Autodesk®

Developer Technical Services

Assignment - Miscellaneous Tools

Topics Covered

- Manipulate Dependency Graph through MDGModifier class
- Implement scene callback functions with MSceneMessage class

setUpTransCircle Plug-in

- Topics Covered
    - Implement a custom command and use MDGModifier to manipulate Dependency Graph (such as adding nodes, building connections)
    - Support undo/redo
- Overview
    - In this exercise, we will create a custom command, and simulate the same functionality of the MEL operations we used in Lab 1 “transCircleNode” Plug-in, which set up the transCircle node.

      Here are the commands you need to simluate:

      ```
      createNode transCircle -n circleNode1;
      sphere -n sphere1 -r 1;
      sphere -n sphere2 -r 2;
      connectAttr sphere2.translate circleNode1.inputTranslate;
      connectAttr circleNode1.outputTranslate sphere1.translate;
      connectAttr time1.outTime circleNode1.input;
      ```
- Exercises
    - 1. Double click on setUpTransCircle.py, the skeleton of the setUpTransCircle has already been provided. Note that this Plug-in also includes the source code for transCircleNode.
    - 2. Implement setUpTransCircleCmd.py, add necessary function declaration and member variable declaration.

      Relevant classes and methods:

      ```
      MPxCommand:: isUndoable(), MPxCommmand::undoIt(),
      MPxCommand::redoIt()
      ```

      Since you want to support undo/redo, you may want to declare an MDGModifier object as a member variable.

- 3. Implement setUpTransCircleCmd.py, in doIt() functions, simulate the MEL commands using your member variable MDGModifier object.
  Relevant classes and methods:
  MDGModifier::createNode(), MDGModifier::renameNode()
  MDGModifier::commandToExecute()
  MDGModifier::doIt(), MDGModifier::undoIt(),
  MDGModifier::redoIt() , MDGModifier::connect()
  MGlobal::getSelectionListByName()
  MSelectionList::getDependNode()
  MFnDependencyNode::attribute()
- 4. In setUpTransCircleCmd.py, implement both initializePlugin() and uninitializePlugin() functions to handle registration and de-registration of the setUpTransCircle command.
  Relevant classes and methods:
  MFnPlugin::registerCommand()
  MFnPlugin::deregisterCommand()
  Note: you will also need to register "transCircle" node to make it work. Relevant classes and methods:
  MFnPlugin::registerNode()
  MFnPlugin::deregisterNode()

- Result
  After you finish and load the plug-in setUpTransCircle.py, execute the follow command:
  setUpTransCircle;

  You will see two nurbs sphere created and once you hit the "play" button, one sphere is rotating around the bigger sphere, you can also set the radius of the circle, and rotating speed by setting the values in attribute editor of circleNode1.

- Bonus Section

  You can use dgMod.commandToExecute() to simulate MEL command "sphere -n sphere1 -r 1;", however you can also simulate the operations from scratch. Go into the hypergraph and try to find what have happened to the Dependency Graph (for examples, what nodes have been created and what connections have been made) when you execute "sphere -n sphere1 -r 1;" and simulate those changes with MDGModifier.

  Tips: There are mainly two types of nodes created, "nurbsSurface" and "makeNurbSphere", they are connected:

  makeNurbSphere1.outputSurface→sphere1Shape.create

sceneMsgCmd Plug-in

- Topics Covered
    - How to use MSceneMessage to register callbacks watching scene operations
- Overview
    - In this exercise, we will create a custom command to watch specific scene operation messages:
      MSceneMessage::kBeforeOpen,
      MSceneMessage::kAfterNew,
      MSceneMessage::kBeforeSaveCheck
      We will implement callback functions which get trigger by those messages. We will also experiment how to abort current scene operation by using checkCallback functions.
- Exercises
    - 1. Go to "sceneMsgCmd\Exercise" folder, double click on sceneMsgCmd.py, the skeleton for this command is already implemented.
    - 2. Implement sceneMsgCmd.py, add necessary function declaration and member variable declaration.
      Relevant classes and functions:

      MPxCommand::isUndoable(), MPxCommmand::undoIt(),
      MPxCommand::redoIt()

      To keep a list of all the registered callbacks, you may want to declare a member variable to record all the callback IDs.

    - 3. Implement sceneMsgCmd.py, in redoIt() function, register callbacks for MSceneMessage::kBeforeOpen, MSceneMessage::kAfterNew, MSceneMessage::kBeforeSaveCheck
      Relevant classes and functions:

      MSceneMessage::addCallback()
      MSceneMessage::addCheckCallback()

      In the callbacks functions, just print out "The callback registered for MSceneMessage::kXXXX is executed." Also in the callback function for MSceneMessage::kBeforeSaveCheck, set "retCode" to be pointing to "false" to abort current operation.
      In undoIt() function, remove callbacks.
      In doIt() function, call redoIt().
    - 4. In sceneMsgCmd, implement both initializePlugin() and uninitializePlugin() functions to handle registration and de-registration of the sceneMsgCmd command.
      Relevant classes and methods:

      MFnPlugin::registerCommand()
      MFnPlugin::deregisterCommand()

Note: In uninitializePlugin() , don't forget to remove all the callbacks you have registered in your plug-in.

- Result

  After you finish and load the plug-in sceneMsgCmd.py, execute the follow command:

  ```
  sceneMsgCmd;
  ```

  When you open a file, there will be a print out in the output window:

  "The callback registered for MSceneMessage::kBeforeOpen is executed."

  After you create a new file, there will be a print out in the output window:

  "The callback registered for MSceneMessage::kAfterNew is executed."

  When you try to save current scene file, there will be a print out in the output window:

  "The callback registered for MSceneMessage::kBeforeSaveCheck is executed."

  There is also an error message popping up in Maya saying "File operation cancelled by user supplied callback" and the file is not saved.