Eclipse Scripting API Reference Guide

Eclipse

P1008614-005-E

OCTOBER 2016

Document ID	P1008614-005-E
Document Title	*Eclipse Scripting API Reference Guide*
Abstract	This document provides information about using Eclipse Scripting API, version 13.6 MR2. This publication is the English-language original.
Manufacturer	Varian Medical Systems, Inc. 3100 Hansen Way Palo Alto, CA 94304-1038 United States of America
European Authorized Representative	Varian Medical Systems UK Ltd. Oncology House Gatwick Road, Crawley West Sussex RH10 9RG United Kingdom
Notice	Information in this user guide is subject to change without notice and does not represent a commitment on the part of Varian. Varian is not liable for errors contained in this user guide or for incidental or consequential damages in connection with furnishing or use of this material. This document contains proprietary information protected by copyright. No part of this document may be reproduced, translated, or transmitted without the express written permission of Varian Medical Systems, Inc.
FDA 21 CFR 820 Quality System Regulations (cGMPs)	Varian Medical Systems, Oncology Systems products are designed and manufactured in accordance with the requirements specified within this federal regulation.
ISO 13485	Varian Medical Systems, Oncology Systems products are designed and manufactured in accordance with the requirements specified within the ISO 13485 quality standard.
CE	Varian Medical Systems, Oncology Systems products meet the requirements of Council Directive MDD 93/42/EEC.
IEC62083	Eclipse™ Treatment Planning System is IEC62083:2009 compliant.
EU REACH SVHC Disclosure	The link to the current EU REACH SVHC disclosure statement can be found at http://www.varian.com/us/corporate/legal/reach.html.
HIPAA	Varian's products and services are specifically designed to include features that help our customers comply with the Health Insurance Portability and Accountability Act of 1996 (HIPAA). The software application uses a secure login process, requiring a user name and password, that supports role-based access. Users are assigned to groups, each with certain access rights, which may include the ability to edit and add data or may limit access to data. When a user adds or modifies data within the database, a record is made that includes which data were changed, the user ID, and the date and time the changes were made. This establishes an audit trail that can be examined by authorized system administrators.
WHO	ICD-0 codes and terms used by permission of WHO, from: ■ International Classification of Diseases for Oncology, (ICD-0) 3rd edition, Geneva, World Health Organization, 2000. ICD-10 codes and terms used by permission of WHO, from: ■ International Statistical Classification of Diseases and Related Health Problems, Tenth Revision (ICD-10). Vols 1-3, Geneva, World Health Organization, 1992.

Electronic labeling

This symbol on the label indicates that the Instructions for Use for the corresponding product are available at www.MyVarian.com. Access the Instructions for Use in electronic form by logging in with your assigned MyVarian user credentials.

In compliance with EU Commission Directive No 207 / 2012, Varian will send EU customers a free printed copy of the Instructions for Use within 7 days. Use the "Paper Document Request" form provided on the Varian webpage to order your copy.

CAUTION: US Federal law restricts this device to sale by or on the order of a physician.

Contents

Chapter 1 Introduction

Eclipse is used to plan radiotherapy treatments for patients with malignant or benign diseases. The users of Eclipse are medical professionals who have been trained in radiation dosimetry. After an oncologist has decided that radiotherapy is the suitable treatment for a patient, the medical professionals use Eclipse to plan the treatment for the patient. Eclipse can be used to plan external beam irradiation with photon, electron, and proton beams, as well as for internal irradiation (brachytherapy) treatments. Eclipse is part of Varian's integrated oncology environment.

The Eclipse Scripting Application Programming Interface (Eclipse Scripting API or ESAPI) is a programming interface and a software library for Eclipse. It allows software developers to write scripts to access the treatment planning information in Eclipse. The scripts can be integrated into the Eclipse user interface, or they can be run as stand-alone executables.

Who Should Read This Manual

This manual is written mainly for medical/technical personnel who wish to write custom scripts to be used in Eclipse. It is assumed that you are familiar with:

- Eclipse Treatment Planning System
- Radiation oncology domain and concepts
- DICOM
- Software engineering practices
- Microsoft Visual Studio development environment
- Microsoft Visual C# programming language and object oriented development

Note *Before creating your own scripts, familiarize yourself with the Eclipse user documentation, especially any safety-related information, cautions, and warnings found throughout the documentation.*

Visual Cues

This publication uses the following visual cues to help you find information:

WARNING: A warning describes actions or conditions that can result in serious injury or death.

CAUTION: A caution describes hazardous actions or conditions that can result in minor or moderate injury.

NOTICE **A notice describes actions or conditions that can result in damage to equipment or loss of data.**

Note *A note describes information that may pertain to only some conditions, readers, or sites.*

Tip *A tip describes useful but optional information such as a shortcut, reminder, or suggestion, to help get optimal performance from the equipment or software.*

Related Publications

RT Administration Reference Guide	P1008617-001-A
Beam Configuration Reference Guide	P1008610-001-A
BrachyVision Instructions for Use	P1005650-001
BrachyVision Reference Guide	P1005651-001
Acuros BV Algorithm Reference Guide	B504878R01A
Eclipse Photon and Electron Algorithms Reference Guide	P1008611-001-A
Eclipse Cone Planning Online Help	B504830R01A
Eclipse Ocular Proton Planning Reference Guide	P1005367-001-A
Eclipse Photon and Electron Instructions for Use	P1008620-001-A
Eclipse Photon and Electron Reference Guide	P1008621-001-A
Eclipse Proton Reference Guide	P1008623-001-A
Eclipse Proton Instructions for Use	P1008622-001-A
Eclipse Proton Algorithms Reference Guide	B504886R01A
Eclipse Scripting API Online Help	P1008612-001-A

Contact Varian Customer Support

Varian Customer Support is available on the internet, by e-mail, and by telephone. Support services are available without charge during the initial warranty period.

