Classic Scripting in Pinnacle^3

Michael Meltsner, Karl Bzdusek, Matthieu Bal, Ying Xiong
Philips Radiation Oncology Systems
May 7, 2014
Version 1.0

PHILIPS

Disclaimer

For non-clinical use

- The following presentation outlines advanced scripting techniques
- Philips officially only supports “Record and Playback” within clinical releases of Pinnacle
- The tools that follow are meant for research mode only and are not validated for use on patients

Overview

What is a Script?

- Scripting in Pinnacle^3 gives the user the ability to communicate to/from the Pinnacle core without having access to the code directly
- Each script is an ASCII file that contains a collection of messages that communicate sequentially with Pinnacle^3
- These messages allow the user to mimic interactions with mouse clicks and keyboard input
- They are analogous to macro type functions in other software like Excel

Scripting Messages

Three basic types

- Pinnacle responds to 3 basic types of Scripting Messages:

    1) **Set command**: tells Pinnacle^3 to change an underlying attribute
        – Beam angle, prescription name

    2) **Query request:** asks Pinnacle^3 to return the value of an attribute
        – Number of beams, Patient's name

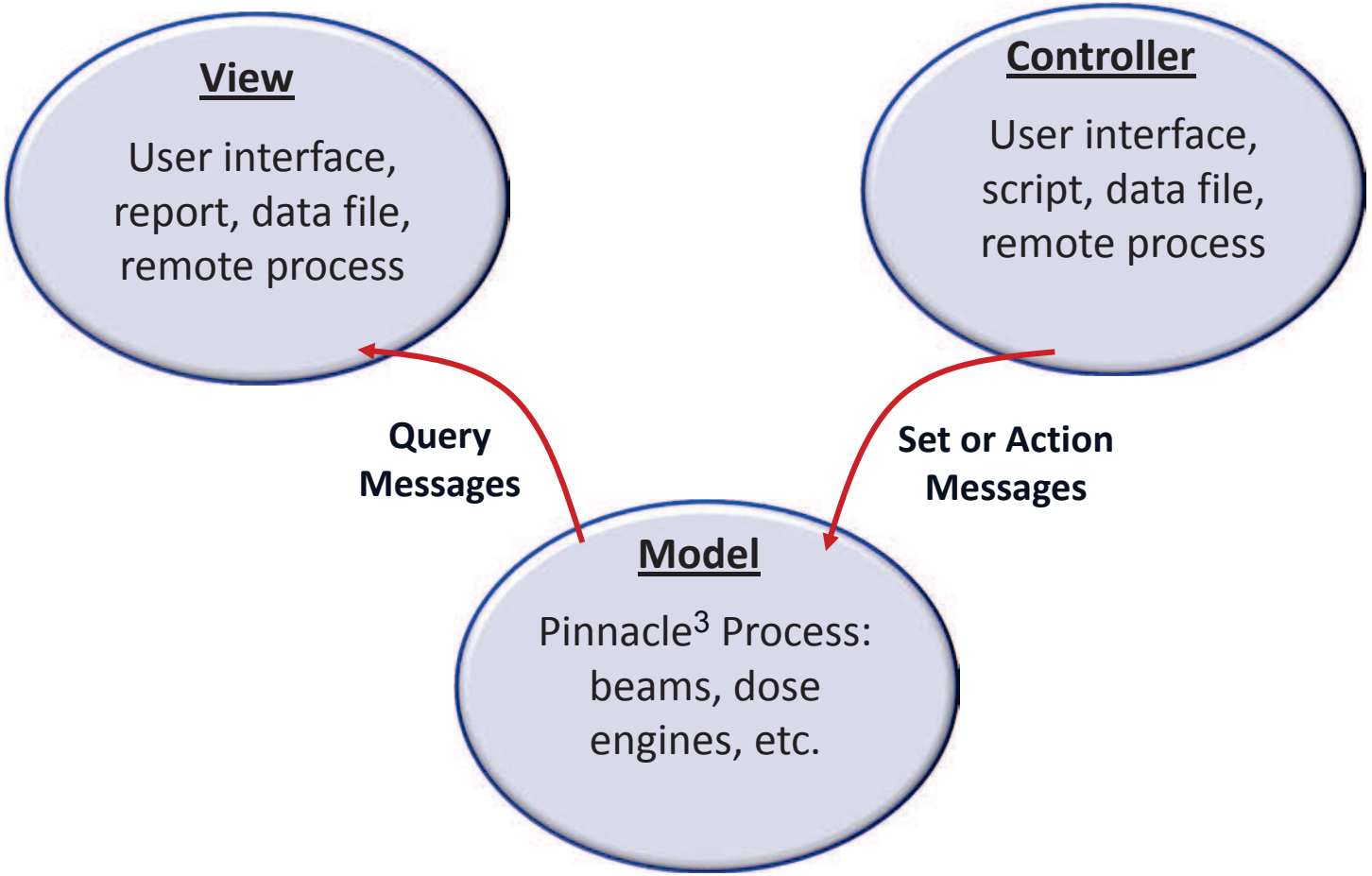
    3) **Action command**: tells Pinnacle^3 to perform a task
        – Add a beam, Delete an ROI

