

CYBERNET

OpenSkies

Networking Engine

Culling Guide



www.openskies.net

MMPG
MASSIVE
MULTIPLAYER
ONLINE GAMING

Openskies Culling Guide

A culling module is a plug-in for FedHost running on a public server, which is referenced in as described in Step 5, “Configure Openskies”. You will never need to worry about culling if you do not use a public server. In the absence of a public server, Openskies runs in a peer-to-peer networking mode. Anyone who has access to the underlying network connections will receive all information in all classes that to which he subscribes. If you do use a public server, however, culling modules may become necessary to avoid network traffic congestions and enable massive multiplayer. C++ Code examples of CullingRules can be found in the NetMAXOS distribution of the Openskies SDK in the following directories:

- \$root/Cybernet/CullingRulesII
- \$root/Cybernet/CullingRulesIII

and can also be found (but won't compile) on the Windows Openskies SDK in the \CybernetRTI_Static\Samples\ServerSamples folder.

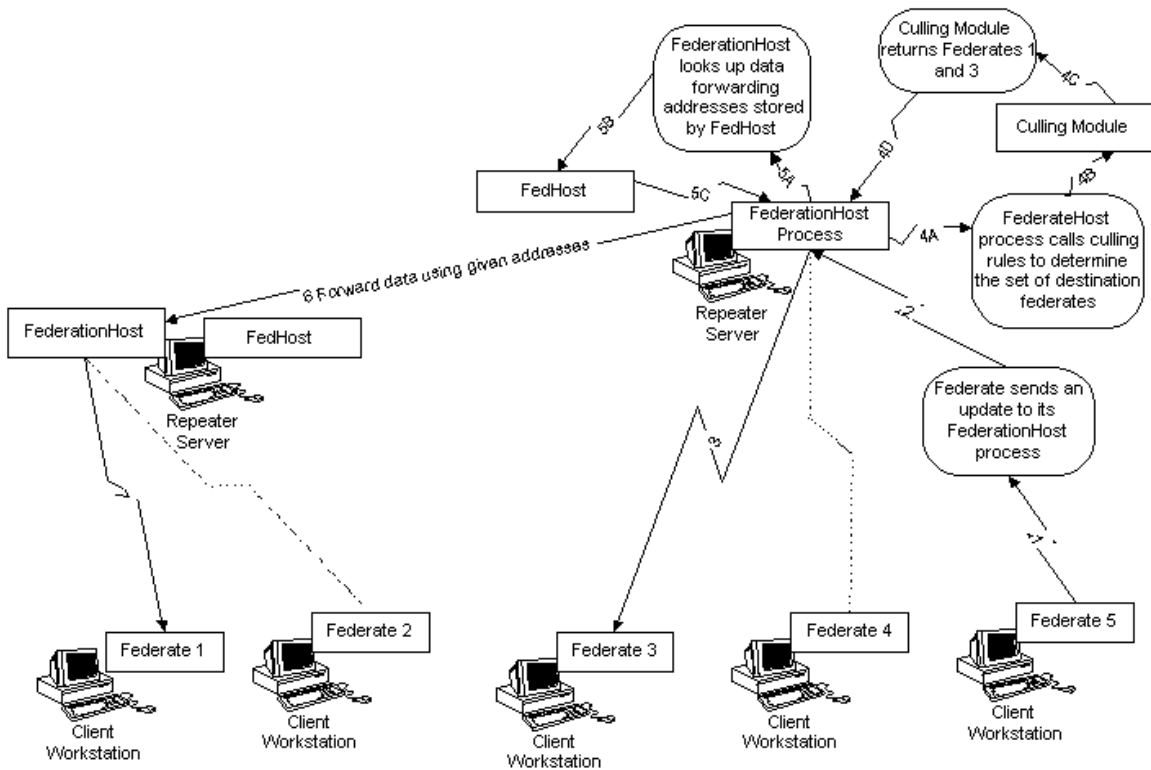


Figure 1: FedHost Culling

Culling modules are connected to attribute set types or classes, which are described in Step 1. Use the Openskies Setup Tool. A culling module may contain culling code for

multiple attribute set classes. In such case, the culling module is a library of many code segments that handle the various attributes defined in the FED file (created by the Setup tool) but may not directly relate to each other.