The MyVarian website provides contact information, product documentation, and other resources for all Varian products.

1. Go to www.MyVarian.com.

2. Choose an option:
   - If you have an account, enter your User login information (email and password).
   - If you do not have an account, click **Create New Account** and follow the instructions. Establishing an account may take up to two working days.
3. Click **Contact Us** at the top of the window to display customer support and training options, and international e-mail addresses and telephone numbers.
4. From the Contact Us page, choose an option:
   - Call Varian Medical Systems support using a phone support number for your geographic area.
   - Complete the form corresponding to your request for use on a call with a live Varian representative; then follow the instructions to complete the remote connection.

   You can order documents by phone, request product or applications support, and report product-related issues. Links on the MyVarian website navigate to other support resources for products, services, and education.
5. To find documents, click **Product Documentation**.

   Online documents in PDF format include customer technical bulletins (CTBs), manuals, and customer release notes (CRNs).

Chapter 2 About the Eclipse Scripting API

The Eclipse Scripting API is a Microsoft .NET class library that gives you read access to the treatment planning data of Eclipse. It allows you to create scripts that leverage the functionality of Eclipse, and lets you retrieve plan, image, dose, structure, and DVH information from the Varian System database. The data is retrieved from the Varian System database also in stand-alone Eclipse installations. You can integrate the scripts into Eclipse, or you can run them as stand-alone executables.

WARNING: **The authors of custom scripts are responsible for verifying the accuracy and correctness of the scripts after developing a new script or after system upgrade for the existing scripts.**

Features

By using the Eclipse Scripting API, you can:

- Write custom scripts and integrate them into the Eclipse user interface.
- Write stand-alone executable applications that leverage the Eclipse Scripting API.

You can access the following information with ESAPI scripts:

- Image and structure models, including their volumetric representations.
- Plans, fields, and accessories.
- Predecessor Plans.
- Plan Protocol Information.
- IMRT optimization objectives and parameters.
- Doses, including their volumetric representations.
- Dose volume histograms.
- Optimal fluences.
- DVH Estimates
- Prescription Information.

The Eclipse Scripting API provides you also the following:

- A wizard that makes it simple to create new scripts.
- Patient data protection that complies with HIPAA.
- Support for user authorization used in Eclipse and ARIA Radiation Therapy Management (RTM).
- API documentation.
- Example applications.
- Full 64-bit support.

System Requirements

The basic system requirements of the Eclipse Scripting API are the same as those of Eclipse. For more information, refer to *Eclipse Customer Release Note*.

To create scripts with the Eclipse Scripting API 13.6, you need:

- Eclipse 13.6 or later.
- A license for the Eclipse Scripting API 13.6.

Note: Microsoft Visual Studio is not needed for creating scripts. However, some features described in this document assume that Microsoft Visual Studio 2010 has been installed.

Version Compatibility

ESAPI 13.6

- The Eclipse Scripting API 13.6 is compatible with Eclipse 13.6.
- Varian Medical Systems provides no guarantee that scripts written with this version of the Eclipse Scripting API will be compatible with future releases.

ESAPI 13.5

- The Eclipse Scripting API 13.5 is compatible with Eclipse 13.5.
- Varian Medical Systems provides no guarantee that scripts written with this version of the Eclipse Scripting API will be compatible with future releases.

ESAPI 13.0

- The Eclipse Scripting API 13.0 is compatible with Eclipse 13.0.
- Varian Medical Systems provides no guarantee that scripts written with this version of the Eclipse Scripting API will be compatible with future releases.

ESAPI 11.0

- The Eclipse Scripting API 11.0 is compatible with Eclipse 11.0.

Incompatibilities between ESAPI 11.0 and ESAPI 13.0

- The type `VMS.TPS.Common.Model.Types.VRect` has been changed to immutable. Scripts that use the set accessors of VRect properties are incompatible with the Eclipse Scripting API 13.0.
- The type `VMS.TPS.Common.Model.ExternalBeam` has been marked as obsolete. It is replaced by the `VMS.TPS.Common.Model.ExternalBeamTreatmentUnit` type.
- The property `VMS.TPS.Common.Model.Beam.ExternalBeam` has been marked as obsolete. It is replaced by the `VMS.TPS.Common.Model.Beam.TreatmentUnit` property.

Note *If you use an obsoleted type, property, field, or method, the compiler shows a warning. In this case, the compilation of a single-file plug-in fails. If the script is a binary plug-in or a standalone executable, the compiler shows an error. This happens only if the "Treat warnings as errors" project setting is turned on in Microsoft Visual Studio.*

Upgrade to ESAPI 13.6

Stand-alone scripts that have been compiled using older versions of Eclipse Scripting API do not work after upgrading to the Eclipse Scripting API 13.6. Additionally, binary plug-ins do not compile after the upgrade.

To make the scripts work with ESAPI 13.6, you need to update the Visual Studio projects to reference the new ESAPI 13.6 assemblies.

Do the following:

1. Open the Eclipse Script Visual Studio project.
2. Expand the References item in the Solution Explorer. You should see the existing references to `VMS.TPS.Common.Model.API` and `VMS.TPS.Common.Model.Types`.
3. Remove both references from the project.
4. Add new references to the ESAPI 13.6 assemblies.
5. In the **Add Reference** dialog box, select the **Browse** tab. You should find the ESAPI 13.6 assemblies from *C:\Program Files (x86)\Varian\Vision\13.6\Bin64.*
6. Add references to both `VMS.TPS.Common.Model.API.dll` and `VMS.TPS.Common.Model.Types.dll`.
7. Recompile the project.

What Is New in Eclipse Scripting API 13.6

You can now add favorite scripts to the Eclipse menu and define keyboard shortcuts for them. See *Launching Scripts*.