Pinnacle Architecture

PHILIPS

Pinnacle^3 MVC Architecture

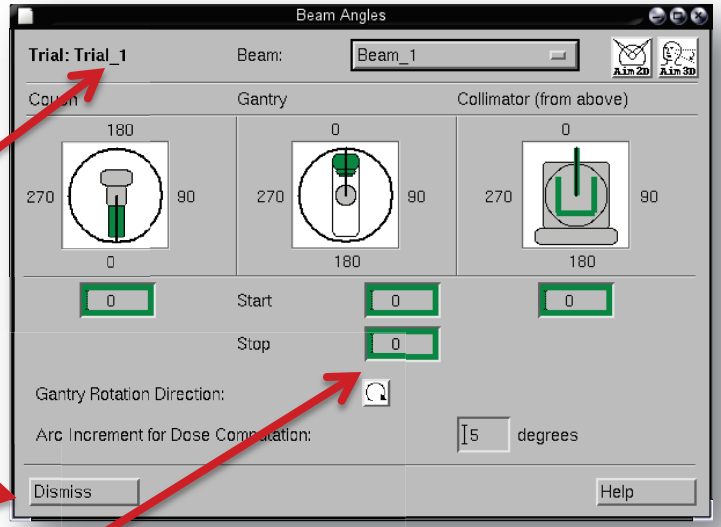
Set, Query, and Action messages



Pinnacle^3 GUI

How widgets work

- The Pinnacle^3 GUI is essentially an extensive set of messages hidden behind widgets
- Each widget in the GUI has 1 or more of the three types of messages associated with it
- The label widget displays the result of a **Query** request
- The button widget in Pinnacle^3 typically performs an **Action**
- The text box displays a **Query** request as its value and sends a **Set** message after the user enters a new value



Pinnacle^3 Internal Design

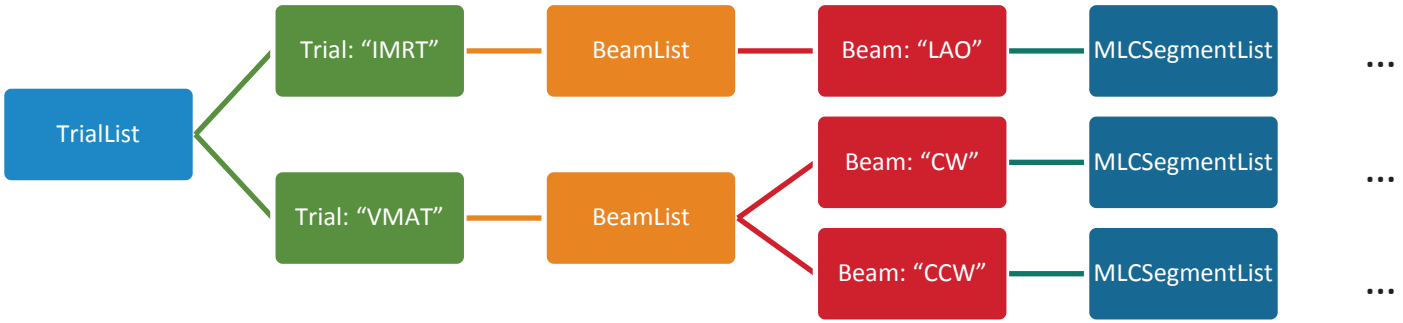
Understanding scripting messages

- Pinnacle^3 is designed in an object-oriented manner
- Every object has associated attributes and functions
- The hierarchy of the Pinnacle^3 objects is important and directly relates to the functionality of the software and hence is mimicked in the scripting language
- Objects of the same class are stored in a container simply called **ObjectList**.
  - Each object within that list may contain subobjects including sublists

Pinnacle^3 Internal Design

Hierarchy

- The hierarchy of Pinnacle^3 (and thus Scripting) follows a logical sense of design
  - For example, each plan may contain multiple **Trials**. Each **Trial** may contain multiple **Beams**. Each **Beam** may contain multiple **MLC Leaves** and so on.



- You can **Set, Query, or Act** on any object by following the hierarchical structure of the Pinnacle^3 design starting from the root level, in this case **TrialList**.

