Introduction to the HDK

by

Gerome Mortelecque

gerome@sidefx.com

with the collaboration of

Mark Elent

This is a simple introduction to the Houdini Development Kit. We will have a look at the following points:

- What is the HDK and its use.
- Where to get it and installation.
- Compiling a standalone application using the HDK
- Compiling and installing a DSOs by creating a new SOP
- How to create a new command for Hscript
- How to create a new expression.
- Extra.

The Houdini Development Kit (HDK) allows to create custom tools that can not be developed in VEX or Hscript.
For example:

- An expression function.
- A Hscript command.
- An output to a non-standard renderer.
- A custom lighting or atmospheric effects to a renderer.
- Operators that need to have access to disk files of any type.

Finally the HDK gives full access to other parts of Houdini (ie. Libraries: UT, GU, etc).

Some examples of use of HDK in the industry:

- Custom image support: Rhythm and Hues has a proprietary image format.
- Interface to other libraries: Fluid simulation.
- Import & Export of geometry: RIB Glue by Martian Toolz.
- Custom deformers where VEX is not flexible enough.

Getting and installing the HDK

There no need of a different license than the one that comes with Houdini Master. Once you have installed Houdini, do the following:

Linux:

1. Do not forget to source *houdini_setup* or *houdini_setup_bash* according to your shell.
2. Go to *$HFS/bin* where $HFS (Houdini root directory) is set up by the *houdini_setup* script (ie:/opt/hfs80_xxx). Run *hdkinstall*. It will unpack the header and install some executables.
3. When installing the HDK it should tell you the status of the installation and also the requirement of *g++* to use (ie: *g++2.96)*.

Windows:

1. Once Houdini installed, from the 'Start menu' execute *hdkinstall*.
2. Make sure you have the same compiler as returned by *hdkinstall*.

First Example: Standalone application.

We start with a simple example. We have a main() function that calls a callback function to create an IsoSurface, in our case a sphere.

```
// we need to include some Houdini libraries, in this case, UT_Math.h a math library as we will
use vectors and GU_Detail.h as we will build a geometry.
#include <UT/UT_Math.h>
#include <GU/GU_Detail.h>

// Callback function sphere() needed by the method polyIsoSurface.
static float
sphere(const UT_Vector3 &p)
{
   float       x, y, z;
   x = p.x();
   y = p.y();
   z = p.z();

    //  The equation of a sphere of radius 1 centered at the origin is given
    //  in Cartesian coordinates by:
    //  x*x + y*y + z*z = 1 therefore:
   return x*x + y*y + z*z- 1;
}

int
main()
{
   // Declare a gdp(geometry)
   GU_Detail                gdp;
   // Declare a bounding box that will contain our sphere.
   UT_BoundingBox       bbox;
   bbox.initBounds(-1, -1, -1);
   bbox.enlargeBounds(1, 1 , 1);
   // we create our geometry by calling the method polyIsoSurface().
   // See GU_Detail.h for what are the other methods that can be called
   // or what  parameters the method takes. Here we need an array of points,
   // a bounding box, divisions  X, Y and Z
   gdp.polyIsoSurface(sphere, bbox, 20, 80, 20);
   // Save the gdp to the disk as a bgeo file.
   gdp.save("iso.bgeo");
   return 0;
}
```

To compile do the following:

```
hcustom -i . -s giso.C
```

The *-i* option tells where to install the compiled application, in this case in the current directory. By default it will install it in *~/houdiniX.X/dso/* directory. The *-s* option tells the compiler that it will be a standalone application.
Run *giso.* It should create a file called *iso.bgeo*. Start a Houdini new session and put down a *file* SOP, load the *iso.*bgeo file and you should see the sphere.

You can change some parameters like the radius of the sphere, for example in the Sphere method:

```
return x*x + y*y + z*z- 2;
```

will make a sphere of radius 2 units. Do not forget to increase the bounding box too.

```
   bbox.initBounds(-2, -2, -2);
   bbox.enlargeBounds(2, 2 , 2);
```

Otherwise your sphere will be clipped.