The following new properties, functions, and classes have been added or changed in ESAPI 13.6. See the detailed documentation in *Eclipse Scripting API Online Help*.

- `VMS.TPS.Common.Model.API.Course`: added properties `Diagnoses`, `Intent`, and `Patient`.
- `VMS.TPS.Common.Model.API.PlanningItem`: changed the type of the property `Dose` from `VMS.TPS.Common.Model.API.Dose` to `VMS.TPS.Common.Model.API.PlanningItemDose`. Added convenience methods `DoseAtVolume` and `VolumeAtDose`.
- `VMS.TPS.Common.Model.API.PlanSetup`: added properties `Series`, `SeriesUID, PlanIntent`, `VerifiedPlan, PredecessorPlan, RTPrescription, PlanObjectiveStructures and ApprovalHistory`. Added new method `GetProtocolPrescriptionsAndMeasures.`

- VMS.TPS.Common.Model.API.PlanSum: added properties Course and PlanSumComponents. Added functions GetPlanSumOperation and GetPlanWeight.
- VMS.TPS.Common.Model.API.Beam: added properties BeamNumber, Dose, ToleranceTableLabel, and TreatmentTime.
- VMS.TPS.Common.Model.API.ControlPoint: added properties TableTopLateralPosition, TableTopLongitudinalPosition, and TableTopVerticalPosition.
- VMS.TPS.Common.Model.API.Dose: added properties Series and SeriesUID,
- VMS.TPS.Common.Model.API.Structure: added property StructureCodeInfos.
- Class VMS.TPS.Common.Model.API.Dose is now a base class. New derived classes are: VMS.TPS.Common.Model.API.PlanningItemDose and VMS.TPS.Common.Model.API.BeamDose.
- A new class VMS.TPS.Common.Model.API.PlanSumComponent was added to provide information about component plans of plan sums. See the new property VMS.TPS.Common.Model.API.PlanSum.PlanSumComponents.
- A new class VMS.TPS.Common.Model.API.Diagnosis was added to provide information about the diagnoses that have been attached to the course. See the new property VMS.TPS.Common.Model.API.Course.Diagnoses.
- A new struct VMS.TPS.Common.Model.Types.StructureCodeInfo was added to provide information about structure codes that have been assigned to a structure. See the new property VMS.TPS.Common.Model.API.Structure.StructureCodeInfos.
- Class VMS.TPS.Common.Model.API.ScriptContext: added properties ApplicationName and VersionInfo.
- Class VMS.TPS.Common.Model.API.Block: added property Outline, the projected block outline on the isocenter plane.
- VMS.TPS.Common.Model.API.Registration: added properties Status, StatusDateTime, StatusUserName.
- VMS.TPS.Common.Model.API.ReferencePoint: added properties TotalDoseLimit, DailyDoseLimit, and SessionDoseLimit.
- A new class VMS.TPS.Common.Model.API.BrachyFieldReferencePoint provides brachytherapy reference point information. Brachytherapy reference points are accessed through new properties VMS.TPS.Common.Model.API.Catheter.BrachyFieldReferencePoints and VMS.TPS.Common.Model.API. SeedCollection.BrachyFieldReferencePoints.
- A new class VMS.TPS.Common.Model.API.RTPrescription gives access to the prescription associated with a plan through the new property PlanSetup.RTPrescription.

- New classes `VMS.TPS.Common.Model.API.ProtocolPhasePrescription`, `VMS.TPS.Common.Model.API. ProtocolPhaseMeasure` provide information about the plan protocols associated with the plan. . The new method `VMS.TPS.Common.Model.API.PlanSetup. GetProtocolPrescriptionsAndMeasures` provides access to protocol information for a plan.
- New class `VMS.TPS.Common.Model.API.ApprovalHistoryEntry` provides information about the approval history associated with the plan. The new method `VMS.TPS.Common.Model.API.PlanSetup.ApprovalHistory` provides access to the approval history for a plan.

Supported Script Types

Eclipse supports two types of scripts: plug-ins and executable applications.

Plug-ins

Plug-ins are launched from the Eclipse user interface. After the launch, the plug-in gains access to the data of the currently open patient.

Eclipse supports two types of plug-ins:

- A single-file plug-in: A source code file that Eclipse reads, compiles on the fly, and connects to the data model of the running Eclipse instance.
- A binary plug-in: A compiled .NET assembly that Eclipse loads and connects to the data model of the running Eclipse instance.

Eclipse creates a Windows Presentation Foundation child window that the script code can then fill in with its own user interface components. The plug-in scripts receive the current context of the running Eclipse instance as an input parameter. The context contains the patient, plan, and image that are active in Eclipse when the script is launched. The plug-in scripts work only for one patient at a time in Eclipse.

Executable Applications

A stand-alone executable is a .NET application that references the Eclipse Scripting API class library. It can be launched just like any Windows application.

Stand-alone executables can be either command-line applications, or they can leverage any .NET user interface technology available on the Windows platform.

While the plug-in scripts are restricted to work for one single patient opened in Eclipse, the stand-alone executable can scan the database and open any patient.

Chapter 3 Eclipse Scripting API Object Model

The Eclipse data model is presented in the Eclipse Scripting API as a collection of .NET classes with properties and methods. The class hierarchy is an abstraction over the ARIA Radiation Therapy Management (RTM) data model and uses similar terminology as the DICOM object model. None of the properties or methods makes any changes to the data in the Varian System database. This fact guarantees safety against any unintended or erroneous script code.

The classes of the object model hide all the details of interacting with the database and creating the in-memory representations of the Eclipse data. Because the Scripting API is a .NET class library, all details of managing the memory and other low-level resources are also transparent to you when you create scripts.

Eclipse Scripting API Concepts

The most important concepts of the Eclipse Scripting API are described below.

