

CYBERNET

OpenSkies

Networking Engine

Point-to-Point Switch (P2PS) API Guide



www.openskies.net

MMPG
MASSIVE
MULTIPLAYER
ONLINE GAMING

Openskies P2PS Guide

The P2P Switch subsystem, or P2PS, gives the developer a means to communicate information between objects without having to deal with issues like when the object is created or whether the object is local or remote. The API is simple and is available as part of the IET library.

Creating an Object

All network messages in the P2PS are sent from one object to another, so in order to send or receive messages, you need objects. To create an object, all you have to do is to derive your own class from *CswitchObject* (RTISwitch.h). Use the following code as a template for deriving your own class:

```
class CTestObject : public CSwitchObject
{
public:
    CTestObject(char *name, char *classname) : CSwitchObject(name,
        classname) {}
    virtual bool HandleData(const unsigned char *data, long size,
        CSwitchObject *from);
};
```

You can add whatever other methods and variables you may desire, but you must include the constructor above as well as a declaration and definition of the HandleData function. HandleData will be called whenever this object is to receive data. In your application, you can now create an instance of the object:

```
m_SendingObject = new CTestObject("bob",
    "COpenskiesRTIObject_S");
```

The first argument to the constructor is the name of the object. This name must be unique to this one object across the entire network. The second argument is the name of the object definition in the CybernetRTI.ini file. As described in the IET section, the CybernetRTI document maps the object variables in the application to the objects and attributes in the FOM (federate object model), which is defined in the *.fed* file. To include the P2P Switch functionality in your application you must include a special object definition in your CybernetRTI.ini file as well as your *.fed* file.

As described in the IET section, the *.fed* file and the CybernetRTI.ini file are both generated automatically using the *Configure.exe* tool. To P2PS-enable your application, simply run the Configure tool, open your existing project (or create a new project), and proceed to the fourth tab labeled *P2P Switch Objects* (See Figure 1).

In this tab, you must first enable P2P switch objects by clicking the checkbox. Then proceed to add new switch names by right clicking in the edit area and adding names.

For each name, you can select whether or not to subscribe and/or publish (record is a future feature) to data from those objects.

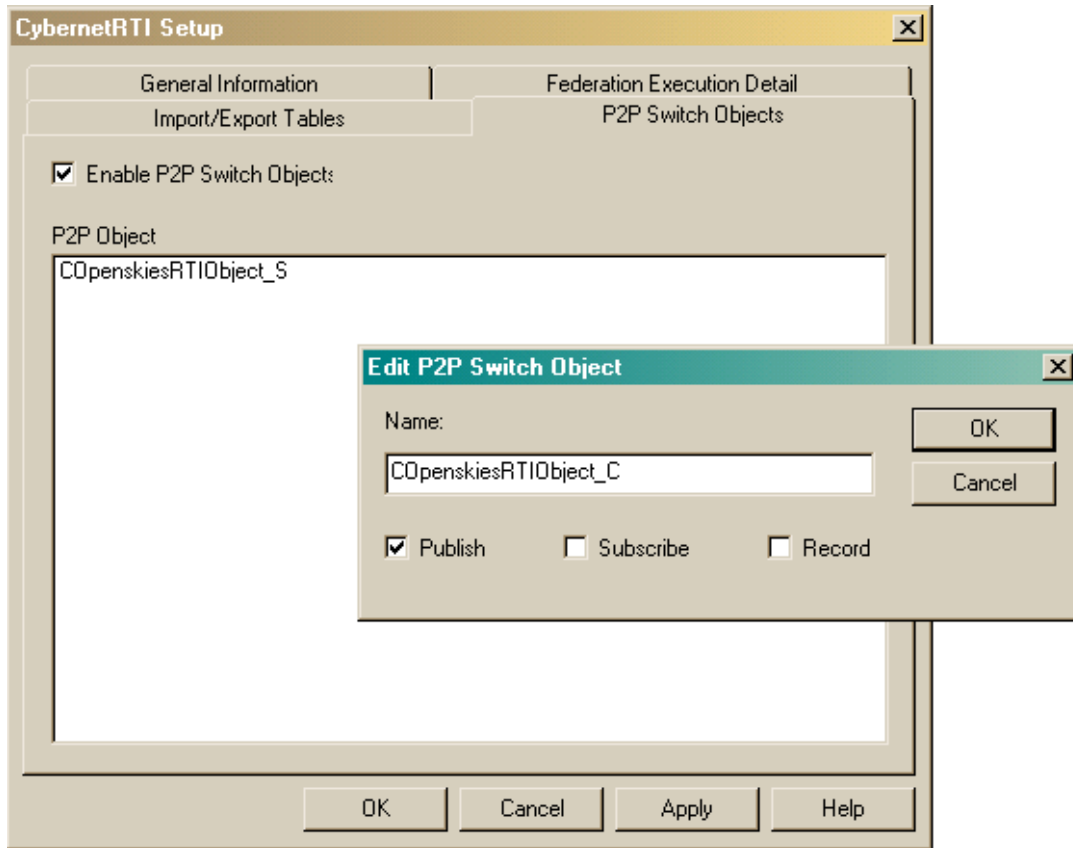


Figure 1: Adding Switch Objects in *Configure.exe*

Turning off subscribe can reduce network traffic by disallowing communication from a particular switch object name. For example, if you want to use the P2P Switch library to communicate between clients/federates and game servers, but do not want clients to communicate directly to other clients, then use Configure to create 2 INI versions, one for client and one for game server.

