Pinnacle^3 Scripting

INSTRUCTIONS FOR USE

CONFIDENTIAL

English

Philips Medical Systems TM-SCRIPT-2006 A

PHILIPS

Pinnacle^3 Scripting Training Manual

Equipment specifications are subject to alteration without notice. All changes will be in compliance with regulations governing manufacture of medical equipment.

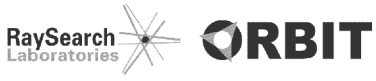
Printed in the United States of America.

Document number
TM-SCRIPT-2006 A

Pinnacle^3, P^{3}IMRT, and P^{3}MD are registered trademarks, and AcQSim^3 and Syntegra are trademarks of Philips Medical Systems.

Sun, UltraSPARC, SPARCstation, and Solaris are trademarks of Sun Microsystems, Inc.

Orbit is a trademark of RaySearch Laboratories AB.



IMFAST is a trademark of Siemens Medical Systems, Inc., Oncology Care Systems.

Other brand or product names are trademarks or registered trademarks of their respective holders.

© Copyright Philips Medical Systems 2006

No part of this publication may be reproduced, transmitted, transcribed, stored in a retrieval system or translated into any human or computer language in any form by any means without the consent of the copyright holder.

Unauthorized copying of this publication may not only infringe copyright but also reduce the ability of Philips Medical Systems to provide accurate and up-to-date information to users and operators alike.

PHILIPS

Philips Medical Systems

Radiation Oncology Systems
5520 Nobel Drive, Suite 125
Fitchburg, WI 53711
USA
Tel: +1 800 722 9377
Fax: +1 608 298 2101
Email: pros.support@philips.com
Web: medical.philips.com
InCenter: incenter.medical.philips.com

North American Headquarters
P.O. Box 3003
22100 Bothell Everett Highway
Bothell, WA 98021-8431
USA
Tel: +1 425 487 7000
Fax: +1 425 487 8130

European Headquarters
Veenpluis 4-6
P.O. Box 10.000
5680 DA Best, The Netherlands
Tel: +31 40 27 62 614
Fax: +31 40 27 64 250

Germany/Eastern Europe Regional Office
Roentgenstrasse 24
Hamburg 22335
Germany
Tel: +49 1805 767 222
Fax: +49 1805 767 229

Latin America Regional Office
Rua Verbo Divino, 1400 - Térreo
Chacara Santo Antonio
04719-002 São Paulo - SP
Brazil
Tel: +55 11 5188 0764
Fax: +55 11 5188 0761

Asia/Pacific Regional Office
Asia Pacific Regional Centre
Level 9, Three Pacific Place
1 Queen's Road East, Wanchai
Hong Kong SAR
Tel: +852 2821 5547
Fax: +852 2527 6727

Philips Medical Systems TM-SCRIPT-2006 A

Philips Medical Systems

Device description

The Pinnacle[3]® Radiation Therapy Planning (RTP) software is composed of several modules including the core Pinnacle[3] functionality, Syntegra™, P[3]IMRT®, P[3]MD®, and AcQSim[3]™. The Pinnacle[3] RTP system is composed of a Sun UNIX workstation (or UNIX-compliant computer) running the Solaris operating system and software, and a UNIX terminal emulation package on a personal computer, which allows the user to enter patient data into the system, use that data to construct a plan for radiation therapy, and evaluate the plan. Optionally, the user may output the plan in an electronic or printed form for use by other systems in the delivery of treatment to a patient.

Pinnacle[3] includes networking capabilities to provide connectivity to other Pinnacle[3], Syntegra, P[3]MD, AcQSim[3], or P[3]IMRT workstations, input devices and output devices, as well as access to the Pinnacle[3] database from any Pinnacle[3] workstation available on the network. Networking connectivity may be via either a local or wide area network.

Intended use

Pinnacle[3] RTP system is a computer software package intended to provide support for radiation therapy treatment planning for the treatment of benign or malignant disease processes.

Pinnacle[3] RTP system assists the clinician in formulating a treatment plan that maximizes the dose to the treatment volume while minimizing the dose to the surrounding normal tissues. The system is capable of operating in both the forward planning and inverse planning modes.

The device is indicated for use in patients deemed to be acceptable candidates for radiation treatment in the judgement of the clinician responsible for patient care.

Plans generated using this system are used in the determination of the course of a patient's radiation treatment. They are to be evaluated, modified, and implemented by qualified medical personnel.

Intended audience

This manual is written for trained users of Pinnacle[3] workstations and for Philips Radiation Oncology Systems (PROS) Field Service Engineers. You should make sure that you have thoroughly read and completely understand the manuals and release notes that are delivered with the software.

Keep this manual and all other manuals delivered with the software near the radiation therapy system and review them periodically.

If you are a customer, the initial installation procedure will be performed by your PROS Field Service Engineer. If you suspect that your system has an error, discontinue its use and contact Customer Support or your local distributor.

Warnings

In the manual we notify you of situations that can potentially affect patient safety or data integrity in Pinnacle³.

WARNING ***This is a warning. It tells you that something that will cause bodily harm could happen or you could lose data if you do not heed this warning.***

CAUTION ***This is a caution. It tells you that something that may cause bodily harm could happen or that you may lose data which is difficult to recover.***

General Device Warnings

WARNING ***Do not load non-system software onto the computer used by this system without the direct authorization of PROS. Feature performance and safety may be compromised.***

WARNING ***To assure proper treatment, it is critical that a qualified individual review and verify all system treatment plan parameters using an independent verification method prior to treating patients using the plan.***

WARNING ***We recommend that you review TG40, TG53, and other pertinent radiation therapy treatment standards and incorporate those methods into your clinical practice to ensure that your use of the system results in the most accurate treatment plans.***

Comprehensive QA for radiation oncology: Report of AAPM Radiation Therapy Committee Task Group 40. **Medical Physics** ***21(4), 1994.***

American Association of Physicists in Medicine Radiation Therapy Committee Task Group 53: Quality assurance for clinical radiotherapy treatment planning. **Medical Physics** ***25(10), 1998.***

WARNING ***U.S.A. law: CAUTION: Federal law restricts this device to sale by or on the order of a physician.***

WARNING ***Only trained personnel should operate the system. New personnel should receive training prior to unsupervised operation of the system. For more information, contact Customer Support or your local distributor.***

Philips Medical Systems

WARNING ***Serious injury to patients can result due to the misapplication of this product. Make sure that you thoroughly understand all the user instructions prior to using this device.***

WARNING ***Pinnacle³ treatment plans may include the statement NOT FOR CLINICAL USE. Based on the machine or isotope data and the treatment plan, the software determined that the plan cannot be delivered clinically. Do not treat patients with plans that are not for clinical use.***

WARNING ***Pinnacle³ includes sample data. This information is for reference purposes only. Do not treat patients with plans based on sample machines or other sample data.***

CE Marking

Medical Device Directive

Pinnacle³ Radiation Treatment Planning System is CE Marked to the Medical Device Directive 93/42/EEC.