Coordinate System and Units of Measurement

The Eclipse Scripting API uses the following coordinate systems and units of measurement.

Distances and Positions

In all methods and properties that work with distances and positions, the unit of measurement is millimeters. The positions in 3D space are returned using the DICOM coordinate system. Note that this differs from the Planning Coordinate system used in the Eclipse user interface, where the unit of measurement is centimeters. In addition, when the coordinate values are displayed in the Eclipse user interface, the following are taken into account:

- The possible user-defined origin of an image.
- The treatment orientation of the plan.
- The axis definition of the planning coordinate system.

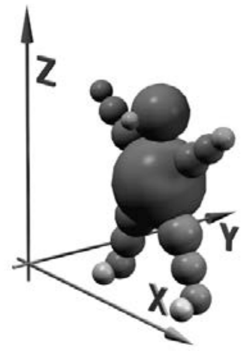

Figure 1 DICOM Coordinate System

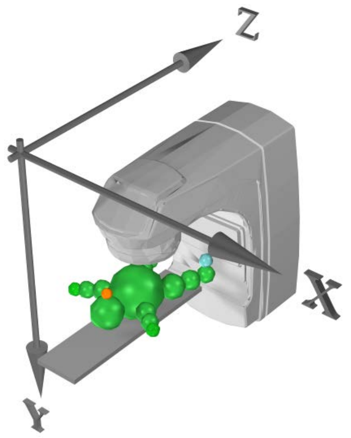

Figure 2 Standard Planning Coordinate System

The Eclipse Scripting API has methods that convert values from the DICOM coordinate system to the same representation that is used in the Eclipse user interface.

For more information on the display of 3D coordinates in the Eclipse user interface, refer to *Eclipse Photon and Electron Reference Guide*.

For more information on the DICOM coordinate system, refer to the DICOM standard.

Dose Values

In the Eclipse Scripting API, dose values are always represented with the separate `VMS.TPS.Common.Model.Types.DoseValue` type. In addition to the actual floating point value of the variable, this type also holds the measurement unit of the dose. The measurement unit can be Gy or cGy, depending on the selected clinical configuration. It can also be a percentage if relative dose is used.

Treatment Unit Scales

All methods and properties of the Eclipse Scripting API return the treatment unit and accessory properties in the IEC61217 scale. This feature allows you to create scripts despite the scale interpretation differences between treatment unit vendors.

User Rights and HIPAA

The Eclipse Scripting API uses the same user rights and HIPAA logging features as Eclipse. When a plug-in script is executed, the script applies the same user rights as were used to log into Eclipse.

If you execute a stand-alone executable script, the script code must provide a valid user name and password to authenticate itself to the system, or it can invoke the interactive login dialog of Eclipse.

According to HIPAA rules, a log entry is made for each patient opened by a standalone script. Additionally, the Eclipse Scripting API follows the rules of department categorization of ARIA RTM.

Working with Several Patients

The context of the running Eclipse instance is passed to plug-in scripts. They work only for the one patient that is selected in that context. In contrast, stand-alone executables can open any patient in the database. However, only the object model of a single patient is available at a time. The previous patient data must be explicitly closed before another patient is opened. If you try to access the data of a patient that has been closed, an access violation exception is generated.

Overview of the Object Model

The following diagram gives an overview of the Image-related objects in the Eclipse Scripting API.

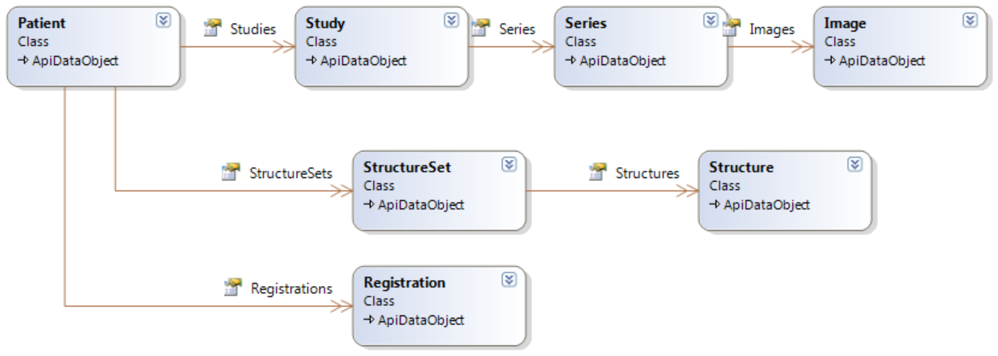

Figure 3 Image Data Model

The diagram contains the following objects:

- A Patient that has a collection of Study, StructureSet and Registration objects.
- A Study that has a collection of Series objects.
- A Series that has a collection of Image objects.
- A StructureSet that has a collection of Structure objects.

Another important section of the Eclipse Scripting API is the model of Plan-related objects shown in the diagram below.

