect"          << endl
202:                   << "          d               [d]isconnect"       << endl
203:                   << "          q               [q]uit"             << endl;
204:           }
205:         }
206:       }
207:       catch(const CORBA::SystemException& e) {
208:     cerr << e << endl;
209:       }
210:     }
211:   }
212:   catch(const CORBA::Exception& e) {
213:       cerr << e << endl;
214:   }
215:
216:   return 0;
217: }
218:
219:
```

```
    1: #include "corba.h"
    2: #include "CosEventComm_s.hh"
    3: #include "CosEventChannelAdmin_c.hh"
    4: #include "vport.h"
    5:
    6: USE_STD_NS
    7:
    8: class PushView : public POA_CosEventComm::PushConsumer
    9: {
   10: public:
   11:     void push(const CORBA::Any& data) {
   12:       cout << "Consumer being pushed: " << data << endl;
   13:     }
   14:
   15:     void disconnect_push_consumer() {
   16:       cout << "PushView::disconnect_push_consumer" << endl;
   17:     }
   18: };
   19:
   20: int main(int argc, char* const* argv)
   21: {
   22:   try {
   23:     // Initialize the ORB.
   24:     CORBA::ORB_var orb = CORBA::ORB_init(argc, argv);
   25:
   26:     // get a reference to the root POA
   27:     CORBA::Object_var obj = orb->resolve_initial_references("RootPOA");
   28:     PortableServer::POA_var rootPOA = PortableServer::POA::_narrow(obj);
   29:
   30:     // Create policies for our persistent POA
   31:     CORBA::PolicyList policies;
   32:     policies.length(1);
   33:     policies[(CORBA::ULong)0] =
   34:       rootPOA->create_lifespan_policy(PortableServer::PERSISTENT);
   35:
   36:     PortableServer::POAManager_var poa_manager = rootPOA->the_POAManager();
   37:
   38:     // Create serverPOA with the right policies
   39:     PortableServer::POA_var serverPOA =
   40:       rootPOA->create_POA("event_service_poa", poa_manager, policies);
   41:
   42:     CosEventChannelAdmin::EventChannel_var channel = NULL;
   43:     PushView* view = NULL;
   44:     CosEventChannelAdmin::ProxyPushSupplier_var pushSupplier = NULL;
   45:
   46:     while(true) {
   47:       try {
   48:         cout << "-> ";
   49:         cout.flush();
   50:
   51:         char cmd;
   52:         if (cin >> cmd) {
   53:             if (cmd == 'e') {
   54:                 obj = orb->resolve_initial_references("EventService");
   55:                 channel = CosEventChannelAdmin::EventChannel::_narrow(obj);
   56:                 cout << "Located event channel: " << channel << endl;
   57:                 continue;
   58:             }
   59:             else if (cmd == 'v') {
   60:                 view = new PushView();
   61:                 CORBA::String_var consumer_name(CORBA::string_dup("PushView"))
;
   62:                 PortableServer::ObjectId_var objId =
   63:                     PortableServer::string_to_ObjectId(consumer_name);
   64:                 serverPOA->activate_object_with_id(objId, view);
   65:                 // Activate the POA Manager
   66:                 serverPOA->the_POAManager()->activate();
   67:                 CORBA::Object_var reference = serverPOA->servant_to_reference(
view);
   68:                 cout << "Created view: " << reference << endl;
   69:                 continue;
   70:             }
   71:             else if (cmd == 'p') {
   72:                 if (channel == NULL) {
   73:                     cout << "Need to locate an [e]vent channel" << endl;
   74:                 }
   75:                 else {
   76:                     pushSupplier = channel->for_consumers()->obtain_push_suppl
ier();
   77:                     cout << "Obtained push consumer: " << pushSupplier << endl
;
   78:                     continue;
   79:                 }
   80:             }
   81:             else if (cmd == 'c' ) {
   82:                 if (view == NULL) {
   83:                     cout << "Need to create a [v]iew" << endl;
   84:                 }
   85:                 else if (pushSupplier == NULL) {
   86:                     cout << "Need to obtain a [p]ush supplier" << endl;
   87:                 }
   88:                 else {
   89:                     cout << "Connecting..." << endl;
   90:                     pushSupplier->connect_push_consumer(view->_this());
   91:                     continue;
   92:                 }
   93:             }
   94:             else if (cmd == 'd') {
   95:                 if (pushSupplier == NULL) {
   96:                     cout << "Need to obtain a [p]ush supplier" << endl;
   97:                 }
   98:                 else {
   99:                     cout << "Disconnecting..." << endl;
  100:                     pushSupplier->disconnect_push_supplier();
  101:                     continue;
  102:                 }
  103:             }
  104:             else if (cmd == 'q') {
  105:                 cout << "Quitting..." << endl;
  106:                 break;
  107:             }
  108:             cout << "Commands: e             [e]vent channel"    << endl
  109:                  << "          p             [p]ush supplier"    << endl
  110:                  << "          v             [v]iew"             << endl
  111:                  << "          c             [c]onnect"          << endl
  112:                  << "          d             [d]isconnect"       << endl
  113:                  << "          q             [q]uit"            << endl;
  114:         }
  115:       }
  116:       catch(const CORBA::SystemException& e) {
  117:           cerr << e << endl;
  118:       }
  119:     }
  120:   }
  121:   catch(const CORBA::Exception& e) {
  122:     cerr << e << endl;
  123:   }
  124:
  125:   return 0;
  126: }
```