You can change other parameters such as the *divx*, *divy* and *divz* in the last arguments of *polyIsoSurface()*. To find out what are the arguments for this method, have a look at the header *GU_Detail.h*. For this you can use *hcbrowser* or go to *$HFS/toolkit/include/GU.*
You will see the following:

```
    // poly-iso surfaces
    GU_PrimPoly    *polyIsoSurface(float (*ptOutside)(const UT_Vector3 &),
                                        const UT_BoundingBox &bbox,
                                        int xdiv, int ydiv, int zdiv);
```

Compile again using the above command, and run *giso* and in Houdini, press the button *reload the geometry* of the *File* SOP.

Add the following lines after *gdp.save("iso.bgeo");*

```
   gdp.clearAndDestroy();
   gdp.polymeshCube(4,4,4);
   gdp.save("cube.bgeo");
```

The method *clearAndDestroy()* will clear the geometry from the sphere. Then *polymeshCube()* will create a cube (4x4x4 divisons) then save the geometry to a bgeo file. You can control more parameters when using *polymeshCube().* If you do not include the method *clearAndDestroy(),* the previous geometry, in this case the sphere, will not be deleted and *cube.bgeo* will be a cube in a sphere (switch to wireframe mode to see this result).

Creating a new SOP.

We will take the previous example from the standalone application and we will turn it into a SOP. It will be equivalent of the *Sphere SOP* but in a very simple way. We will parameterize the divisons. Then we will add more parameters in order to choose between a sphere and a cube and control the size of the later. We will also have a look how to disable a parameter.

Below this is the code of SOP_ *mySphere.h* and SOP_*mySphere.C*, is commented to help you to understand the work flow:

- *SOP_mySphere.h*

```
/*
 * PROPRIETARY INFORMATION.  This software is proprietary to
 * Side Effects Software Inc., and is not to be reproduced,
 * transmitted, or disclosed in any way without written permission.
 *
 * Produced by:
 *      Side Effects
 *      123 Front St. West,
 *      Suite 1401
 *      Toronto, Ontario
 *      Canada M5J 2M2
 *      Tel: (416) 504 9876
 *
 *   NAME:      SOP library (C++)
 *
 *   COMMENTS:  A very Dummy SOP doing nothing
 */

#ifndef __SOP_mySphere_h__
#define __SOP_mySphere_h__

#include <SOP/SOP_Node.h>

class SOP_mySphere : public SOP_Node
{
public:
             SOP_mySphere(OP_Network *net, const char *name, OP_Operator *op);
   virtual ~SOP_mySphere();
```

```
    static   PRM_Template  myTemplateList[];
    static   OP_Node      *myConstructor(OP_Network*, const char *,
                                 OP_Operator *);

protected:

    virtual OP_ERROR           cookMySop(OP_Context &context);

    // function declarations to evaluate the div parameters.
    int DIVX()        { return evalInt( 0, 0, 0 ); }
    int DIVY()        { return evalInt( 0, 1, 0 ); }
    int DIVZ()        { return evalInt( 0, 2, 0 ); }
};
```