Pinnacle^3 Internal Design

Root level objects

- The top most objects are called “Root level”
  - Examples of Root Level **ObjectLists**
    - TrialList - Root level container of trials
    - RoiList - Root level container of ROIs
    - DVHList - Root level container of DVH
    - VolumeList - Root level container of volumes
    - ViewWindowList - Root level container of view windows
    - WindowList - Root level container of windows

- An example of a non-root container follows from the previous slide
  - TrialList.IMRT.BeamList …- Each **Trial** contains a **BeamList**

- **Note: Not all Root level objects are ObjectLists**
  - PlanInfo - Root level object that holds information about the plan/patient

Message Syntax and Examples

Basic Syntax

Scripting rules

- Each message within the script must end with a semi colon
- With some exceptions, there is one “=“ per line for each message
- All script commands are case sensitive
- The script may contain comments denoted by starting a line with a “//”

```
//This is my first script!
```

Basic Syntax

Set message

- The set message sends information to Pinnacle^3
- It contains the attribute you want to set on the left and its new value on the right

<SetMessage> = <Value>;

***<Value>* can be one of the following:**

- **A string enclosed in double quotes**
- **A floating point or integer number**
- **A Query message**

- **Set** the color of the ROI named “Bladder” to green

  RoiList.Bladder.Color = “green”;

Basic Syntax

Query request

- The query request asks Pinnacle^3 to return some information about an object
- The query request is most commonly used as input for a **Set** message

<SetMessage> = <Query Request>;

- **Set** gantry of "Beam_1" equal to gantry of beam "AP" via **Query**

TrialList.IMRT.BeamList.Beam_1.Gantry =
TrialList.IMRT.BeamList.AP.Gantry;

PHILIPS

Basic Syntax

Action message

- The **Action** command tells Pinnacle^3 to perform a task
- The **Action** message ignores the right side of the equals sign, but still requires one to properly execute

<ActionMessage> = "";

where *"" is just an empty string indicated by quotations*

- Recompute the statistics of an ROI via **Action message**

RoiList.Current.RecomputeStatistics = "";

Example Messages

Reference specific objects

- Special name **Current, First, Last** messages highlighted object in GUI's list

  TrialList.Current.BeamList.Last.Couch = 180.0;

- Reference by name (delimiter necessary if name has a space)

  TrialList.Current.BeamList.Lateral.Couch = 180.0;

  OR **TrialList.Current.BeamList.# "Lateral".Couch = 180.0;**

  TrialList.Current.BeamList.# "Ant Post".Couch = 180.0;

- Reference the 3rd element in the list by Index

  TrialList.Current.BeamList.# "#2".Couch = 180.0;

- Special character * loops over all objects in list

  TrialList.Current.BeamList.# "*".Couch = 180.0;

  TrialList.# "*".BeamList.# "*".Couch = 180.0;

Creating Children

Expand the Objectlist

- A **child** is an object in a list (e.g. **Prescription** in a **PrescriptionList**)
- The **CreateChild Action** message appends a new instance of the child class to the list

  TrialList.Current.PrescriptionList.CreateChild = "";
  TrialList.Current.PrescriptionList.CreateChildMakeCurrent = "";
- Alternatively, send the class name to the **ObjectList**

  TrialList.Current.PrescriptionList.Prescription= "";
- Exception: the **CreateChild Actions** for adding beams and regions of interest

  CreateNewBeam = ""; CreateNewROI = "";

Remove Children

Contract ObjectList

- Destroy the current child and remove it from the list
  TrialList.Current.BeamList.Current.Destroy = "";
- Remove the current child from the list (don't free memory)
  TrialList.Current.BeamList.Current.Remove = "";
- Make the last child current
  TrialList.Current.BeamList.Last.MakeCurrent = "";
- Print to terminal the number of elements in the list
  Echo = TrialList.Current.BeamList.Count;
- Add 3 children (Note: this won't work for beams or ROI's)
  PoiList.AddChildren = 3;

Sorting Children

Order the objects intelligently

- An **ObjectList** can be sorted by any child object key

- Sort the beam list by couch angle in ascending order

  TrialList.Current.BeamList.SortBy.Couch = "";

- Multiple keys can be specified

  TrialList.Current.BeamList.SortBy.Couch.Gantry = "";

- To do a descending sort, precede the attribute with "D'‘

  TrialList.Current.BeamList.SortBy.Couch.D.Gantry = "";

- The primary sort key is the couch angle in ascending order
- The secondary sort key is the gantry angle in descending order
- All capitals in names are higher in sort order than lowercase

Root Level Commands

Examples

- Pinnacle^3 contains various root level commands
- These commands tell Pinnacle^3 to perform some **Action** on the entire plan