CE 0086

Electromagnetic Compatibility (computer hardware only)

This equipment has been tested and found to comply with the EMC limits for the EMC Directive 89/336/EEC. These limits are designed to provide reasonable protection against harmful interference in a typical installation. The equipment generates, uses, and can radiate radio frequency energy and, if not installed and used in accordance with the instructions, may cause harmful interference to other devices in the vicinity. However, there is no guarantee that interference will not occur in a particular installation. If the equipment does cause harmful interference with other devices, which can be determined by turning the equipment off and on, the user is encouraged to try to correct the interference by one or more of the following measures:

- Reorient or relocate the receiving device.
- Increase the separation between the equipment.
- Connect the equipment into an outlet on a circuit not used by the other device(s).
- Consult the manufacturer or field service technician for help.

Philips Medical Systems

Table of Contents

Philips Medical Systems TM-SCRIPT-2006 A

Philips Medical Systems

Philips Medical Systems TM-SCRIPT-2006 A

Philips Medical Systems

1 Introduction

This document describes the scripting capabilities of the Pinnacle^3 system. It explains how you can write scripts to customize the system or extend the system for research purposes.

WARNING ***Great care should be taken when scripting. It is possible with scripts to cause system crashes, to modify dose distributions, or to corrupt data. User-defined scripts are meant for straightforward customizing, or for non-clinical research purposes.***

Philips Medical Systems TM-SCRIPT-2006 A

Philips Medical Systems

2 Objects and Messages

Pinnacle³ is developed using object-oriented techniques. The scripting capability is one benefit of this design.

2.1 Model-view-controller architecture

The Pinnacle³ system is developed using the ANSI C programming language, but provides many of the object-oriented features provided by languages like Smalltalk or Objective-C.

The **model** is the core of the application. It includes the objects and inter- and intra-object dependencies that provide the core functionality of the application. Examples in Pinnacle³ include a beam, a CT data set, or a dose computation engine.

The model is presented to the end user using **views**. The user interface is an obvious view of the model. Reports are another view, as are saved data files.

A **controller** is an element that modifies the model. Text fields and other widgets on the user interface are controllers. A script file is a controller. When a saved data file is read, it also acts as a controller.

The true benefit of this architecture comes from the separation between the model, views, and controllers. Multiple views and controllers interact with the model using messages. The model has no knowledge of the views or controllers. It simply returns information in response to query messages and performs operations in response to command messages.

When a script is played back, it sends messages to objects in the model. These messages are interpreted using the identical mechanisms used to process messages issued from the user interface or from a data file, including the application of any filters or limits.

2.2 Objects

An object is an encapsulation of the data and methods (functions) used to model a real-world object. For example, the Beam object consists of a representation of all beam attributes (a structure) and all methods that operate on the beam attributes (subroutines).

Objects in Pinnacle³ are similar to Smalltalk objects. They are treated as generic objects that respond to certain messages.

Philips Medical Systems TM-SCRIPT-2006 A

Each object is associated with a Class. The Class contains a template for the object and has knowledge of its attributes and messages. It has the ability to create or destroy individual instances of the Class.

When a message is sent to an object, the message list for the object is obtained from the Class. If a matching message is found, the Class knows how to process it, which may involve calling some function, or directly interacting with some attribute of the object. If no matching message is found, a given Class may ignore the message, issue an error message, or pass the message on to a Class from which it inherits behavior.

The *root object* is the highest level object. In Pinnacle[3], the root object is named Pinnacle and contains all elements of the Pinnacle[3] application. All script messages are sent to the root object, which then propagates the messages on to the proper sub-objects.

2.3 Messages

A message is a command that is sent to an object. In its simplest form, a message is a string on the left-hand side and a value on the right-hand side. For example:

```
Name = "test";
```

2.3.1 Basic message syntax

The basic syntax of a command message is:

```
<SetMessage> = <Value>;
```

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

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

NOTE ***A semi-colon must be used to terminate each message.***

The message may identify an object or an attribute of an object. A message is routed through the hierarchy of objects using key pieces delimited by periods. For example, the following message modifies the gantry angle for the current beam in the current trial (sub-plan):

```
TrialList.Current.BeamList.Current.Gantry = 180.0;
```

This message is sent to the TrialList sub-object of the root object. The TrialList is an ObjectList that contains a list of Trial objects. The Current message tells it to route the message to the currently selected Trial in the list. The current Trial accepts the message and routes it to its BeamList

Philips Medical Systems

sub-object, which is an ObjectList containing Beam objects. Eventually the message reaches the current Beam, which recognizes the Gantry message and sets its gantry angle accordingly.

The right hand side of the message can be a Query message. For example, the following example sets the couch angle to be equal to the gantry angle:

```
TrialList.Current.BeamList.Current.Couch =
    TrialList.Current.BeamList.Current.Gantry;
```

Note that messages can be split across lines. All white space and hard returns between messages are ignored. Thus, the following messages are equivalent:

```
TrialList. Current. BeamList. Current. Gantry = 180.0;

TrialList.Current.BeamList.Current.Gantry = 180.0;

TrialList.
    Current.
        BeamList.
            Current.
                Gantry = 180.0;
```

2.3.2 Query messages

If a message has no right hand side, a query is performed with the message and the result is echoed to standard output (the console). The following message echoes the value of the beam gantry angle:

```
TrialList.Current.BeamList.Current.Gantry;
```

For this reason, it is sometimes necessary to include a “dummy” right-hand side argument for a message that has no arguments. Typically an empty string is used:

```
Quit = "";
```

The Quit command has no arguments, but the empty string is necessary so the system knows that a command is being sent and not a query.

2.3.3 Nesting messages

Curly brackets can be used to nest messages. For example, a series of messages can be sent to an object by using curly brackets at a given level.