- Game Server:
 - Publishes “*COpenskiesRTIObject_S*”
 - Subscribes to “*COpenskiesRTIObject_C*”
 - Subscribes to “*COpenskiesRTIObject_S*”
- Client:
 - Publishes “*COpenskiesRTIObject_C*”
 - Subscribes only to “*COpenskiesRTIObject_S*”

Registering an Object

Once you've created an object, you'll want to register it with the network. If you don't register your object, you will not be able to send or receive messages using this object. Registering the object is very easy:

```
m_SendingObject->Register();
```

When you want to remove this object from the network, simply call the `Unregister` function:

```
m_SendingObject->Unregister();
```

Finding an Object

In order to send a message to an object, you need to get an object pointer. To get an object pointer you use the `Find` function. The `Find` function is available globally as a member function of the `CRTISwitch` class. Use it like this:

```
CTestObject *object_steve = (CTestObject *) (CRTISwitch::Instance()->Find("steve",  
"COpenskiesRTIObject_S"));
```

The `find` function returns an object pointer for the object with the name you query. If no object with that name has been registered, the function will still return a valid pointer. In this case, any data you send to that object will be buffered until such time as the object is registered.

Sending a Message

Messages are sent from and received by objects. Sending a message is accomplished by using the `Send` function. The `Send` function is available globally as a member function of the `CRTISwitch` class.

```
Char *stufftosend = "hello steve, this is bob";  
CRTISwitch::Instance()->Send(m_SendingObject, object_steve,  
0, stufftosend, strlen(stufftosend));
```

And that's really all there is to the P2PS interface. The following describes the functions used in the examples above in more detail.

Reference

First the global functions:

CRTISwitch::Find

```
CSwitchObject *CRTISwitch::Find(const char *name, const char
    *p2ps_object_name);
```

Usage:

```
CTestObject *object = (CTestObject
    *) (CRTISwitch::Instance()->Find("steve",
    "COpenskiesRTIObject_S"));
```

Description:

The find function looks through the list of objects registered locally or remotely for a unique name (e.g. "steve"). The type of P2PS object that it is looking for is specified as well (e.g. "COpenskiesRTIObject_S")

Arguments:

- *Name*: this is the unique name being queried.
- *P2ps_object_name*: this is the name for the P2PS object type. This is specified in the Configure.exe tool under the P2PS tab.

Return Value:

If successful, this function returns a pointer to the object. It returns NULL on failure. The object does not need to have been registered or even to exist in order for the function to successfully return a pointer. If the object has not been registered, the returned pointer is a placeholder where messages will be buffered until the unique name is registered.

CRTISwitch::Send

```
int CRTISwitch::Send(CSwitchObject *from_object, CSwitchObject
    *to_object, long dataID, char *data, long size);
```

Usage:

```
Char *stufftosend = "hello steve, this is bob";
int rtn = CRTISwitch::Instance()->Send(object,
    m_SendingObject, 0, stufftosend, strlen(stufftosend));
```

Description:

The send function simply sends raw data from one object, which must be local, to another object, which can be local or remote.

Arguments:

- *From_object*: this a pointer to the sending object
- *To_object*: this a pointer to the receiving object (typically retrieved using *Find*)
- *DataID*: a 4-byte integer that will be transmitted to the receiver as well.
- *Data*: a pointer to the beginning of the data buffer to be transmitted.
- *Size*: a 4-byte integer that specifies the length, in bytes, of the outgoing data.

Return Value:

If successful, this function returns 1. On failure it returns 0. One possible reason for failure is that the sending object is not local.

The remainder of the functions are CSwitchObject functions:

CSwitchObject::HandleData

```
virtual bool HandleData(const unsigned char *data, long size,  
                        long dataID, CSwitchObject *from);
```

Usage:

You do not call this function from within your code.

Description:

The HandleData function is the mechanism by which the application is informed of incoming Switch traffic. You must override this function in each of your locally created SwitchObjects (objects derived from CSwitchObject).

Arguments:

- *from*: this a pointer to the object that sent the data
- *To_object*: this a pointer to the receiving object (typically retrieved using *Find*)
- *DataID*: the 4-byte integer that was be transmitted by the sender
- *Data*: a pointer to the beginning of the data buffer containing the received data.
- *Size*: a 4-byte integer that specifies the length, in bytes, of the incoming data.

Return Value:

If successful, your overridden HandleData should return 1. On failure it returns 0.

CSwitchObject::Register

```
int Register(void);
```

Usage:

```
int rtn = object->Register();
```

Description:

The Register function makes your object visible across the network.

Arguments:

none

Return Value:

If successful, returns 1. On failure it returns 0

CSwitchObject::Unregister

```
int Unregister(void);
```

Usage:

```
int rtn = object->Unregister();
```

Description:

The Register function removes this object from the network.

Arguments:

none

Return Value:

If successful, returns 1. On failure it returns 0.