- Exiting the System

  Quit = “”;

  QuitWithSave = “”;

- Saving Plan

  SavePlan = “”;

- Echo String to the Console (i.e. print statement)

  Echo = “Starting My Script”;

GUI Control Messages

Examples

- Displaying Wait Messages in Status Bar

```
WaitMessage = "Doing something slow...";
...
WaitMessage = “Doing something slower…”;
...
WaitMessageOff = "";
```

- Issuing a Warning Message (Requires user interaction)

```
WarningMessage = "This is a test.";
```

- Issuing an AskYesNo Window

```
AskYesNoPrompt = “Are you having fun?”;
AskYesNoDefault = 1;  // Default yes
IF.AskYesNo.THEN.WarningMessage = “Having fun!”;
```

Dismissing Warnings and Yes/No messages

Examples

- Dismiss the next yes/no message and reply yes

```
Test.ExpectAskYesNo = 1;
Test.ExpectedAskYesNoReply = 1;    // Yes
```

- Dismiss the next yes/no message with the text “Warning text” and reply no

```
Test.ExpectAskYesNo = “Warning text”;
Test.ExpectedAskYesNoReply = 0;    // No
```

- Dismiss the next warning message

```
Test.ExpectWarningMessage= 1;
```

- Dismiss the next warning message with the text “Warning text”

```
Test.ExpectWarningMessage= “Warning text”;
```

Creating Your Own Variables

Using the “Store”

Creating your own variables

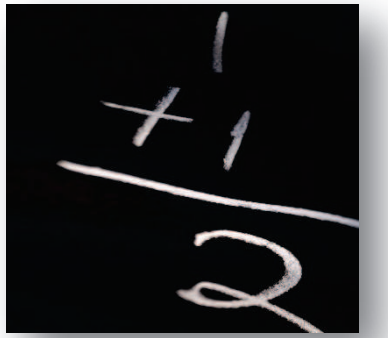
- Pinnacle^3 allows the user to create 3 types of variables that can be used inside your script. These three types are:

    1) **Float** - numerical or boolean operations
    2) **String** - character based information to be stored and manipulated
    3) **Pointer** - references something in Pinnacle^3 memory directly

- These three variables are collectively maintained in the Pinnacle^3 “**Store**”
- The **Store** is not saved with the plan by default
- The **Store** can be a root level Object, or can be created for specific, existing Pinnacle^3 Objects

Store Variables

Float creation/deletion



- Creating Float Entry

  Store.FloatAt.MyFloat = 1.23;

- Modifying Float Entry

  Store.At.MyFloat.Value = 2.34;

  OR **Store.FloatAt.MyFloat = 2.34;**

- Deleting a Float Entry in the store

  Store.FreeAt.MyFloat = “”;

- Printing a float value to the xterm

  Echo = Store.At.MyFloat.Value;

Store Variables

Float numeric operations

- Add, Subtract, Multiply, and Divide syntax
  Store.At.MyFloat.Add = 23.0; //Adds 23 to MyFloat

- Square, SquareRoot, Negate, and Invert(1/x) Syntax
  Store.At.MyFloat.SquareRoot = “”; //Replaces MyFloat with SQRT(MyFloat)

- Round, Nint, and Absolute
  Store.At.MyFloat.Round = “”; //Rounds MyFloat

- Can also be used as a Query
  Store.FloatAt.MyFloat = Store. At.MyOtherFloat.Round;

- Query attributes of Pinnacle^3 objects to set the value of the float
  Store.FloatAt.MyFloat = TrialList.Current.BeamList.Current.Gantry;

Store Variables

String operations

- Creating a String Entry

  Store.StringAt.MyString = “ABC”;

- Modifying a String Entry

  Store.At.MyString.String = “DEF”;

  OR **Store.StringAt.MyString = “DEF”;**

- Concatenating Strings

  Store.At.MyString.AppendString = RoiList.Current.Name;

- Deleting a String Entry

  Store.FreeAt.MyString = “”;

- Getting Environment Variables

  Store.StringAt.HomeDir = GetEnv.HOME;

Store Variables

Pointers

- Creating the Reference by querying the memory address of an Object

```
Store.At.MyBeamReference =
    TrialList.Current.BeamList.Current.Address;
```

- Modifying the Reference

```
Store.At.MyBeamReference.Gantry = 123;
Store.At.MyBeamReference.Couch = 234;
```

- Removing the Reference

```
Store.RemoveAt.MyBeamReference  = "";
```

- **Note: Do not use FreeAt; it destroys the Beam object**

Advanced Techniques

Conditional Operations

Structure

- **IF.key.THEN.action.ELSE.message = value;**
  Where key is a boolean query.