For example, the following message sets the couch, gantry, and collimator angles for the current beam:

```
TrialList.Current.BeamList.Current = {
    Couch = 180.0;
    Gantry = 90.0;
    Collimator = 0.0;
    };
```

Philips Medical Systems TM-SCRIPT-2006 A

Multiple levels of nesting are possible:

```
TrialList.Current = {
    DoseGrid = {
        VoxelSize.X = 0.2;
        VoxelSize.Y = 0.2;
        VoxelSize.Z = 0.2;
        };
    BeamList.Current = {
        Couch = 180.0;
        Gantry = 90.0;
        Collimator = 0.0;
        };
    };
```

The previous script is equivalent to the following script:

```
TrialList.Current.DoseGrid.VoxelSize.X = 0.2;
TrialList.Current.DoseGrid.VoxelSize.Y = 0.2;
TrialList.Current.DoseGrid.VoxelSize.Z = 0.2;
TrialList.Current.BeamList.Current.Couch = 180.0;
TrialList.Current.BeamList.Current.Gantry = 90.0;
TrialList.Current.BeamList.Current.Collimator = 0.0;
```

2.3.4 Special characters

Certain characters must be escaped when present in scripts. Two common characters which must be escaped are the asterisk (*) and pound sign (#).

The pound sign transforms its string argument into a string literal without interpretation of special characters. The two statements in the following example are identical:

```
TrialList.# "Current".Name = "test"; // Escaped
TrialList.Current.Name = "test"; // Not escaped
```

In the following example, the first line will generate a syntax error because of the special character (asterisk.) The second line will be processed correctly.

```
TrialList.*.Name = "test"; // Syntax error
TrialList.# "*".Name = "test"; // Escaped - no syntax error
```

Unfortunately, the pound sign itself is used in some messages, which leads to some fairly confusing syntax, as in the following example:

```
TrialList.# "#0".Name = "test";
```

The #0 message selects the first (index 0) trial object.

Philips Medical Systems

3 Container Objects

Pinnacle[3] uses lists extensively to store collections of objects. For example, Pinnacle[3] has lists of beams, windows, regions of interest, points of interest, etc.

The ObjectList object is a generic container object that can store an array of *child* objects. Most ObjectList instances have an installed child class, and can create instances of this class. In other words, the list knows what type of object is going to be stored in the list.

3.1 Creating child objects

The "CreateChild" message creates a new instance of the child class and adds it to the list. Objects are always added to the end of the list.

The following message creates a new beam:

```
TrialList.Current.BeamList.CreateChild = "";
```

In addition, sending the class name to the ObjectList performs an identical function.

```
TrialList.Current.BeamList.Beam = "";
```

This is useful because the "Beam" message will also route messages to the object that was last added to the list. Combined with nesting, this is useful for adding a new object and setting some of its parameters, as in the following example:

```
TrialList.Current.BeamList.Beam = {
    Couch = 180.0;
    Gantry = 90.0;
    Collimator = 0.0;
    };
```

The "Beam" message creates a new beam. The "Couch" message is routed through Beam, which sends it to the last added object.

Philips Medical Systems TM-SCRIPT-2006 A

3.2 Referencing child objects

Various messages can be used to route a message to a child object.

3.2.1 Identifying a child object by name

Most objects respond to the message "Name". It is possible to route a message to a child object by specifying the child object's name, as in the following example:

```
TrialList.Current.BeamList.Lateral.Couch = 180.0;
TrialList.Current.BeamList.Ant Post.Couch = 180.0;
```

This message sets the couch angles of the beam objects named "Lateral" and "Ant Post". Note that white space inside a message is significant.

There can be ambiguity when addressing objects by name. For example, a child object that was named "Current" would cause problems since this name conflicts with the ObjectList message named "Current". A message is first compared to known ObjectList messages before being compared to child object names. Most ObjectLists prevent the name of a child object from being set to an ObjectList message name.

3.2.2 Identifying a child object by number

Child objects can also be specified by index. Indexing in an ObjectList starts at zero.

The following message sets the couch angle of the third beam (at index 2).

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

NOTE ***The "#" sign is a special character that must be escaped. See* Special characters *in the* Objects and Messages *chapter.***

The pound sign can be omitted, as in the following example:

```
TrialList.Current.BeamList.2.Couch = 180.0;
```

But this is risky since a beam may have a numeric name. For example, if a beam existed that was named "2", the message above would be routed to this beam even if it was not at index 2.

Philips Medical Systems

3.2.3 Referencing the current child object

The ObjectList maintains a *current* index that points to one of the child objects. Messages are often routed to the current object, so a given script will operate on the beam or other object that is currently selected by the user.

For example:

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

3.2.4 Referencing the last child object

The "Last" message references the last object in the list.

For example:

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

Since newly-created objects are added to the end of the list, the "Last" message can be used to reference an object that has just been created.

3.2.5 Passing a message to all child objects

The wildcard message (*) can be used to send a message to all members of the list. For example, the following message sets the couch angle for all beams to 180.0.

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

Multiple wildcards can be used. The following message sets the couch angle for all beams in all trials.

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

NOTE ***The asterisk message must be escaped. See* Special characters *in the* Objects and Messages *chapter.***

Philips Medical Systems TM-SCRIPT-2006 A

3.3 Destroying child objects

To destroy a child object, pass the message "Destroy" to the child after referencing it using one of the techniques above.

For example, the following commands will destroy the current beam.

```
TrialList.Current.BeamList.Current.Destroy = "";
```

Note that it is not always safe to destroy an object because other objects may reference it. For example, a point of interest object may be referenced by a beam that is using it as an isocenter. Some objects have a sub-object called a "RelyOnList" that can be used to test whether it is safe to destroy the object.

For example, the following could be used to delete a point of interest:

```
IF.PoiList.Current.RelyOnList.HasNoElements.
THEN.PoiList.Current.Destroy.
ELSE.WarningMessage = "Unable to delete POI";
```

3.4 Making a child object the current object

The "MakeCurrent" message is used to make a child object current.

For example, the following message will make the last object in the list the current object:

```
TrialList.Current.BeamList.Last.MakeCurrent = "";
```

3.5 Sorting the ObjectList

An ObjectList can be sorted by any child object key. The following example sorts 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" as in the following example where the primary sort key is the couch angle in ascending order, and the secondary sort key is the gantry angle in descending order.

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

Philips Medical Systems

4 Control Commands

4.1 Exiting the system

The “Quit” and “QuitWithSave” messages exit the core planning software. The “QuitWithSave” message saves all plan data before quitting.

```
Quit = "";
```

or

```
QuitWithSave = "";
```

4.2 Saving the plan

To save the current plan, use the “SavePlan” message.

```
SavePlan = "";
```

Note that there is no argument for this command. The plan files are saved in the plan directory.