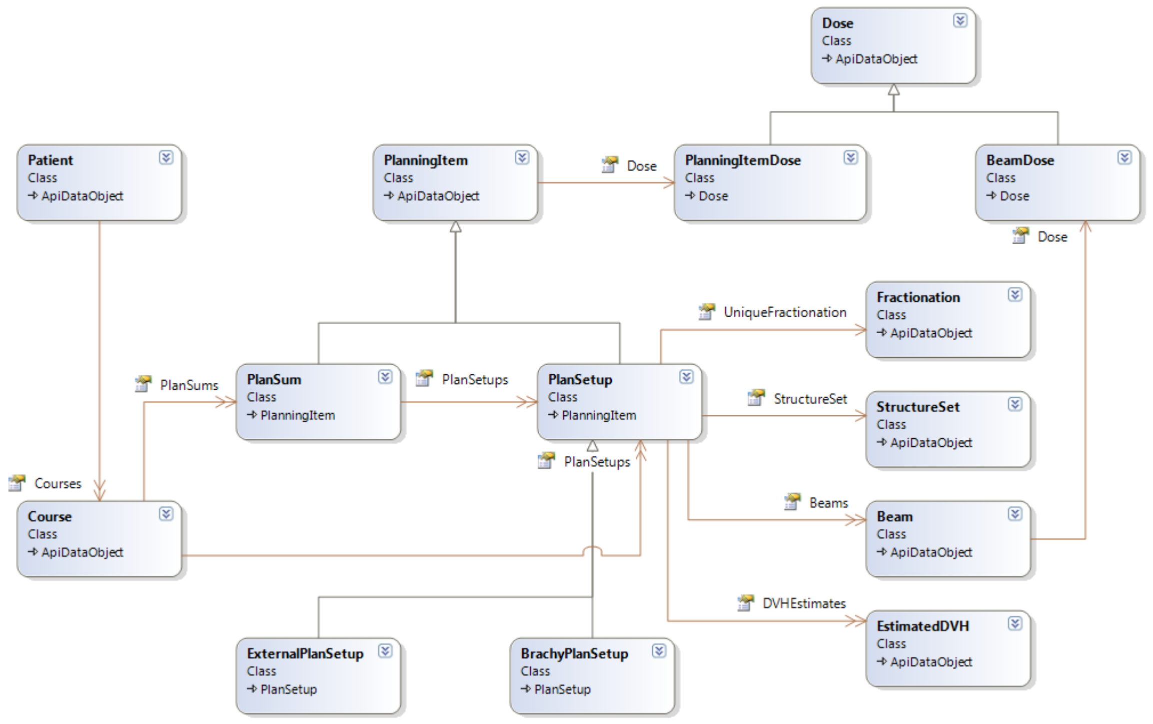

Figure 4 Plan Data Model

The diagram contains the following objects:

- A Patient that has a collection of Course objects.
- A Course that has a collection of PlanSetup and PlanSum objects. Each of them is derived from the common PlanningItem base class. Each PlanSetup object is either an ExternalPlanSetup or a BrachyPlanSetup.
- A PlanningItem class that has a direct (but nullable) relationship with a PlanningItemDose class.
- A PlanSetup that has a collection of Beam objects. Beam has a direct (but nullable) relationship with a BeamDose class.
- A PlanSetup that has a direct (but nullable) relationship with the Fractionation, StructureSet and EstimatedDVH objects.

The object model related to Plan optimization is visualized in Figure 5.

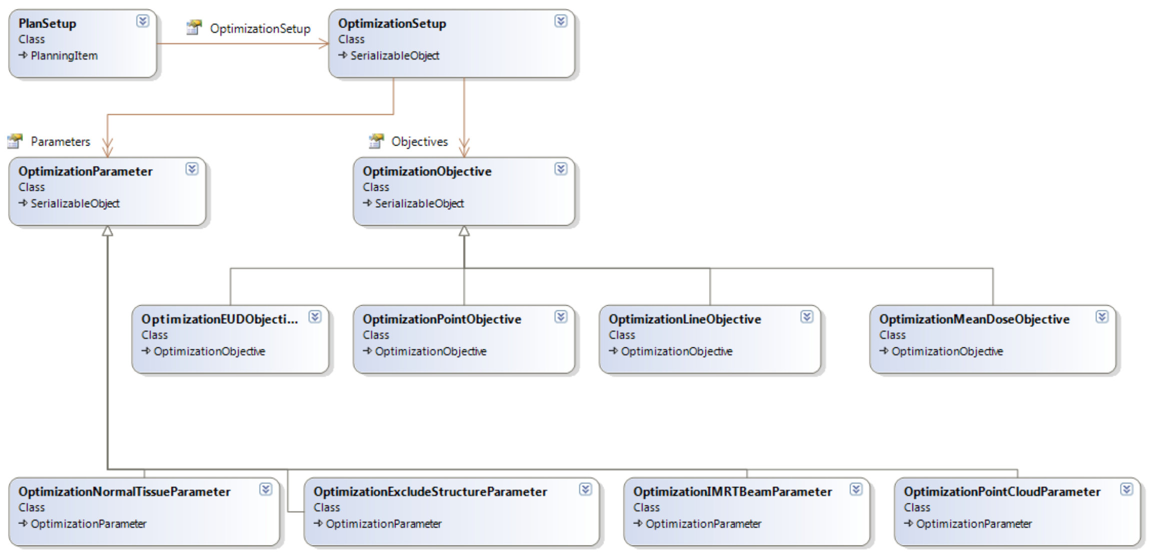

Figure 5 Plan Optimization Data Model

The diagram contains the following objects:

- A PlanSetup that has an association to the OptimizationSetup.
- An OptimizationSetup that has a collection of OptimizationParameter objects. Each OptimizationParameter object is either an OptimizationNormalTissueParameter, OptimizationExcludeStructureParameter, OptimizationIMRTBeamParameter, or OptimizationPointCloudParameter.
- An OptimizationSetup that has a collection of OptimizationObjective objects. Each object is either an OptimizationPointObjective, OptimizationEUDObjective, OptimizationLineObjective or OptimizationMeanDoseObjective

The next diagram shows the objects related to an individual Beam:

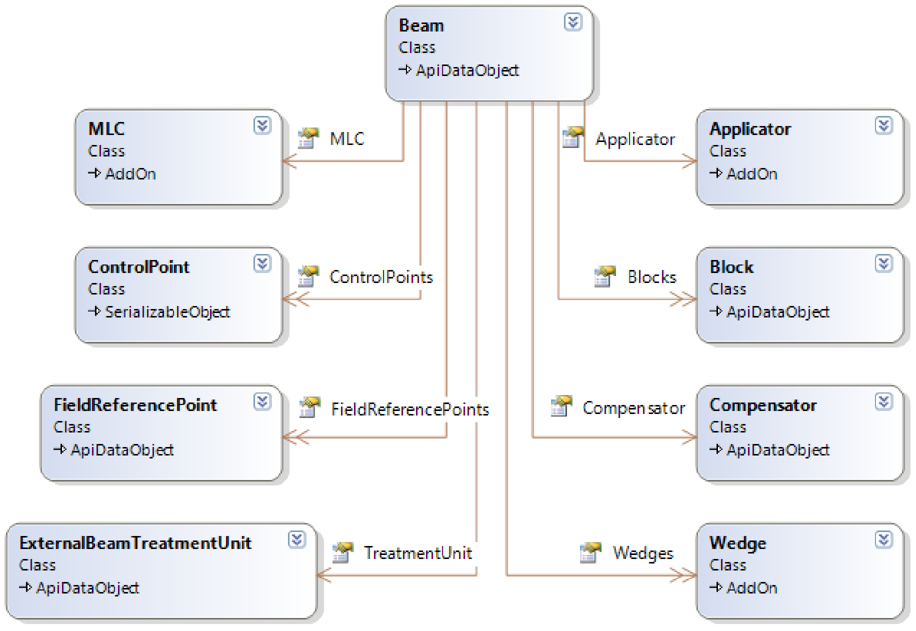

Figure 6 Beam Data Model

The diagram contains the following objects:

- An MLC and a ControlPoint collection of the Beam. Note that for proton beams, the control points are not currently supported and an empty collection is returned.
- An Applicator, a Compensator and a collection of Blocks and Wedges if defined for the Beam.
- A collection of FieldReferencePoint objects for the Beam.
- An ExternalBeamTreatmentUnit object that represent the treatment unit.

The next diagram shows the data model for brachytherapy plans:

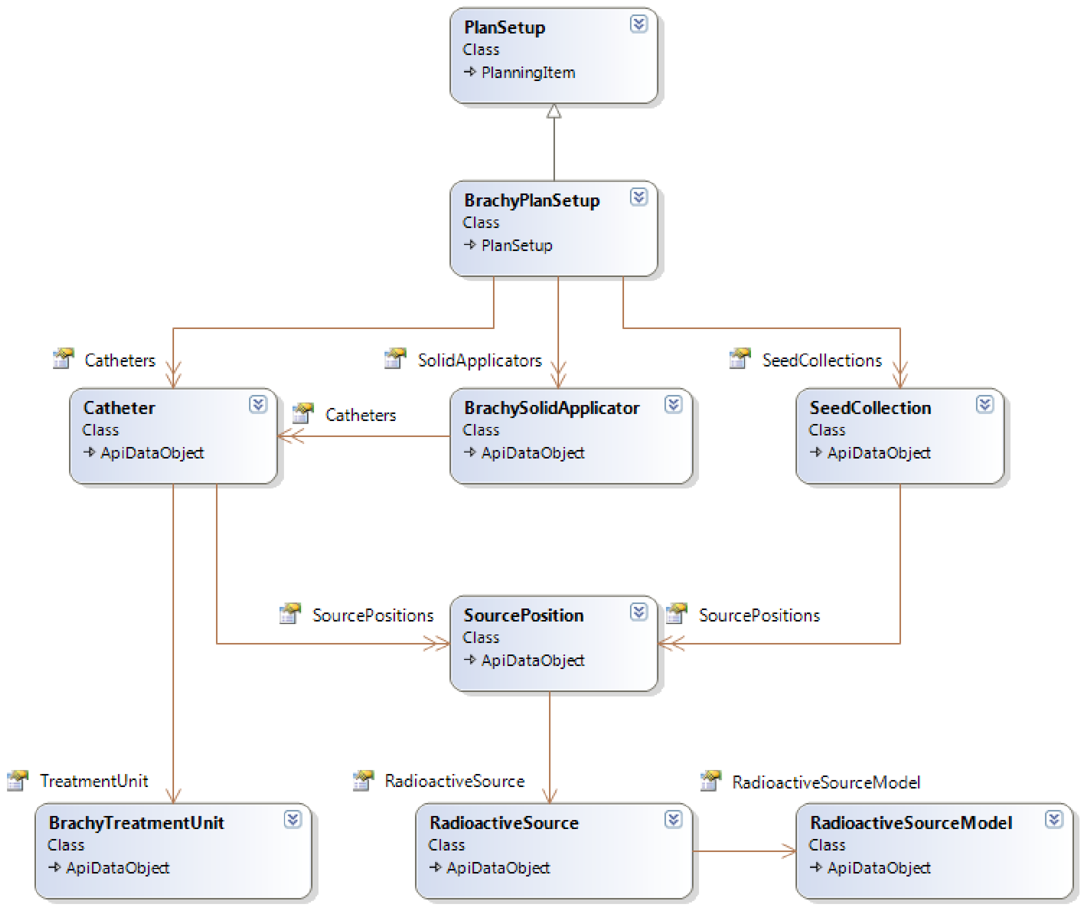

Figure 7 Brachytherapy Data Model

The diagram contains the following objects:

- A BrachyPlanSetup is derived from PlanSetup. The BrachyPlanSetup has a collection of Catheters, BrachySolidApplicators, and SeedCollections. Note that BrachyPlanSetups can be accessed through the Course in the same way as PlanSetups.
- A BrachySolidApplicator has a collection of Catheters.
- A Catheter (applicator channel central line or needle) has a BrachyTreatmentUnit and a collection of SourcePositions.
- A SeedCollection has a collection of SourcePositions.
- A SourcePosition has a RadioactiveSource.
- A RadioactiveSource has a RadioactiveSourceModel.

The properties of each object are described in detail in the *Eclipse Scripting API Online Help*.