- *SOP_mySphere.C*

```
/*
 * PROPRIETARY INFORMATION.  This software is proprietary to
 * Side Effects Software Inc., and is not to be reproduced,
 * transmitted, or disclosed in any way without written permission.
 *
 * Produced by:
 *      Side Effects
 *      123 Front St. West,
 *      Suite 1401
 *      Toronto, Ontario
 *      Canada M5J 2M2
 *      Tel: (416) 504 9876
 *
 *   NAME:   SOP library (C++)
 *
 *   COMMENTS:           A simple SOP to create a sphere and control the divisions
 */

// Some headers are needed
#include <UT/UT_DSOVersion.h>      // every DSO needs this header.
#include <PRM/PRM_Include.h>       // we will include some parameters
#include <OP/OP_Operator.h>        // we are creating a new OP
#include <OP/OP_OperatorTable.h>   // and we need to register it in Houdini.

// same headers are needed as for the standalone application.
#include <UT/UT_Math.h>
#include <GU/GU_Detail.h>

#include "SOP_mySphere.h"
```

```
// Where the new Operator will be defined.
// new entries to the operator table for a given type of network.
// Each entry in the table is an object of class OP_Operator which
// basically defines everything Houdini requires in order to create nodes of the new type
void
newSopOperator(OP_OperatorTable *table)
{
   table->addOperator(new OP_Operator(
                 "mysphere",  // The short operator name need for Houdini
                 "mySphere",  // Readable version of the name(for menus and such)
                 SOP_mySphere::myConstructor,    // method which constructs nodes
                                                 // of this type
                 SOP_mySphere::myTemplateList, // A list of the templates
                                                 // defining the parameters to this operator.
                 0,      // Minimum  number of inputs required
                 0,      // maximum number of inputs required
                 0));    // A list of any local variables used by the operator
}

// parameters labelling
static PRM_Name        names[] =
{
   // division is short parameter name recognized by Houdini
   // Division X/Y/Z is a more readable label for the user.
   PRM_Name("division",  "Division X/Y/Z"),
};

// declare some defaults values for the divisions parameter.
// If nothing is set, the default values will be zero.
static PRM_Default     divDefaults[] =
{
   // we set a default value for each component.
   PRM_Default(20), PRM_Default(40), PRM_Default(20)
};

// we can limit the user to set the divisions according to a range.
// Check PRM_Range.h for the options.
PRM_Range  divRange(PRM_RANGE_RESTRICTED, 0, PRM_RANGE_RESTRICTED, 100);

// TemplateList will be where all the parameters are defined.
PRM_Template
SOP_mySphere::myTemplateList[] = {
   // PRM_XYZ: Parameter type XYZ see $HFS/toolkit/html/op/prm.html
   // for the different types available
   // 3: it has 3 components, X, Y and Z.
   // &names[0]: we will associate the first PRM_NAME declared above.
   // PRM_Template can take more parameters, see PRM_Template.h
   PRM_Template(PRM_XYZ,    3, &names[0], divDefaults, 0, &divRange ),
};

// Node Constructor
OP_Node *
SOP_mySphere::myConstructor( OP_Network *net, const char *name, OP_Operator *op )
{
    return new SOP_mySphere(net, name, op);
}

// Constructor for mySphere
SOP_mySphere::SOP_mySphere( OP_Network *net, const char *name, OP_Operator *op )
    : SOP_Node(net, name, op)
{}

// Destructor
SOP_mySphere::~SOP_mySphere() {}

// We include our sphere function here.
static
float sphere(const UT_Vector3 &p)
{
    float        x, y, z;
    x = p.x();
    y = p.y();
    z = p.z();

    return x*x + y*y + z*z- 1;
}

// Main where all the "cooking" will happen.
OP_ERROR
SOP_mySphere::cookMySop(OP_Context &context)
{
    // we include what was in the main() here.
    UT_BoundingBox        bbox;

    // This time we have divx, divy and divz as parameters.
    int                   divx, divy, divz;

    // because every SOP will have a gdp declared already
    // we do not need to re-declare it but it needs to be
    // initialized using clearAndDestroy().
    gdp->clearAndDestroy();

    // evaluate values from the fields.
    divx = DIVX();
    divy = DIVY();
    divz = DIVZ();
    bbox.initBounds(   -1, -1, -1 );
    bbox.enlargeBounds( 1,  1,  1 );

    // build the geometry.
    gdp->polyIsoSurface(sphere, bbox, divx, divy, divz);

    return error();
}
```

Compile *hcustom SOP_mySphere.C*. *Hcustom* will install the DSO in your home directory *houdini/dso*. When you launch Houdini, should see in SOP an operator called *mySphere.*

Now, we will add some extra parameters given us the choice between a sphere and a cube. Again we will control the number of divisons but for the cube, we will control its size. But the *size* parameter will be disable for the sphere.
Below it is the commented source code to create a SOP *mySphereCube*.