4.3 Writing a message to the console

The “Echo” message will send a string to the console window.

```
Echo = "Starting My Script";
```

4.4 Turning on the hourglass cursor

If an operation is going to be lengthy, an hourglass cursor can be installed and a status message can be displayed.

```
WaitMessage = "Doing something slow...";
...
WaitMessageOff = "";
```

Multiple “WaitMessage” messages can be used to show the status of a multi-part process.

```
WaitMessage = "Step 1...";
...
WaitMessage = "Step 2...";
...
WaitMessage = "Step 3...";
...
WaitMessageOff = "";
```

Philips Medical Systems TM-SCRIPT-2006 A

4.5 Issuing a warning message

To issue a warning message that appears in a modal dialog window, use the "WarningMessage" message.

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

The script will continue to execute after the user dismisses the warning message dialog.

4.6 Executing a system command

The "SpawnCommand" and "SpawnCommandNoWait" messages can be used to spawn a new process and execute the specified command. The "SpawnCommand" message will not return control to Pinnacle[3] until the spawned process completes. The "SpawnCommandNoWait" message starts the new process and returns control immediately to Pinnacle[3].

```
WaitMessage = "Checking disk space...";
SpawnCommand = "df > /usr/tmp/MyDiskSpace";
SpawnCommand = "xterm -e vi /usr/tmp/MyDiskSpace";
SpawnCommand = "rm -f /usr/tmp/MyDiskSpace";
WaitMessageOff = "";
```

or

```
SpawnCommandNoWait = "xclock";
```

4.7 Launching other programs

Command	Result
`Store.At.MyString.Execute`	Executes a command
`Script.ExecuteNow`	Executes a script
`SpawnCommand`	Executes an external program
`SpawnCommandNoWait`	Executes an external program and continues processing

Philips Medical Systems

4.8 Running a script from a script

The ExecuteNow command executes the specified script, then returns control to the "calling" script.

```
TrialList.Current...   //Do something
...
Script.ExecuteNow =
"/home/pinnbeta/Scripts/MyOtherScript.Script";
...
TrialList.Current... //Do more stuff.
```

4.8.1 Asking the user a Yes/No question

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

4.8.2 Automatically answering an expected Yes/No question

```
Test.ExpectAskYesNo=1;
Test.ExpectedAskYesNoReply=1; // yes=1, no=0
```

4.8.3 Automatically dismissing a warning

```
Test.ExpectWarning = "Text";
```

NOTE ***Text can be used to dismiss a certain expected warning message. Set text to "1" to dismiss the next warning message.***

Philips Medical Systems TM-SCRIPT-2006 A

4.9 If...then...else

4.9.1 Example 1

```
IF.key1.OP.key2.THEN.action1.ELSE.action2 = value;
```

Where OP is:

Type	Operator
Numeric	EQUALTO
	NOTEQUALTO
	GREATERTHAN
	LESSTHAN
	GREATERTHANOREQUALTO
	LESSTHANOREQUALTO
String	STRINGEQUALTO
	STRINGNOTEQUALTO
Logic	AND
	OR
	XOR

4.9.2 Example 2

```
IF.key.THEN.action1.ELSE.action2 = value;
```

Where “key” is a boolean query.

4.9.3 Example 3

```
IF.key.THEN.action = value;
```

4.10 Iterating

```
RoiList.ChildrenEachCurrent.#"@".Script.ExecuteNow =
"/home/pinnbeta/Scripts/MyScript.Script";
```

The script makes each beam current, then executes the rest of the command.

Philips Medical Systems

5 Find Valid Object Messages

The Pinnacle³ development team uses various code and runtime browsers to view available messages for objects. At this time these browsers are not accessible to end users. In addition, it is not possible to make our internal documentation public because it contains proprietary information.

Pending the development of additional documentation on specific messages, there are a number of methods for finding the available messages of an object or feature.

5.1 Recording commands

The script record capability lets you record a series of operations. The resulting script file will contain script commands. It is often useful to record a script as a starting point, and then edit this script to customize or generalize it. Consult the *Classic Pinnacle³ Planning Instructions for Use*, *Pinnacle³ Planning Instructions for Use*, *Pinnacle³ Planning Reference Guide*, or *AcQSim³ Instructions for Use* for information on recording scripts.

5.2 Transcript files

Every command issued from the user interface is recorded in a plan transcript file, which is a regular script file. Contact PROS for information on retrieving this file.

The file can be used in a similar fashion to a recorded script.

Note that the transcript files are removed periodically by the system.

Philips Medical Systems TM-SCRIPT-2006 A

5.3 Extracting objects

A given object can be saved to a file. This file is a script and will list persistent attributes of the object.

For example, to view the persistent attributes of the beam object, the following script could be used:

```
TrialList.Current.BeamList.Current.Save =
"/files/rtp/Beam.out";
```

The file will contain the following:

```
IsocenterName = "Poi_1";
PrescriptionName = "Prescription_1";
PrescriptionPointName = "isocenter";
Name = "AP";
MachineNameAndVersion = "PhotonQAMachine2: 96/03/29
15:59:29";
Modality = "Photons";
MachineEnergyName = "10 MV";
...
```

Note that most messages have fairly clear names. All of the messages in this file could be used in a script to set the corresponding parameter.

Philips Medical Systems

6 Advanced scripting and store variables

6.1 Variables - the store

The store is a dictionary that associates objects with user-specified strings.

MyFloatAttribute	ptFloatObject
MyStringAttribute	ptStringObject
MyBeamReference	ptBeamReference

The store is a non-persistent attribute of the Root object, accessed via the "Store" message. Store objects can be created for any object.

6.1.1 Creating a float entry in the store

```
Store.FloatAt.MyFloatAttribute = 1.23;
```

Variable label		MyFloatAttribute
New object reference	Class name	Float
	Class attributes	Value = 1.23

6.1.2 Creating a string entry in the store

```
Store.StringAt.MyStringAttribute = "ABC";
```

Variable label		MyStringAttribute
New object reference	Class name	SimpleString
	Class attributes	String = "ABC"

6.1.3 Creating a beam reference in the store

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

Variable label	MyBeamReference
Existing object reference	TrailList Current BeamList Current Address;

The object at "MyBeamReference" is a reference to the beam in the beam list. If the beam in the list is deleted, the Store will not be aware of it and will reference an invalid object.

Philips Medical Systems TM-SCRIPT-2006 A

6.1.4 Modifying a float entry in the store

```
Store.At.MyFloatAttribute.Value = 2.34;
```

Variable label	MyFloatAttribute
Attribute name	Value

Alternative:

```
Store.FloatAt.MyFloatAttribute = 2.34;
```

6.1.5 Modifying a string entry in the store

```
Store.At.MyStringAttribute.String = "DEF";
```

Variable label	MyStringAttribute
Attribute name	String

Alternative:

```
Store.StringAt.MyStringAttribute = "DEF";
```

6.1.6 Modifying a beam reference in the store

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

Object label	MyBeamReference
Attribute name	Gantry, Couch

Alternative:

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

6.1.7 Deleting an entry in the store

```
Store.FreeAt.MyFloatAttribute  = "";
Store.FreeAt.MyStringAttribute  = "";
```

Object label	MyFloatAttribute, MyStringAttribute
Value not used	"";

Philips Medical Systems

6.1.8 Removing an entry in the store

Use “Store.RemoveAt” to remove a reference.