Exported Function

Each culling module must export a function prototyped as follows:

```
extern "C" CAttributeSetClassEx
    *GetAttributeSetClassEx(const CAttributeSetClass
        *pClass)
```

where *CAttributeSetClass* and *CAttributeSetClassEx* are classes defined in *AttributeSetClass.h*.

Suppose that there are two attribute set classes, *AttributeSetClass1* and *AttributeSetClass2* defined in your FED file, and you are building a culling module for both, the following code sample can be used for your version of this function:

```
extern "C" CAttributeSetClassEx *GetAttributeSetClassEx(
    const CAttributeSetClass *pClass)
{
    /* The pClass->GetName() function holds the name of the
       object or interaction where this culling module was
       found */
    if(!strcmp(pClass->GetName(), "AttributeSetClass1"))
    {
        try
        {
            /* If we found the class to be
               AttributeSetClass1, we create a class extension
               for it. Only one of these will be created, since
               the fed file is only parsed once. The class
               definition is in AttributeSetClass1Ex.cpp */

            return new CAttributeSetClassEx(pClass);
        }
        catch(...)
        {
            return NULL;
        }
    }

    if(!strcmp(pClass->GetName(), "AttributeSetClass2"))
    {
        try
        {
```

```

        /* If we found the class to be
        AttributeSetClass2, we create
        a class extension for it. Only one of these will
        be created, since the fed file is only parsed
        once. The class definition is in
        AttributeSetClass2Ex.cpp */

        return new CAttributeSetClass2Ex;
    }
    catch(...)
    {
        return NULL;
    }
}

// Unknown class
return NULL;
}

```

where you must define the classes *CAttributeSetClass1Ex* and *CAttributeSetClass2Ex*, and derive each class from *CAttributeSetClassEx*. In general, culling modules must define such attribute set class extensions, and derive them from *CAttributeSetClassEx*. From this point on, each time FedHost creates a new attribute set of *CAttributeSetClass1* type, for example, it will call *CAttributeSetClass1Ex::CreateAttributeSet*.

Attribute Set Class Extensions

When defining an attribute set class extension, you must define the first of two functions, and can optionally define the second. The first function is:

```

virtual CAttributeSet * CAttributeSetClassEx::
    CreateAttributeSet(DWORD dwClassHandle, DWORD
        dwOwnerHandle, const char *pszName)

```

This is a purely virtual function that must be overridden. It is used only if the attribute set is not an interaction. If the attribute set *is* an interaction, you can simply let it return NULL since it is never supposed to be used. When the FedHost calls your *CreateAttributeSet* function is saying

“I’m reading the FED file and just found a new object. Here is it’s name pszName. Here is its classHandle and owner handle in case you need them. Build an instance of this object I can store or if you don’t want to, just return NULL”

```

virtual bool CAttributeSetClassEx::
    GetPreferredRecipientIDArray(const CNetDataBuffer
        &NetDataBuffer, CAttributeSet *pCullingAttributeSet,
        CSortedDWORDArray &RecipientIDArray)

```

This is not a purely virtual function, however you must override this function if the attribute set is an interaction. Otherwise you do not need to override it. For non-interactions you will instead override member functions of the attribute set as described in the next section. *NetDataBuffer* contains the values of this interaction, i.e. the actual data that came over the network from the federate. *pCullingAttributeSet* is an attribute set of a different class that is not an interaction and associated with this interaction for culling. It is specified by the client application by calling the function *CImportExportTable::SetAttributeSetCullingHandle* in the *Openskies IET API*.

The FedHost will call your *GetPreferredRecipientIDArray* function and will expect that it will copy into *RecipientIDArray* a list of attribute set handles whose owners should receive this interaction. ***GetPreferredRecipientIDArray is where culling for interactions takes place.***

Derived Attribute Set Class

If the attribute set is not an interaction, besides defining *GetAttributeSetClassEx* and attribute set class extensions, you must also define derived classes from *CAttributeSet* defined in *AttributeSet.h*. The reason for this is that interactions are transitory in nature and have no persistency on the FedHost. The FedHost only knows that there will be interactions with a certain name, which is what *AttributeSetClassEx* is for. Objects, on the other hand can have a persistent representation on the FedHost in addition to the abstract class representation. In other words, an instance of the object will be created on the FedHost for every client object that is registered. Such instances are derived from *CAttributeSet*. Your *CAttributeSet* derived class must override the following member functions:

SetCulleeHandle:

```
virtual void CattributeSet::SetCulleeHandle(DWORD  
    dwCulleeClassHandle, DWORD dwCulleeHandle)
```

The FedHost calls your *SetCulleeHandle* when an object or interaction, whose culling is based on this attribute set, is registered. Culling associations are set on the client using the function *CImportExportTable::SetAttributeSetCullingHandle* in the *Openskies IET API*. When the FedHost calls this function, it is telling the *CattributeSet*

“Another attribute set with this handle is using you as a culling base”. You can override this function and store this information if your culling module requires it.