- *SOP_mySphereCube.h*:

```
/*
 * PROPRIETARY INFORMATION.  This software is proprietary to
 * Side Effects Software Inc., and is not to be reproduced,
 * transmitted, or disclosed in any way without written permission.
 *
 * Produced by:
 *      Side Effects
 *      123 Front St. West,
 *      Suite 1401
 *      Toronto, Ontario
 *      Canada M5J 2M2
 *      Tel: (416) 504 9876
 *
 *   NAME:   SOP library (C++)
 *
 *   COMMENTS:            A SOP that allows use to create a sphere or a cube
 *              according to the will of the user.
 */

#ifndef __SOP_mySphereCube_h__
#define __SOP_mySphereCube_h__
```

```
#include <SOP/SOP_Node.h>

class SOP_mySphereCube : public SOP_Node
{
public:

                SOP_mySphereCube(OP_Network *net, const char *name, OP_Operator *op);
   virtual ~SOP_mySphereCube();
   static   PRM_Template  myTemplateList[];
   static   OP_Node          *myConstructor(OP_Network*, const char *,
                                                       OP_Operator *);

protected:

   virtual unsigned                          disableParms();
   virtual OP_ERROR                          cookMySop(OP_Context &context);

   // declaration of the functions to evaluate the div parameters.
   int     DIVX()        { return evalInt(   0, 0, 0 ); }
   int     DIVY()        { return evalInt(   0, 1, 0 ); }
   int     DIVZ()        { return evalInt(   0, 2, 0 ); }

   int               OPTIONS()      { return evalInt(   1, 0, 0 ); }

   // we evaluate this parameter using the time, so we can animate it
   float   SIZE( float t ) { return evalFloat( 2, 0, t ); }
};
#endif
```