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

Object label	MyBeamReference
Value not used	"";

Do not use “Store.FreeAt” to remove a reference. In the example below, the beam is destoyed but the beam list is unaware of the deletion and the system will crash.

```
Store.FreeAt.MyBeamReference  = "";
```

6.1.9 Saving and reloading the store

```
Store.Save = "/home/pinnbeta/Scripts/MyStore.Script";
```

The file name is /home/pinnbeta/Scripts/MyStore.Script.

```
Store.Save = "/home/pinnbeta/Scripts/MyStore.Script";
```

NOTE ***The existing store is not emptied before loading.***

6.2 String operations

Strings can be concatenated to create file names, unique beam names, etc. The “SimpleString” object and the “Store” are used to build strings.

6.2.1 Building strings

```
Store.StringAt.MyString = "ABC";
Store.At.MyString.AppendString = "DEF";
```

Now the stored string is “ABCDEF”.

```
Store.At.MyString.AppendString = "xyz";
```

Now the stored string is “ABCDEFxyz”.

Philips Medical Systems TM-SCRIPT-2006 A

6.2.2 Useful string queries

Code	String
Store.StringAt.MyString = GetEnv.HOME;	/home/
Store.StringAt.MyString = UserName;	pinnbeta (or p3rtp)
Store.StringAt.MyString = LexicalTimeStamp;	2005/11/22 12:20:23
Store.StringAt.MyString = ReportComment;	whatever is typed in the report comment field
Store.StringAt.MyString = PatientDirectory;	.../Institution_0/Mount_0/Patient_42
Store.StringAt.MyString = PlanName;	plan
Store.StringAt.MyString = PublicScriptDirectory;	.../PinnacleSiteData/Scripts
Store.StringAt.MyString = SystemScriptDirectory;	.../PinnacleStatic/Scripts
Store.StringAt.MyString = PlanInfo.PatientName;	Smith, John
Store.StringAt.MyString = PlanInfo.PlanName;	HeadAndNeck
Store.StringAt.MyString = PlanInfo.Institution;	ClinicAB
Store.StringAt.MyString = PlanInfo.Planner;	Bunker, Gloria
Store.StringAt.MyString = PlanInfo.MedicalRecordNumber;	345126
Store.StringAt.MyString = PlanInfo.Physician;	Doctor, John
Store.StringAt.MyString = PlanInfo.Physicist;	Physicist, Jane

6.2.3 Determining query keys

```
PlanInfo.Save = "/home/Temp.PlanInfo";
```

yields...

```
Institution = "University of XYZ";
PatientName = "Johnson, John";
MedicalRecordNumber = "123-45-6789";
PlanName = "4-field boost";
Planner = "Smith";
Physician = "DocName";
Physicist = "PhysName";
```

6.2.4 File name example

```
// Just in case these didn't get freed for some reason.
Store.FreeAt.TempBaseFileName = "";
Store.FreeAt.TempInputFileName = "";
Store.FreeAt.TempOutputFileName = "";
```

NOTE ***The "FreeAt" command does nothing if the string does not exist.***

Philips Medical Systems

```
// Create a base name for all files and scripts.
Store.At.TempBaseFileName = SimpleString{};
Store.At.TempBaseFileName.AppendString =
PatientDirectory;
Store.At.TempBaseFileName.AppendString =
"/.Temp.MachineUniquenessList";

// Create input file name
Store.At.TempInputFileName = SimpleString{};
Store.At.TempInputFileName.String =
Store.At.TempBaseFileName.String;
Store.At.TempInputFileName.AppendString = ".in";

// Create output file name
Store.At.TempOutputFileName = SimpleString{};
Store.At.TempOutputFileName.String =
Store.At.TempBaseFileName.String;
Store.At.TempOutputFileName.AppendString = ".out";
```

6.3 String conditional statements

Is

```
IF.Store.At.MyString.Is.xyz.
THEN.WarningMessage = "It's xyz!";
```

IsNull

```
IF.Store.At.MyString.IsNull.
THEN.WarningMessage = "It's empty!";
```

Contains

```
IF.Store.At.MyString.Contains.xyz.
THEN.WarningMessage = "Contains xyz!";
```

Philips Medical Systems TM-SCRIPT-2006 A

6.4 The execute statement

```
Store.StringAt.MyCommand = "RoiList.Current.Density =
1.23";
Store.At.MyCommand.Execute = "";

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

NOTE ***#Bone is a string literal deliminator.***

6.5 Numeric operations

Code	Value
`Store.FloatAt.MyFloat = 1.23;`	1.23
`Store.At.MyFloat.Add = 23.0;`	24.23
`.Subtract = 1.0;`	23.23
`.Multiply = 2.0;`	46.46
`.Divide = 2.0;`	23.23

Code	Value
`Store.FloatAt.MyFloat = 9.0;`	9.0
`Store.At.MyFloat.SquareRoot = "";`	3.0
`.Square = "";`	9.0
`.Negate = "";`	-9.0
`.Invert = "";`	1/9.0 = 0.111111

Code	Value
`Store.FloatAt.MyFloat = 1.78;`	1.78
`Store.At.MyFloat.Round = "";`	1.0

Philips Medical Systems

Code	Value
`Store.FloatAt.MyFloat = 1.78;`	1.78
`Store.At.MyFloat.Nint = "";`	2.0

Code	Value
`Store.FloatAt.MyFloat = -1.78;`	-1.78
`Store.At.MyFloat.Absolute = "";`	1.78

NOTE ***These operators support queries:***