- **IF.key.THEN.message = value;**

- **IF.!key.THEN.message = value;**
  Where **!** is used for to check for false boolean keys

Comparator Operations

Compare values

- **IF.key1.OP.key2.THEN.action.ELSE.message = value;**
- “OP” can be:

Numeric	String	Logical
EQUALTO	Is	AND
NOTEQUALTO	IsNull	OR
GREATERTHAN	Contains	XOR
LESSTHAN	STRINGEQUALTO	
GREATERTHANOREQUALTO	STRINGNOTEQUALTO	
LESSTHANOREQUALTO		

Conditional and Comparator

Examples

- Is:

  IF.Store.At.MyString.Is.# “xyz”.THEN.WarningMessage = “It’s xyz!”;

- Contains:

  IF.Store.At.MyString.Contains .# “xyz”.THEN.WarningMessage = “Has xyz!”;

- **Note: Right side must be a string not a variable**
- Use **STRINGEQUALTO** for two queries instead of **Is**

- IsNull:

  IF.Store.At.MyString.IsNull.THEN.WarningMessage = “It’s empty!”;

- IF - THEN - ELSE numeric:

  IF.TrialList.Count.EQUALTO.# "#1".THEN.
  TrialList.CreateChild.ELSE.TrialList.Last.MakeCurrent=1;

Executing Scripts from Within Scripts

Examples

- Run another script via message

  ExecuteNow = "/home/pinnbeta/MyScript.Script";

- Note the use of two ''='' signs

  Store.StringAt.MyCommand =
  "RoiList.Current.Density = RoiList.Bone.Density";
  Store.At.MyCommand.Execute = "";

- Use "#" to delimit a string within a string

  Store.StringAt.MyCommand ="RoiList.Current.Name = *#Bone";*
  Store.At.MyCommand.Execute = "";

- Use **LoadNoCheckSum** with a string variable if you want to execute another script

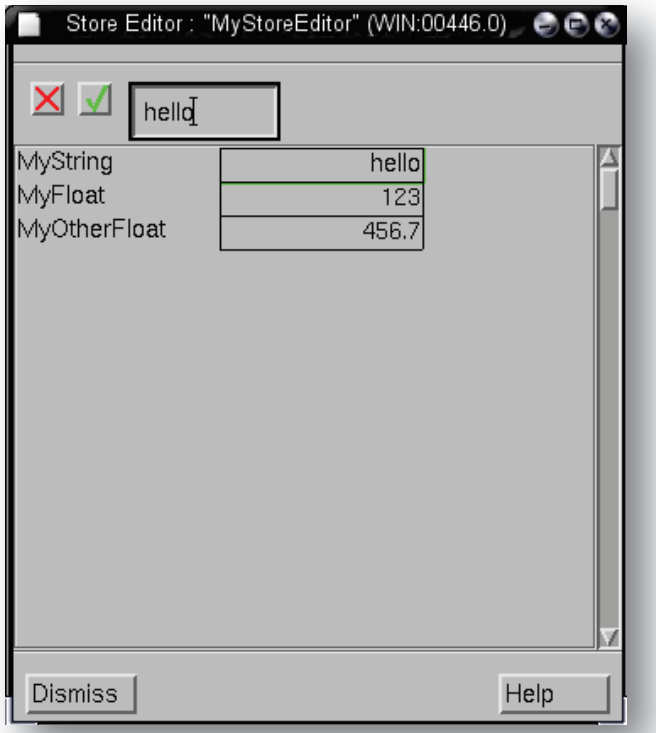
  Store.StringAt.MyCommand =
  "LoadNoCheckSum =/usr/tmp/AddToCount.Script";
  Store.At.MyCommand.Execute = "";

Edit the Store Via the GUI

Ask user for input Via window

- Create a window to display and edit variables
  WindowList.MyStoreEditor.CreateStoreEditor = “Store.Address”;

- Supports Strings and Floats
- Re-launch using the same command (existing window will be created)
- Parameters can be edited by the user.
- Script continues after window launches without waiting for user input



PHILIPS

Force User Entry into the Store Editor

Stop Script from continuing until entry completed

- Create a window to display and edit variables

  WindowList.MyStoreEditor.CreateStoreEditor = “Store.Address”;

- Dismiss the window

  WindowList.MyStoreEditor.Unrealize = “”;

- Make it modal

  WindowList.MyStoreEditor.IsModal = 1;

- Re-launch the window

  WindowList.MyStoreEditor.Create = “”;

  …

  // User dismisses the store window and script continues

Add Attributes to Existing Objects

Append custom variables to any Pinnacle object

- The custom **Store** is specific to each object, not to the class (e.g. two beams might have different variables)
- Functionality is identical to the root **Store**, except that the data is persisted (i.e. saved and copied with object)
- Create an attribute specific to the current beam called MyFloat with value 123.4

```
TrialList.Current.BeamList.Current.
        Store.FloatAt.MyFloatAttribute = 123.4;
```