- *SOP_mySphereCube.C*:

```
/*
* PROPRIETARY INFORMATION.  This software is proprietary to
* Side Effects Software Inc., and is not to be reproduced,
* transmitted, or disclosed in any way without written permission.
*
* Produced by:
*     Side Effects
*     123 Front St. West,
*     Suite 1401
*     Toronto, Ontario
*     Canada M5J 2M2
*     Tel: (416) 504 9876
*
*   NAME:    SOP library (C++)
*
*   COMMENTS:            A SOP that allows use to create a sphere or a cube
*            according to the will of the user.
*            It is based on the previous example mySphere.C. We had
*            few options to create a cube, and some extra parameters
*            to choose between a sphere and a cube and control the size
*            of the cube. We also have an example to disable a parameter
*/

// Some headers are needed
#include <UT/UT_DSOVersion.h>      // every DSO needs this header.
#include <PRM/PRM_Include.h>       // we will include some parameters
#include <OP/OP_Operator.h>        // we are creating a new OP
#include <OP/OP_OperatorTable.h>   // and we need to register it in Houdini.

// headers we needed for the standalone application.
#include <UT/UT_Math.h>
#include <GU/GU_Detail.h>

#include "SOP_mySphereCube.h"

// Where the new Operator will be defined. New entries to the operator table for a
// given type of network. Each entry in the table is an object of class OP_Operator
// which basically defines everything Houdini requires in order to create nodes of the
// new type
void
newSopOperator(OP_OperatorTable *table)
{
    table->addOperator(new OP_Operator(
                    "mysphere",  // The short operator name need for Houdini
                    "mySphere",  // Readable version of the name(for menus and such)
                    SOP_mySphere::myConstructor,    // method which constructs nodes
                                                    // of this type
                    SOP_mySphere::myTemplateList, // A list of the templates
                                                  // defining the parameters to this operator.
                    0,      // Minimum  number of inputs required
                    0,      // maximum number of inputs required
                    0));    // A list of any local variables used by the operator
}

// parameters labelling
static PRM_Name        names[] =
{
    // division = short parameter name recognized by Houdini
    // Division X/Y/Z is a more readable label for the user.
    PRM_Name("division",  "Division X/Y/Z"),
    // label for the menu to switch between a sphere and a cube
    PRM_Name("geometry",  "Geometry"),
    PRM_Name("size",      "Size"),
};
```

```
// declare some defaults values for the divisions parameter.
// If nothing is set, the default values will be zero.
static PRM_Default     divDefaults[] =
{
   // we set a default for each component.
   PRM_Default(20), PRM_Default(40), PRM_Default(20)
};
// Create items for the menu
static PRM_Name optMenuName[] = {
   PRM_Name("sphere", "Sphere"),
   PRM_Name("cube",   "Cube"),
};

// Create the actual menu using the item declared above
static PRM_ChoiceList optMenu((PRM_ChoiceListType)
                                          (PRM_CHOICELIST_EXCLUSIVE |
                                           PRM_CHOICELIST_REPLACE), optMenuName);


// we can limit the user to set the divisions according to a range.
// Check PRM_Range.h for the options.
PRM_Range divRange( PRM_RANGE_RESTRICTED,     0,
                              PRM_RANGE_RESTRICTED, 100 );

// for the size we restrict the minimum only
PRM_Range  sizeRange(PRM_RANGE_RESTRICTED, 0.001 );

// TemplateList will be where all the parameters are defined.
PRM_Template
SOP_mySphereCube::myTemplateList[] = {

   // PRM_XYZ: Parameter type XYZ see $HFS/toolkit/html/op/prm.html
   // for the different types available
   // 3: it has 3 components, X, Y and Z.
   // &names[0]: we will associate the first PRM_NAME declared above (index 0 and so on).
   // PRM_Template can take more parameters, see PRM_Template.h
   PRM_Template(PRM_XYZ,    3, &names[0], divDefaults, 0, &divRange ),
   // The menu
   PRM_Template(PRM_ORD,    1, &names[1], PRMzeroDefaults, &optMenu),
   // The size - PRMoneDefaults a variable declared PRM_Shared.h
   PRM_Template(PRM_FLT,    1, &names[2], PRMoneDefaults,  0, &sizeRange ),

};
```

```
// We will disable the parameter size when the sphere is chosen.
unsigned
SOP_mySphereCube::disableParms()
{
    unsigned changed = 0;

    changed  = enableParm(2, !OPTIONS());  // Turn size off if option is 0 (sphere)
    changed += enableParm(2,  OPTIONS()); // Turn size on  if option is 1 (cube)

    return changed;
}

// Node Constructor
OP_Node *
SOP_mySphereCube::myConstructor( OP_Network *net, const char *name, OP_Operator *op )
{
    return new SOP_mySphereCube(net, name, op);
}

// Constructor for mySphereCube
SOP_mySphereCube::SOP_mySphereCube( OP_Network *net, const char *name, OP_Operator
*op )
    : SOP_Node(net, name, op)
{}

// Destructor
SOP_mySphereCube::~SOP_mySphereCube() {}

// We include our sphere function here.
static
float sphere(const UT_Vector3 &p)
{
    float       x, y, z;
    x = p.x();
    y = p.y();
    z = p.z();

    return x*x + y*y + z*z- 1;
}

// Main where all the "cooking" will happen.
OP_ERROR
SOP_mySphereCube::cookMySop(OP_Context &context)
{
    // we include what was in the main() here.

    UT_BoundingBox       bbox;
    // This time we have divx, divy and divz as parameters.
    int                 divx, divy, divz, option;
    float               size, now;

     // This is the way to get the current time.
    now = context.myTime;

    // Evaluate the values from the fields.
    divx = DIVX();
    divy = DIVY();
    divz = DIVZ();

    // Evaluate the option from the menu
    // Sphere will return 0
    // Cube   will return 1
    // if we have more options they will be indexed 0,1,2,3..n etc
    option = OPTIONS();

    switch( option )
    {
    case 0:  // the sphere is chosen
                // Initialize the gdp to make sure we will have a clean
                // geometry every time we change the menu
                gdp->clearAndDestroy();
                bbox.initBounds(   -1, -1, -1 );
                bbox.enlargeBounds( 1,  1,  1 );
                // build the geometry.
                gdp->polyIsoSurface(sphere, bbox, divx, divy, divz);
                break;

    case 1: // the cube is chosen.

                gdp->clearAndDestroy();
                / / Evaluate size;
                size  = SIZE( now );

                // create a cube with the given divisions and size.
                // See polymeshCub() in GU_Detail.h to see what arg you
                // can modify
                gdp->polymeshCube( divx, divy, divz,
                                            -size, size,  // xmin and xmax
                                            -size, size,  // ymin and ymax
                                            -size, size); // zmin and zmax
                break;
    }

    return error();
}
```