```
Store.FloatAt.MyFloat = Store.FloatAt.MyOtherFloat.Round;
Store.FloatAt.MyFloat = Store.FloatAt.MyOtherFloat.Nint;
Store.FloatAt.MyFloat =
Store.FloatAt.MyOtherFloat.Absolute;
```

6.5.1 Numeric operation example

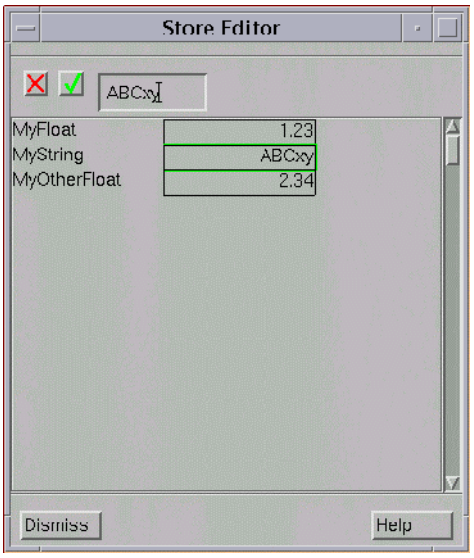
```
Store.FloatAt.MyFloat =
TrialList.Current.BeamList.Current.Gantry;
Store.At.MyFloat.Add = 10.0;
TrialList.Current.BeamList.Current.Gantry =
Store.FloatAt.MyFloat;
Store.FreeAt.MyFloat = "";
```

Philips Medical Systems TM-SCRIPT-2006 A

6.6 Editing the store

- Supports Strings and Floats
- Relaunch using the same command (existing window will be launched)
- Parameters can be edited by the user.

```
WindowList.MyStoreEditor.CreateStoreEditor =
"Store.Address";
```



6.7 Forcing user entry

```
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 = "";
// Continue with script
...
```

6.8 Nesting stores

- StringKeyDict is the class name for the Store object.
- Nesting stores allows editing of a limited set of parameters.

```
Store.At.MyPrivateStore = StringKeyDict{};
Store.At.MyPrivateStore.FloatAt.MyFloat = 1.23;
```

Philips Medical Systems

6.9 Custom data

- A Store object can be created for any object in the hierarchy (e.g. ROI, beam, wedge)
- All functionality is identical to the root Store, except that
  - the data is persistent, and
  - the data is copied with the object.
- The Store is specific to each instance of the class, not the object class (e.g., two beams might have different sets of parameters).

6.9.1 Example 1

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

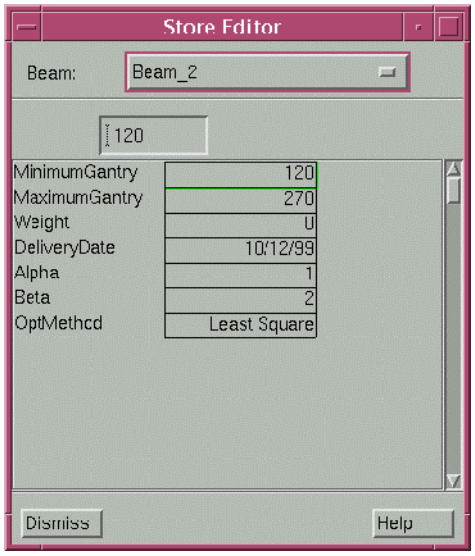
This command

- creates a Store specific to the current beam, and
- adds a float object to the Store named MyFloatAttribute with a value of 123.4.

Philips Medical Systems TM-SCRIPT-2006 A

6.9.2 Example 2

```
TrialList.Current.BeamList.Current.Store = {
    FloatAt.MinimumGantry = 120.0;
    FloatAt.MaximumGantry = 270.0;
    FloatAt.Weight = 0.0;
    StringAt.DeliveryDate = "10/12/99";
    FloatAt.Alpha = 1.0;
    FloatAt.Beta = 1.0;
    StringAt.OptMethod = "Least Square";
};
WindowList.BeamStoreEditor.
CreateStoreEditor = "TrialList.Current.BeamList.
       Current.Store.Address";
```



6.10 Dependencies

- When X occurs, do Y
- X is some event in Pinnacle³ (e.g., the current beam's gantry angle changing, or the current beam changing)
- Y is an action (e.g., executing a script)

Philips Medical Systems

Example

```
KeyDependencyList.CreateChild = "";
KeyDependencyList.Last = {
   Name = "IsocenterDependency";
   KeyString =
"TrialList.Current.BeamList.Current.Isocenter.ZCoord";
   AddAction = "ViewWindowList.#0.SlicePosition =

TrialList.Current.BeamList.Current.Isocenter.ZCoord";
    };
```

- Pans the first viewing window to the isocenter slice.
- Pans the image when:
  - The isocenter Z coordinate changes ZCoord
  - The beam's isocenter changes Isocenter
  - The current beam changes BeamList.Current
  - The current trial changes TrialList.Current

6.10.1 Removing a dependency

```
KeyDependencyList.CreateChild = "";
KeyDependencyList.Last = {
   Name = "IsocenterDependency";
   KeyString =
"TrialList.Current.BeamList.Current.Isocenter.ZCoord";
   AddAction = "ViewWindowList.#0.SlicePosition =

TrialList.Current.BeamList.Current.Isocenter.ZCoord";
    };
KeyDependencyList.IsocenterDependency.Destroy = "";
```

6.10.2 Executing a script from a dependency

- Executes MyRoiScript.Script whenever the number of ROIs changes.
- ExecuteNow executes the specified script immediately.

```
KeyDependencyList.CreateChild = "";
KeyDependencyList.Last = {
   Name = "NewRoiDependency";
   KeyString = "RoiList.Count";
   AddAction = "Script.ExecuteNow =
#/home/pinnbeta/Scripts/MyRoiScript.Script";
   };
```

Philips Medical Systems TM-SCRIPT-2006 A

6.11 Avoiding 'vi'

The following sample scripts allow you to view the results of the `df -k` command in your default editor.

6.11.1 Example 1

```
WaitMessage = "Checking disk space...";
SpawnCommand = "df -k > /usr/tmp/MyDiskSpace";
Store.StringAt.MyEditCommand = "xterm -e ";
Store.At.MyEditCommand.AppendString = GetEnv.EDITOR;
```

NOTE ***Use the EDITOR environment variable.***

```
Store.At.MyEditCommand.AppendString = "
/usr/tmp/MyDiskSpace";
SpawnCommand = Store.StringAt.MyEditCommand;
Store.FreeAt.MyEditCommand = "";
SpawnCommand = "rm -f /usr/tmp/MyDiskSpace";
WaitMessageOff = "";
```

6.11.2 Example 2

```
WaitMessage = "Checking disk space...";
SpawnCommand = "df -k > /usr/tmp/MyDiskSpace";
Store.StringAt.Editor = GetEnv.EDITOR;
IF.Store.StringAt.Editor.IsNull.THEN.Store.StringAt.
Editor = "vi";
```

NOTE ***The line above does error-checking.***

```
Store.StringAt.MyEditCommand = "xterm -e ";
Store.At.MyEditCommand.AppendString =
Store.StringAt.Editor;
Store.At.MyEditCommand.AppendString =
" /usr/tmp/MyDiskSpace";
WarningMessage = Store.StringAt.MyEditCommand;
```

NOTE ***The line above is a debugging message and displays the edit command.***