- You may add as many custom attributes as desired to an existing Pinnacle^3 object and edit them via the **Store Editor**

Adding a Dependency

Trigger something to happen automatically

- Execute MyRoiScript.Script whenever the number of ROIs changes

```
KeyDependencyList.CreateChild = “”;
KeyDependencyList.Last .Name = “NewRoiDependency”;
KeyDependencyList.Last .KeyString = “RoiList.Count”;
KeyDependencyList.Last .AddAction =
        “Script.ExecuteNow = #/home/pinnbeta/MyRoiScript.Script”;
```

Iterating

Method 1: using the * and @ characters

- Special character * loops over all objects in list setting one attribute = 180.0

  TrialList.Current.BeamList.# "*".Couch = 180.0;

- Make each ROI current, then execute another script using **@**

  RoiList.ChildrenEachCurrent.# "@".Script.ExecuteNow =

  "/home/pinnbeta/MyScript.Script";

PHILIPS

Iterating

Method 2: use recursive scripts part 1

- **RecursiveLoopExample.Script** calls iteratively the **AddToCount.Script**

  Store.StringAt.ScriptCommand = "LoadNoCheckSum = /home/pinnbeta/source/Scripts/AddToCount.Script";

- Initialize the counter and limit

  Store.FloatAt.Counter = 0.0;
  Store.FloatAt.Limit = 10.0;

- Start the loop

  Store.At.ScriptCommand.Execute = "";

- Free the store variables

  Store.FreeAt.ScriptCommand = "";
  Store.FreeAt.Counter = "";
  Store.FreeAt.Limit = "";

Iterating

Method 2: Use recursive scripts part 2

- ***AddToCount.Script***

```
Store.At.Counter.Add = 1.0;
Echo = “Adding to counter...”;
Echo = Store.FloatAt.Counter;

IF.Store.FloatAt.Counter.LESSTHAN.Store.FloatAt.Limit.
        THEN.Store.At.ScriptCommand.Execute = “”;
```

Initialization Scripts

Have scripts run every time Pinnacle^3 opens

- There are initialization scripts that can be created to run at Pinnacle^3 and Launchpad startup.
  - **LaunchpadInit** - all commands are run after the database is loaded
  - **PinnacleInit** - all commands in the script are run after the plan is loaded
- These scripts do not exist by default
- Create them and put them in your home directory e.g. /home/pinnbeta

Communicating with the Operating System

Examples

- Write disk usage info to a text file

  SpawnCommand = “df -k > /usr/tmp/MyTempFile.txt”;

- Write text to a text file (overwrites contents) of file

  SpawnCommand = “echo HelloWorld > /usr/tmp/MyTempFile.txt”;

- Write text to a text file (appends contents)

  SpawnCommand = “echo HiAgain >> /usr/tmp/MyTempFile.txt”;

- Launch the gedit text editor and view file as background

  SpawnCommand = “gedit /usr/tmp/MyTempFile.txt &”;

- Execute command in terminal while Pinnacle continues to run

  SpawnCommandNoWait = “xterm –e *OSCommand*”;