Chapter 4 Getting Started with the Eclipse Scripting API

To get quickly started with the Eclipse Scripting API, you can:

8. Copy the code shown below to a file.
9. Save the file with a *.cs* extension on the hard disk of your workstation.

```
using System;
using System.Text;
using System.Windows;
using VMS.TPS.Common.Model.API;

namespace VMS.TPS
{
  class Script
  {
    public Script()
    {
    }
    public void Execute(ScriptContext context)
    {
      if (context.Patient != null)
      {
        MessageBox.Show("Patient id is " + context.Patient.Id);
      }
      else
      {
        MessageBox.Show("No patient selected");
      }
    }
  }
}
```

Figure 8 Sample Script Code

10. In Eclipse, select **Tools > Scripts**.
11. Select the **Directory: [path_to_your_own_scripts]** option.
12. To locate the script that you created, click **Change Directory**.
13. In the **Scripts** dialog box, select the script from the list and click **Run**. The script displays a message box which contains the ID of the patient that is open in Eclipse.

Chapter 5 Using Example Scripts

The Eclipse Scripting API includes example scripts for each of the supported script types. You can first copy the example scripts by using the Script Wizard, and then compile them by using Visual Studio.

If you do not have Visual Studio available, you can compile the examples with the MSBuild program, which is included in the Microsoft .NET framework.

Copying Example Scripts

To copy the example scripts to your own location:

1. From the **Start** menu, select **Varian > Eclipse Scripting API > Eclipse Script Wizard**.
2. Click the **Copy Example Scripts** tab.
3. To select a location for copying the example scripts, click **Browse**.
4. Click **Copy**. The example scripts are copied to the specified location.

Compiling Example Scripts

To compile the examples by using Visual Studio:

1. Open the Visual Studio project files.
2. Compile the examples.

After this, you can launch the example scripts.

To compile the examples by using MSBuild:

1. In the file browser, go to the directory where you copied the example scripts.
2. Open Command Prompt.
3. Enter the following information on the command line:
   - The path to the directory where *MSBuild.exe* is located.
   - The name of the project file.
   - Platform specification for x64.

   For example:

   ```
   C:\Windows\Microsoft.NET\Framework64\v4.0.30319\MSBuild.exe
   Example_DVH.csproj /p:Platform=x64
   ```
4. To compile the example, press ENTER.

Chapter 6 Creating Scripts

You can create scripts manually or by using the Script Wizard.

Creating Plug-in Scripts

The following sections give you step-by-step instructions on creating different types of plug-in scripts supported by the Eclipse Scripting API.

Creating Single-File Plug-ins with the Script Wizard

To create a single-file plug-in with the Script Wizard, follow these guidelines:

1. From the **Start** menu, select **Varian > Eclipse Scripting API > Eclipse Script Wizard**.
2. Enter a name for the new script.
3. Select the **Single-file plug-in** option.
4. To select the location for storing the script, click **Browse**. By default, the script is stored in the *Documents/Eclipse Scripting API* folder.
5. Click **Create**.
6. The Script Wizard creates the following folders in the location that you selected:
   - *Project* folder: Contains a script-specific sub-folder where the Microsoft Visual Studio project file is stored.
   - *Plugins* folder: Contains the source code file for the single-file plug-in.

   The Script Wizard launches Visual Studio.
7. Edit the source code file according to your needs. You can use Visual Studio and its IntelliSense support for editing the file, but they are not required.
8. You do not have to compile the plug-in, because Eclipse compiles it automatically on the fly.

Creating Binary Plug-ins with the Script Wizard

To create a binary plug-in with the Script Wizard, follow these guidelines:

1. From the **Start** menu, select **Varian > Eclipse Scripting API > Eclipse Script Wizard**.
2. Enter a name for the new script.
3. Select the **Binary plug-in** option.
4. To select the location for storing the script, click **Browse**. By default, the script is stored in the *Documents/Eclipse Scripting API* folder.
5. Click **Create**.

6. The Script Wizard creates the following folders in the location that you selected:
   - *Project* folder: Contains a script-specific subfolder where the Microsoft Visual Studio project file and source code file are stored.
   - *Plugins* folder: Contains the compiled plug-in dlls. From this folder, the dll can be loaded into Eclipse.

   The Script Wizard launches Visual Studio.
7. Edit the source code file according to your needs.
8. Compile the plug-in, for example, by using Visual Studio. The resulting plug-in dll is saved into the *Plugins* folder. Note that you can also use the MSBuild tool to compile the binary plug-in. For an example, see *Compiling Example Scripts*. For more information about MSBuild, refer to Microsoft documentation.

Creating Single-File Plug-ins Manually

If you want to create a single-file plug-in without the Script Wizard, follow the guidelines below. For an example of a source code file, see *Getting Started with the Eclipse Scripting API*.

1. Create an empty C# source code file.
2. Add the using statements for the `System` and `System.Windows` namespaces.
3. Add the using statements for the following namespaces:
   - `VMS.TPS.Common.Model.API`
   - `VMS.TPS.Common.Model.Types`
4. Add a namespace called `VMS.TPS`.
5. To the `VMS.TPS` namespace, add a public class called `Script`.
6. To the `Script` class, add a constructor without parameters, and a method called `Execute`.
7. Define the return type of the `Execute` method as `void`.
8. To the ‘`Execute`’ method, add the following parameters:
   - The context of the running Eclipse instance. The parameter type is `VMS.TPS.Common.Model.API.ScriptContext`.
   - A reference to the child window that Eclipse creates for the user interface components (optional). The parameter type is `System.Windows.Window`.
9. You do not have to compile the plug-in, because Eclipse compiles it automatically on the fly.

Creating Binary Plug-ins Manually

If you want to create a binary plug-in without the Script Wizard, follow these guidelines:

1. In Microsoft Visual Studio, create a new Class Library project. Select **x64** as the Solution Platform.
2. Create the source code in the same way as for a single-file plug-in. For instructions, see *Creating Single-File Plug-ins Manually*.