```
SpawnCommand = Store.StringAt.MyEditCommand;
Store.FreeAt.MyEditCommand = "";
Store.FreeAt.Editor = "";
SpawnCommand = "rm -f /usr/tmp/MyDiskSpace";
WaitMessageOff = "";
```

Philips Medical Systems

7 Example Scripts

7.1 Loading a dose distribution

WARNING ***It is possible to compute a dose distribution outside of Pinnacle³ and load this distribution into a beam. Obviously, great care must be taken with this option and it should be used in RESEARCH MODE ONLY.***

The dose is loaded from a binary file. Dose is assumed to be either a relative quantity or in units of cGy/MU. Each element of the dose array is a 4-byte floating point number. The data is stored like a C array with the X dimension changing fastest.

The message used to insert a dose distribution is "DoseVolume", which is sent to a beam object, as in the following example:

```
TrialList.Current.BeamList.Current.DoseVolume =
\BOB{L}:/files/rtp/MyDose.img\;
```

The DoseVolume attribute points to a binary file. The format of the right-hand side of the DoseVolume line is:

```
\BinaryType:FileName\
```

There are various binary file types that can be used:

- External Data Representation. Best when you have various platforms with different byte order. (This is what Pinnacle³ uses to save the dose distributions.)
- {L} Binary data with little-endian byte order.
- {B} Binary data with big-endian byte order.
- Hexadecimal data with big-endian byte order.

The FileName is a standard UNIX file name. Alternatively, an index can be specified, in which case the system looks for the name of the script with ".binary.<Index>" appended.

For example, if the script is named /files/rtp/InsertDose.Script and it contains the following:

```
TrialList.Current.BeamList.#"#0".DoseVolume = \BOB{L}:0\;
TrialList.Current.BeamList.#"#1".DoseVolume = \BOB{L}:1\;
```

It will look for the binary data in two files:

- /files/rtp/InsertDose.Script.binary.000, and
- /files/rtp/InsertDose.Script.binary.001.

Philips Medical Systems TM-SCRIPT-2006 A

The system will assume that the array dimensions match the dose grid dimensions. Thus it is often necessary to specify the dose grid geometry before inserting the dose distribution. The dose grid geometry is set using the "Dimension", "VoxelSize", and "Origin" messages, which are sent to the DoseGrid sub-object of the Trial object.

```
TrialList.Current.DoseGrid =
    Dimension.X = 44;
    Dimension.Y = 73;
    Dimension.Z = 35;
    VoxelSize.X = 0.4;
    VoxelSize.Y = 0.4;
    VoxelSize.Z = 0.4;
    Origin.X = -8.022;
    Origin.Y = -29.866;
    Origin.Z = -6.216;
    ;
TrialList.Current.BeamList.Current.DoseVolume =
BOBL:/files/rtp/MyDose.img
TrialList.Current.BeamList.Current.MonitorUnitsValid = 1;
```

For an array with units of cGy/MU, the "MonitorUnitsValid" attribute of the beam should be set to 1. If the dose is relative, set "MonitorUnitsValid" to 0.

7.2 Extracting a block outline for a beam

The blocks for a beam are stored by definition. The actual final block outline is not stored. For example, the beam data file will store the rule that the block should expose some structure with a given margin. It is sometimes useful to extract the actual block contour—for export to a block cutter, for example.

The internal block contour that results from all automatic and manual block definitions can be extracted by messaging the BlockContourList sub-object of the Beam object. The BlockContourList stores an array of contours that define the edge of the blocks. Note that all edges will be included, so there may be some ambiguity with "island" blocks where two contours will be present in the list.

Philips Medical Systems

If the beam blocking is described with a single contour, this contour can be extracted to a file as follows:

```
TrialList.Current.BeamList.Current.BlockContourList.
Current.WriteToFile = "/files/rtp/Blocks.out";
```

The "WriteToFile" message is a special message of the CurveNd object that writes the points in the curve to the file preceded by the number of points. For example:

```
24
22.3 11.2
22.5 10.4
...
```

For a beam with multiple block contours, the entire block contour list can be sent to a file. In this case, the regular object persistence mechanism is used.

```
TrialList.Current.BeamList.Current.BlockContourList.
Save = "/files/rtp/BlockList";
```

This results in a file like the following:

```
CurveNd =
   NumberOfDimensions = 2;
   NumberOfPoints = 64;
   Points[] =
       -0.125,-5.275,
       1.375,-5.275,
       1.925,-5.225,
       ...
   ;
;
 CurveNd =
    NumberOfDimensions = 2;
    NumberOfPoints = 34;
   Points[] =
        -2.875,-2.525,
       -2.725,-2.925,
       -2.525,-3.175,
       ...
   ;
;
```

Philips Medical Systems TM-SCRIPT-2006 A

7.3 Changing an automatic block to a manual block

It is sometimes desirable to generate a block automatically and then edit the resulting block outline. To do this the block must be converted from an automatic block to a manual block. The following script converts an automatic block to a manual block:

```
// This script will copy an automatically generated block
// contour into a manual block contour.
// NOTE: This script will only copy the first
// automatically generated contour.
// It will not copy multiple automatic contours
// Before running this script, create a beam, add a block
// to the beam,and generate an automatic block for the
// target volume. Delete any existing manual contours.
TrialList.Current.BeamList.Current.
   ModifierList.Current.ContourList.DestroyAllChildren =
"";
// Create a new contour
TrialList.Current.BeamList.Current.
   ModifierList.Current.ContourList.CreateChild = "";
// Copy the number of points.
TrialList.Current.BeamList.Current.

ModifierList.Current.ContourList.Current.NumberOfPoints =
TrialList.Current.BeamList.Current.BlockContourList.
Current.NumberOfPoints;
// Copy the points. (In version 1.5, a simple Copy could
// be used. For older versions, must use PointsAsBlob.)
TrialList.Current.BeamList.Current.
   ModifierList.Current.ContourList.Current.PointsAsBlob
= TrialList.Current.BeamList.Current.BlockContourList.
Current.PointsAsBlob;
// Set the margin to zero. Otherwise the margin will be
// applied to the new manual contour and it won't match
// the auto-generated contour.
TrialList.Current.BeamList.Current.
   ModifierList.Current.Margin = 0.0;
// Set the block to be manual instead of automatic.
TrialList.Current.BeamList.Current.
   ModifierList.Current.StructureToBlock = "Manual";
```

Philips Medical Systems

7.4 Forcing a viewing window to track the beam isocenter

Using a dependency, it is possible to specify that a given viewing window should display the slice containing the current beam's isocenter, and pan to the proper slice when the isocenter is moved, or when a new beam becomes current.

The following script will set up such a dependency. It sets the first window (window 0) to always display the slice corresponding to the isocenter for the current beam.