Creating a new Hscript command

This part is based on the *date* command example taken from the page *$HFS/toolkit/html/cmd/command.html.*

The built-in Houdini commands can listed by typing *help* in the textport. This section will show you how to add your own custom commands to Houdini.
As an example, suppose we want to add the command *date* which simply prints out the current date. We want *date* to behave as follows:

```
/ -> date
Mon Mar 10 15:16:18 EST 1997
```

For pedagogical reasons, let's also make *date* accept an option -*s num_spaces* which left pads the returned date string with the specified number of spaces.

```
/ -> date -s 5
     Mon Mar 10 15:16:18 EST 1997
```

- *date.C*:

```
/*
 * PROPRIETARY INFORMATION.  This software is proprietary to
 * Side Effects Software Inc., and is not to be reproduced,
 * transmitted, or disclosed in any way without written permission.
 *
 * Produced by:
 *      Side Effects
 *      123 Front St. West,
 *      Suite 1401
 *      Toronto, Ontario
 *      Canada M5J 2M2
 *      Tel: (416) 504 9876
 *
 *   COMMENTS:           Hscript command 'date' to return the date from the system.
 *
 */

#include <UT/UT_DSOVersion.h>
// header for any new command
#include <CMD/CMD_Manager.h>
```

```
#include <CMD/CMD_Args.h>

// we need this library which will return the system time.
#include <time.h>

// The callback function should be declared static to avoid possible
// name clashes with other Houdini functions. The callback takes one
// argument, a reference to a CMD_Args object. This object has two main
// functions: to provide access to the command line options for the
// command and to provide an output stream so you can return output to
// the Houdini shell.
static
void cmd_date( CMD_Args &args ) {
    time_t      time_struct;
    char        *time_string;

    int         i;

    // the 's' option left pads with some spaces,     // just for an example
    if( args.found('s') )
    {
                    int num_spaces =args.iargp( 's' );
                    for( i = 0; i < num_spaces ; i++ )
                        args.out() << " ";
    }

    // call the standard C function 'time'.
    // see 'man time' for details
    time_struct = time(0);
    time_string = ctime( &time_struct );
    // printout the date to the args out stream
    args.out() << time_string;
}

//  this function gets called once during Houdini initialization
void CMDextendLibrary( CMD_Manager *cman )
{

    // install the date command into the command manager
    // name              = "date",
    // options_string = "s:",
    // callback          = cmd_date
    cman->installCommand("date", "s:", cmd_date );
}
```

To compile: *hcustom date.C* and it will be installed in the dso path and the command *date* should appear in the textport when you type *help*.

Creating a new expression function.

Adding a custom expression can be done anywhere and is not confined to a particular part of the code. For example, if a custom SOP needs a specific expression function, that function can be added in the SOP code.
This document deals with the mechanism for adding custom expressions using Command Library extensions.

We will create two simple expressions *min* which will return the smallest of two numbers and *max* which will return the greatest number of two numbers.