3. Use the following file name extension for the dll: *.esapi.dll*. In this way, Eclipse recognizes the plug-in and can load it.
4. Add references to the following class libraries of the Eclipse Scripting API:
   - `VMS.TPS.Common.Model.API.dll`
   - `VMS.TPS.Common.Model.Types.dll`.

   On the basis of this information, the dll can access the Eclipse Scripting API. These files are located in the installation directory of Eclipse.
5. Compile the plug-in into a .NET assembly (a dll), for example, by using Visual Studio.

For more information on how to create a .NET assembly and add references to class libraries, refer to Microsoft documentation.

Storing Plug-in Scripts

If you want to make the created scripts available for all workstations, store them into the *System Scripts* directory. The *System Scripts* directory is a shared directory on the Varian System server.

You can access the *System Scripts* directory by clicking the **Open Directory** button in the Script dialog.

Creating Stand-alone Executable Applications

The following sections give you step-by-step instructions on creating stand-alone executables supported by the Eclipse Scripting API.

Creating Stand-alone Executables with the Script Wizard

To create a stand-alone executable with the Script Wizard, follow these guidelines:

1. From the **Start** menu, select **Varian > Eclipse Scripting API > Eclipse Script Wizard**.
2. Enter a name for the new script.
3. Select the **Standalone executable** option.
4. To select the location for storing the script, click **Browse**.
5. Click **Create**.
6. The Script Wizard creates a *Projects* folder in the location that you selected. The folder contains a script-specific subfolder where the Microsoft Visual Studio project file and source code file are stored. The Script Wizard launches Visual Studio.
7. Edit the source code file according to your needs.

Creating Stand-alone Executables Manually

If you want to create stand-alone executables without the Script Wizard, follow these guidelines:

1. In Microsoft Visual Studio, create a new project file for the executable. Select **x64** as the Solution Platform.
2. Add references to the following class libraries of the Eclipse Scripting API:
   - `VMS.TPS.Common.Model.API.dll`
   - `VMS.TPS.Common.Model.Types.dll`.

   On the basis of this information, the dll can access the Eclipse Scripting API. These files are located in the installation directory of Eclipse.
3. In the main method of the executable file, use the static `CreateApplication` method to create an instance of the `VMS.TPS.Common.Model.API.Application` class. This class represents the root object of the data model. The `CreateApplication` method also initializes the Eclipse Scripting API.
4. Dispose of the instance when the stand-alone executable exits to free the unmanaged resources in the Eclipse Scripting API. For more information on disposing of objects, refer to Microsoft documentation of the IDisposable interface.
5. To the `CreateApplication` method, add the following parameters:
   - A user name and password for logging into the ARIA RTM system. If you do not define the user name or password (values remain null), the system shows a log-in dialog requesting the user credentials.
6. Use a single-threaded apartment (STA) as the COM threading model of the executable. The Eclipse Scripting API must only be accessed from a single thread that runs in the default application domain. For more information about threading and application domains, refer to Microsoft documentation.

The following is the code for a sample stand-alone executable in C# language:

```
using System;
using System.Linq;
using System.Text;
using System.Collections.Generic;
using VMS.TPS.Common.Model.API;
using VMS.TPS.Common.Model.Types;

namespace StandaloneExample
{
  class Program
  {
    [STAThread]
    static void Main(string[] args)
    {
      try
      {
        using (Application app = Application.CreateApplication(null, null))
        {
          Execute(app);
        }
      }
      catch (Exception e)
      {
        Console.Error.WriteLine(e.ToString());
      }
    }
    static void Execute(Application app)
    {
      string message =
        "Current user is " + app.CurrentUser.Id + "\n\n" +
        "The number of patients in the database is " +
        app.PatientSummaries.Count() + "\n\n" +
        "Press enter to quit...\n";
      Console.WriteLine(message);
      Console.ReadLine();
    }
  }
}
```

Figure 9 Sample Code for Stand-alone Executable

7. Compile the project. The stand-alone executable is ready to be run.

For more information on creating and compiling .NET applications, refer to Microsoft documentation.

Chapter 7 Launching Scripts

You can launch plug-in scripts from Eclipse, and stand-alone executables as any Windows application.

Launching Plug-in Scripts

To launch a plug-in script:

1. In Eclipse, select **Tools > Scripts**. The Scripts dialog box opens.
2. To locate the script that you want to run, select one of the following options:
   - **System Scripts:** The scripts that are available for all users are shown on the list.
   - **Directory: [path_to_your_own_scripts]**. Click **Change Directory** and select a folder. All files with the *.cs* or *.esapi.dll* file name extension become available on the list.
3. In the Scripts dialog, select the script file on the list.
4. Click **Run**.
5. If the execution of the script takes a very long time, you can click the **Abort** button. The execution of the script is aborted the next time the script accesses a property or method of the Eclipse Scripting API. Note that this procedure is meant only for recovering from programming errors and should not be considered a normal practice.

Launching Stand-alone Executable Applications

You can launch a stand-alone executable like any Windows application on the workstation where Eclipse is installed. You can also debug the stand-alone executable using normal Windows debugging tools.

Adding and Removing Favorite Scripts

You can add favorite scripts to the Eclipse External Beam and BrachyVision Tools menu and define keyboard shortcuts for them. To add a favorite script to the menu:

1. In Eclipse, select **Tools > Scripts**. The Scripts dialog box opens.
2. Select the script that you want to add to the menu.
3. Click **Add**…
4. A dialog box is opened. You can define a keyboard shortcut for the favorite script.
5. Click **OK**.

To remove a favorite script from the Tools menu:

1. In Eclipse, select **Tools > Scripts**. The Scripts dialog box opens.
2. Select a favorite script.
3. Click **Remove**.