UpdateValues

```
virtual bool CattributeSet::UpdateValues(const  
    CNetDataBuffer &NetDataBuffer)
```

Your culling module will receive value updates via this function. Return true means that this has caused a change in the “preferred recipient ID array” for this attribute set or some other attribute sets. Additionally, a return of true will cause the update to be forwarded to other FedHosts on the network. When the FedHost calls this function, it is telling the CattributeSet:

“This object has just sent us another update. Here is a buffer with the data it sent.”

GetPreferredRecipientIDArray

```
virtual bool  
    CattributeSet::GetPreferredRecipientIDArray(CSortedDWord  
        DArray &RecipientIDArray)
```

Override this function if the culling is not based on any other attribute set class. This function should copy into RecipientIDArray a list of attribute set handles whose owners should discover this attribute set. Note that discovery does not guarantee that these owners will receive updates unless the Cull function returns true also. Return false if all subscribers should discover this attribute set. Be very careful when returning false since it disables culling. When the FedHost calls this function, it is telling the CattributeSet:

“Give me a list of handles for all the attribute sets that should see you”

This function does culling. It is actually the first of a two-stage culling process. GetPreferredRecipientIDArray (both versions) allows you to construct a list of other objects that should get this object’s updates.

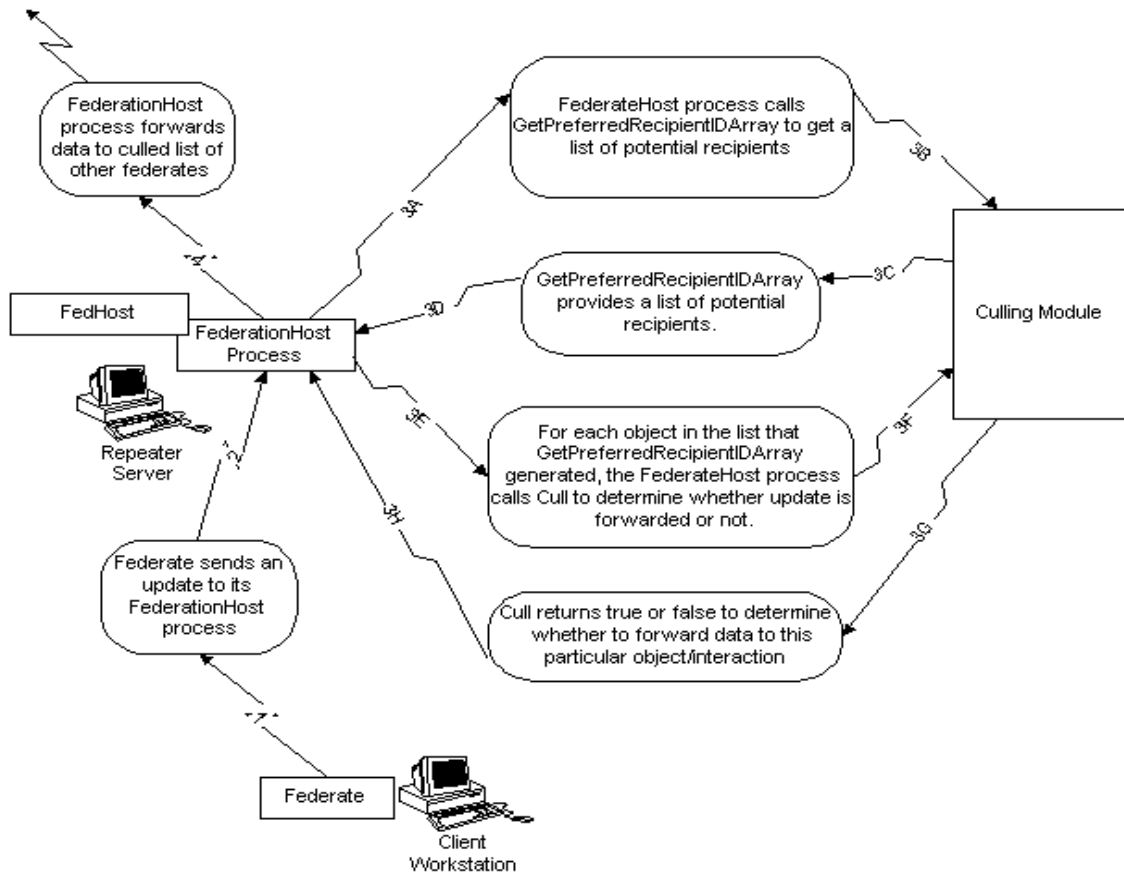


Figure 2: 2-Stage Culling

The goal should be to make this list as small as possible to reduce network traffic. Beware that this function is call a *LOT* so you want to make it as computationally light as possible!

GetPreferredRecipientIDArray 2

```

virtual bool
CAAttributeSet::GetPreferredRecipientIDArray(CSortedDWOR
DArray &RecipientIDArray, CAAttributeSet
*pCullingAttributeSet)

```

Override this function if the culling is based on another attribute set class. pCullingAttributeSet is associated with this class. This function should copy into RecipientIDArray a list of attribute set handles whose owners should discover this attribute set. Note that discovery does not guarantee that these owners will receive updates unless the Cull function returns true also. Return false if all subscribers should discover this attribute set. Be very careful when returning false since it disables culling. When the FedHost calls this function, it is telling the CAAttributeSet:

“Here is the attribute set that you are associated with. Use it to give me a list of handles for all the attribute sets that should see you”