- *Funtion.C:*

```
/*
 * PROPRIETARY INFORMATION.  This software is proprietary to
 * Side Effects Software Inc., and is not to be reproduced,
 * transmitted, or disclosed in any way without written permission.
 *
 * Produced by:
 *      Side Effects
 *      123 Front St. West,
 *      Suite 1401
 *      Toronto, Ontario
 *      Canada M5J 2M2
 *      Tel: (416) 504 9876
 *
 *   COMMENTS:            Expression functions min and max.
 *
 */

// As for command we needed basic headers for creating a new expression
#include <math.h>
#include <UT/UT_DSOVersion.h>
#include <EXPR/EXPR.h>
#include <CMD/CMD_Manager.h>
```

```
// Callback function to evaluate the max of two numbers
static void
fn_max(EV_FUNCTION *, EV_SYMBOL *result, EV_SYMBOL **argv, int)
{
   // grab the argument 1 and 2 and compare them
   if (argv[0]->value.fval > argv[1]->value.fval)
               result->value.fval = argv[0]->value.fval;
   else
               result->value.fval = argv[1]->value.fval;
}

// Callback function to evaluate the minimum of two numbers
static void
fn_min(EV_FUNCTION *, EV_SYMBOL *result, EV_SYMBOL **argv, int)
{
   if (argv[0]->value.fval > argv[1]->value.fval)
               result->value.fval = argv[1]->value.fval;
   else
               result->value.fval = argv[0]->value.fval;
}

// A couple of defines to make life a lot easier for us
#define EVFEV_TYPEFLOAT // return type in this case a float

// we declared the array of arguments that the function will take.
static int      floatArgs[] = { EVF, EVF };

// We are creating a table with the function we want to register.
static EV_FUNCTION funcTable[] = {

   // register the function max and min:
   // 0         is the flag, if your function is time dependent, set it to anything but 0.
   // "max"     is the name of the function.
   // 2         is the number of arguments.
   // EVF       is the return type.
   // floatArgs is the argument type in our case floats.
   // fn_max    is the callback
   EV_FUNCTION(0, "max",         2, EVF,     floatArgs,    fn_max),
   EV_FUNCTION(0, "min",         2, EVF,     floatArgs,    fn_min),
   EV_FUNCTION(),
};
```

```
// Initialization of the expressions.
void
CMDextendLibrary(CMD_Manager *)
{
   int                              i;
   // go through the items of our table and initialize them
   for (i = 0; funcTable[i].getName(); i++)
                ev_AddFunction(&funcTable[i]);
}
```

Again use *hcustom function.C* to compile and *min* and *max* will be installed.

Type the following command in the textport:

-> *echo `min(2,-3)`*

-*3* should be returned.

More information can be found in the following help page: *$HFS/toolkit/html/cmd/expr.html*

EXTRA

This concludes this introduction. Below are some useful tips, more can be found in the FAQ page of the HDK documentation.

- *My SOP doesn't load into Houdini.*

  This is most likely due to a DSO error. An external library may not be referenced properly, there might be missing symbols in the DSO etc. Set the following variable:

  setenv HOUDINI_DSO_ERROR.

  This will cause DSO errors to be printed to the window shell when the DSO is accessed. They should appear if you run *hscript* or *Houdini.*

- *How to install icons?*

To install an icon for your new OP you will need to install an image (ie, jpg or any format supported by Houdini) into the directory *$HOME/houdini(Ver)/config/Icons/* where the name of the image should be the same as your operator, for example in our case *SOP_mySphere.so* the image should be *SOP_mySphere.jpg*.
Another way to have an icon is to use *gicon;* it will convert houdini geometry files into iconsmith-style icons that can be used in houdini. Create a geometry, export it as a *geo* file and then type *gicon myicon.geo.* Again the name of your icon should be the same as the name of your operator with the extension *.icon.* See *gicon* for more information.

- *Using Doxygen to make a documentation from the headers.*

It could be useful to have the HDK classes as a Doxygen document, for example a web page. If you have *doxygen* installed you can have a config file as below and type *doxygen config.file.* A *html* directory will be created and a series of web pages. Load *index.html* in a browser.

Config.file for Doxygen.

```
PROJECT_NAME                    = HDK Headers
INPUT                           =   ./
OUTPUT_DIRECTORY                =  ./html
RECURSIVE                       = YES
WARNINGS                        = YES
FILE_PATTERNS                   = *.h
HAVE_DOT                        = YES
GRAPHICAL_HIERARCHY             = YES
CLASS_GRAPH                     = YES
INCLUDE_GRAPH                   = YES
COLLABORATION_GRAPH             = YES
CALL_GRAPH                      = YES
GENERATE_HTML                   = YES
HAVE_DOT                        = YES
```