Writing Out Specific Data to OS

Write select Pinnacle^3 data to text file via OS

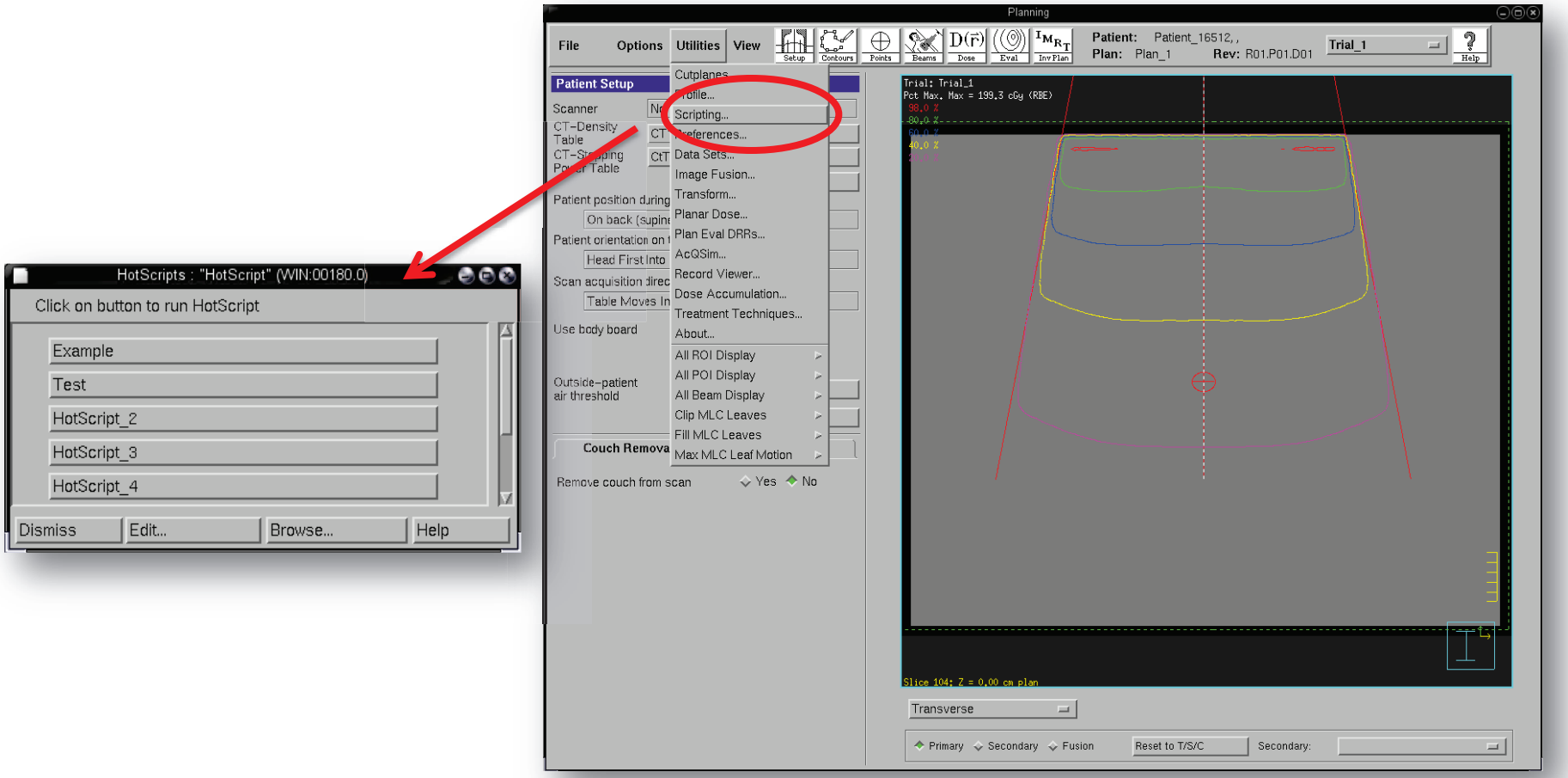
- Find the DVH's first data point and write it out to text file

```
Store.StringAt.EchoCommand = "echo ";
Store.FloatAt.Dose = DVHList.Current.Data.PointsFirst;
Store.At.EchoCommand.AppendString = Store.At.Dose.Value
Store.At.EchoCommand.AppendString = "  >>
        /usr/tmp/OutputFile.txt";
SpawnCommand = Store.At.EchoCommand.String;
```