This function does culling. It is actually the first of a two-stage culling process. `GetPreferredRecipientIDArray` (both versions) allows you to construct a list of other objects that should get this object’s updates. The goal should be to make this list as small as possible to reduce network traffic. Beware that this function is call a *LOT* so you want to make it as computationally light as possible!

Cull

```
virtual bool CAttributeSet::Cull(const CAttributeSet
    *pAttributeSet)
```

Override this function if the culling is not based on any other attribute set class. Return true if the owner of *pAttributeSet* needs to be updated. When the `FedHost` calls your `Cull` function above, it is asking the `CAttributeSet` instance a question

“should this `pAttributeSet` object be able to see you?”

Your culling rule will normally answer this question by comparing values in the *pAttributeSet* to the values in the *this* pointer.

This function does culling. It is the second of a two-stage culling process. `Cull` (both versions) allows you to compare this object to another object to determine whether communication should flow between them. The object received as an argument to this function has already passed the first stage of culling (`GetPreferredRecipientIDArray`). `Cull` gives you a mechanism for a more precise comparison of attribute sets. The goal should be to return *false* as often as possible to reduce network traffic. Beware that this function is call a *LOT* so you want to make it as computationally light as possible!

Cull 2

```
virtual bool CAttributeSet::Cull(const CAttributeSet
    *pAttributeSet, const CAttributeSet
    *pCullingAttributeSet1, const CAttributeSet
    *pCullingAttributeSet2)
```

Override this function if the culling is based on another attribute set. This is set on the client using the function `CImportExportTable::SetAttributeSetCullingHandle` in the *Openskies IET API*. `pCullingAttributeSet1` is associated with this class, and `pCullingAttributeSet2` is associated with `pAttributeSet`. Return true if the owner of *pAttributeSet* needs to be updated. When the `FedHost` calls this `Cull` function, it is asking the `CAttributeSet` instance a question

“here is another attribute set *pAttributeSet*. It is associated with the attribute set *pCullingAttributeSet2*. You are associated with *pCullingAttributeSet1*. Should this object be able to see you?”

This function does culling. It is the second of a two-stage culling process. Cull (both versions) allows you to compare this object to another object to determine whether communication should flow between them. The object received as an argument to this function has already passed the first stage of culling (*GetPreferredRecipientIDArray*). Cull gives you a mechanism for a more precise comparison of attribute sets. The goal should be to return *false* as often as possible to reduce network traffic. Beware that this function is call a *LOT* so you want to make it as computationally light as possible!