```
KeyDependencyList.CreateChild = "";
KeyDependencyList.Last =
   Name = "IsocenterDependency";
   KeyString =
"TrialList.Current.BeamList.Current.Isocenter.ZCoord";
   AddAction = "ViewWindowList.0.SlicePosition =
TrialList.Current.BeamList.Current.Isocenter.ZCoord";
;
```

The first viewing window will be panned to the proper slice when any of the following occurs:

- Z coordinate of isocenter is modified.
- The beam is aimed at a different isocenter (point of interest).
- A different current beam is selected.
- A different current trial is selected.

The following script removes the dependency:

```
KeyDependencyList.IsocenterDependency.Destroy = "";
```

Philips Medical Systems TM-SCRIPT-2006 A

7.5 Timing a process

This example demonstrates how to create a Timer object to time a given process. It also demonstrates use of the temporary object database to store the temporary Timer object.

```
Store.At.Clock = Timer ; // Create Timer object. Reference
as "Clock".
SavePlan = ""; // Save all plan files.
Echo = "Elapsed time to store plan: ";
Echo = Store.At.Clock.ElapsedTime; // Query elapsed time
from timer.
Store.FreeAt.Clock = ""; // Destroy the timer.
```

The first line of the script creates a new Timer object. The curly brackets {} are the syntax used to indicate that a new instance of the class named "Timer" should be created. Initial values for the object could be placed inside these brackets. The "Store.At.Clock" message tells the system to store the object using the name "Clock". Messages can then be routed to the object through "Store.At.Clock".

7.6 Adding four blocked beams

The following is the "four field prostate" script, which is one of the scripts shipped with Pinnacle[3].

```
TrialList.Current.BeginMessageBatch = "";
/* Add a new beam. */
TrialList.Current.BeamList.CreateChild = "";
/* Set the gantry angle to zero for the last (newly added)
beam. */
TrialList.Current.BeamList.Last.Gantry = 0.0;
/* Add a new block. */
IF.RoiList.ContainsObject.prostate.THEN.

TrialList.Current.BeamList.Last.ModifierList.CreateChild.
ELSE.
   WarningMessage = "No blocking added because no ROI
named 'prostate'";
/* Automatically block the current ROI. */
TrialList.Current.BeamList.Last.ModifierList.Last.Structu
reToBlock
   = RoiList.Current.Name;
/* Set the automatic blocking margin to 1.0 cm. */
TrialList.Current.BeamList.Last.ModifierList.Last.Margin
= 1.0;
```

Philips Medical Systems

```
/* ============= 90 ================================= */
/* Copy last beam (one which was set up above.) This
copies blocking.*/
TrialList.Current.BeamList.Copy =
   TrialList.Current.BeamList.Last.Address;
/* Set gantry to 90 degrees. */
TrialList.Current.BeamList.Last.Gantry = 90.0;
TrialList.Current.BeamList.Last.NextColor = "";
/* ============= 180 ================================
*/
TrialList.Current.BeamList.Copy =
   TrialList.Current.BeamList.Last.Address;
/* Set gantry to 180 degrees. */
TrialList.Current.BeamList.Last.Gantry = 180.0;
TrialList.Current.BeamList.Last.NextColor = "";
/* ============= 270 ================================
*/
TrialList.Current.BeamList.Copy =
   TrialList.Current.BeamList.Last.Address;
/* Set gantry to 270 degrees. */
TrialList.Current.BeamList.Last.Gantry = 270.0;
TrialList.Current.BeamList.Last.NextColor = "";
TrialList.Current.EndMessageBatch = "";
```

This script demonstrates various features:

- The “Copy”' message used to copy the first beam to the three final beams is a message handled by ObjectList. It creates a copy of the object specified on the right hand side of the message and adds it to the list.
- The “IF” conditional message can be used for simple branching.
- The “BeginMessageBatch” and “EndMessageBatch” messages can be sent to any object when a series of messages is going to be sent to the object. The object may or may not behave differently when in *message batch mode.* This prevents lengthy operations from being performed repeatedly during filing or script playback. For example, the Trial object does not sum together dose grids while in message batch mode. When it sees the “EndMessageBatch” message, it sums the beam dose grids.

Philips Medical Systems TM-SCRIPT-2006 A

7.7 Sorting a list of objects

An ObjectList can be sorted by any child object key. The following example sorts 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" as in the following example where the primary sort key is the couch angle in ascending order, and the secondary sort key is the gantry angle in descending order.

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

7.8 Adding an ROI curve

```
// Create a new region-of-interest.
CreateNewROI = "";
```

NOTE ***You must use CreateNewROI, not RoiList.CreateChild.***

```
// Define the points for a new curve.
RoiList.Last.EditCurve = {
   SliceCoordinate = 0.7;   // Slice position (cm)
   Orientation = "Transverse";   // (Optional -
transverse default)
   Curve = {
      NumberOfPoints = 5;
      Points[] = {
         1.0, 7.0,     // These are CT coordinates (cm).
         1.0, -7.0,
         -2.0, -7.0,
         -2.0, 7.0,
         1.0, 7.0
         };
      };
   };
// Add the new curve to the ROI
RoiList.Last.CopyEditCurveToNewCurve = "";
```

NOTE ***You must use the indirect method since ROI curves do not support messaging.***

Philips Medical Systems

7.9 Controlling viewing windows

For a window named XYZ (see the BEV selection menu for names):

```
ViewWindowList.XYZ.Orientation = "Sagittal";    (or
"Transverse" or "Coronal")
ViewWindowList.XYZ.InterpMode = "Nearest Neighbor
Interpolation";
(or "Linear Interpolation")
ViewWindowList.XYZ.Enter2dMode = 1;
    .Enter3dMode = 1;
    .SetNextTrial = 1;
    .NextImage = 1;
    .PreviousImage = 1;
    .SliceNumber = 23;
    .UserSliceNumber = 24;
    .SlicePosition = 13.5;
    .PanToFraction = 0.5; // Pan to slice which is 50%
through data set.
    .ScaleZoom = 2.0; // Doubles the current zoom factor.
    .Zoom = 1.0; // Sets zoom to 1.0 (life size.)
    .ZoomToWindow = 1; // Fit slice to window
```

Philips Medical Systems TM-SCRIPT-2006 A

Philips Medical Systems