How to Run and Create Your Own Scripts

How to Run Your Script

Opening the scripting window

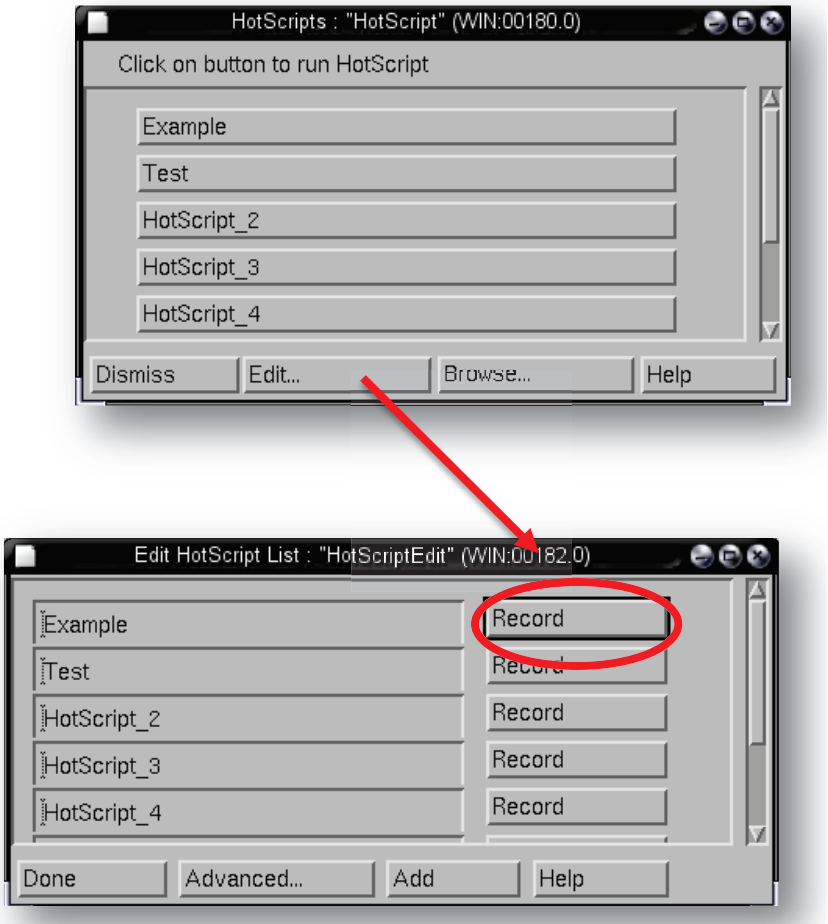


PHILIPS

Finding Messages

Record and playback your GUI interactions

- Recording a script writes out to a text file all the messages that Pinnacle^3 is executing via user interaction
- You may view, edit, rerun the recorded script
- Recording scripts is an easy way to learn the Pinnacle^3 scripting language and find specific messages



PHILIPS

Finding Messages

Read/Write objects to ASCII files on disk

- Writing out an object will place all the persisted data into a text file
  TrialList.Current.BeamList.Current.Save = "/files/rtp/Beam.out";

- Read all the data back in from the text file
  TrialList.Current.BeamList.Current.Load = "/files/rtp/Beam.out";

- Save and Load the custom **Store**
  Store.Save = "/home/pinnbeta/MyStore.Store";
  Store.Load = "/home/pinnbeta/MyStore.Store";

- **Note: The existing Store is not emptied before loading and variables are overwritten if identical names exist**

Finding Messages

Examine Pinnacle^3 files

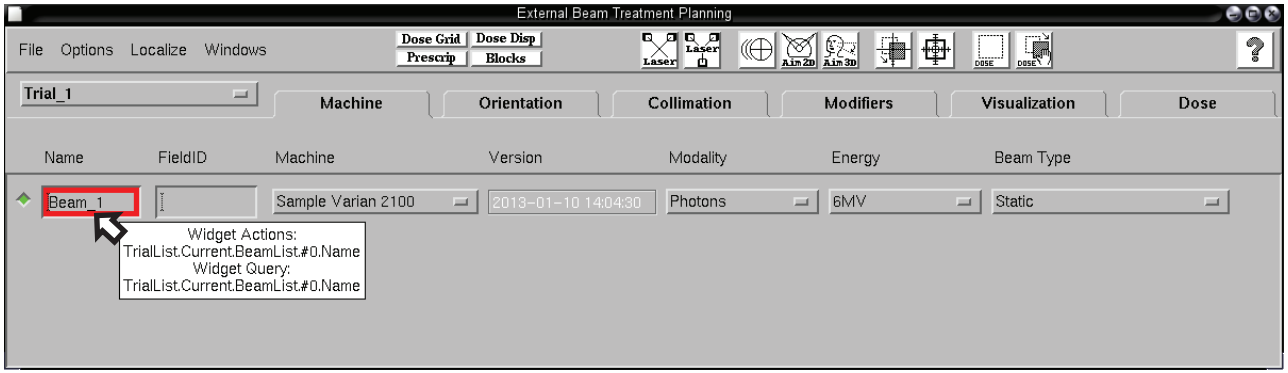
- Nearly all data within a Pinnacle^3 plan are stored to disk
- Pinnacle^3 stores nearly all of its data in Script format, i.e. a series of scripting messages exactly like the ones you would create on your own
- You may examine any of these files to find all the persisted objects, attributes, and functions that can be messaged
- These files include:
  - **Transcript Files - /Institution_X/Mount_Y/Patient_Z/Plan**
  - **Plan Files - Institution_x/Mount_y/Patient_z/Plan_a/plan.Trial**
  - **GUI Configuration files - PinnacleStatic/Config/*.cfg**
- **Note: Do not modify these files!**

Finding Messages

Using the ToolTips

- Pinnacle^3 can display to you the scripting messages behind every widget in the GUI simply by hovering your mouse
- Enable **ToolTips** in Planning mode (available only in research version after 3/2014)

  ShowToolTips = 1;
- To enable this in Launchpad, add the same message to the **LaunchpadInit** file



PHILIPS

Finding Messages

Search reference materials

- As part of this training, your Philips instructor provides additional documentation such as:
  - **Advanced scripting manual**
  - **Example scripts**
  - **Text files with lists of the most common objects and all their messages**

THANK YOU