FED File Extensions For Culling Rules

In order to do Culling, Cybernet has added certain extensions to the FED file. One is the culling rules section:

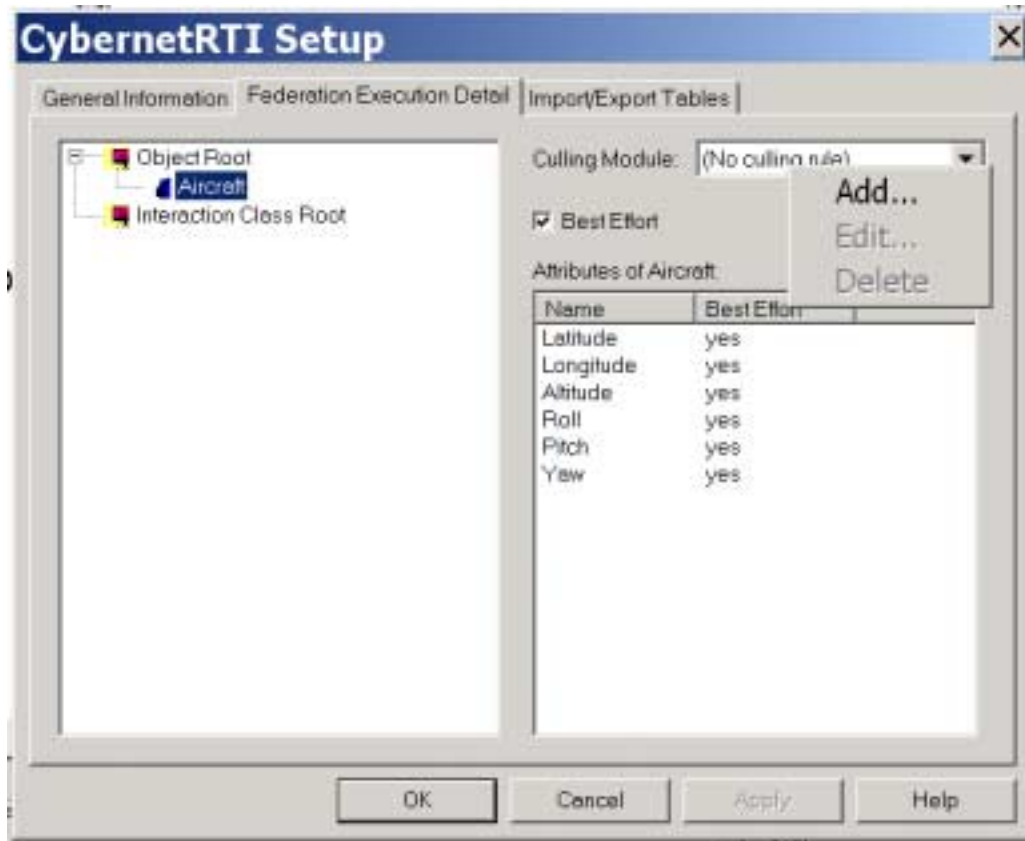
```
(FED
  (Federation OpenSkies)
    (FEDversion v1.3)
    (culling-rules
      (module main)
      (module CullingRulesII.so)
    )
  (spaces
)
(objects
  (class ObjectRoot
    ... ..
```

In this example, we have two culling modules listed in the “culling-rules” section, main and CullingRulesII.so. The module “main” is an internal module, therefore it does not have the “.so” file extension.

Besides listing culling modules in the “culling-rules” section, you must also list the specific culling modules in each attribute set that requires culling:

```
(class Aircraft
  (attribute Latitude best_effort receive)
  (attribute Altitude best_effort receive)
  (attribute Longitude best_effort receive)
  (attribute Roll best_effort receive)
  (attribute Pitch best_effort receive)
  (attribute Yaw best_effort receive)
  (culling-module CullingRulesII.so)
  ... ..
```

Please note that you do not need to edit the FED file directly as described above. In the CybernetRTI Setup tool, you can setup FED file for culling on the second tab:



First you select an attribute set, say “Aircraft”, and then you can select a culling module for “Aircraft” from the combo box. If the desired culling module is not yet in the list, you may right-click on it to display a pop-up menu, and select “Add...”. Menu items “Edit...” and “Delete” are enabled when such actions are allowed on a selected culling module.