varian

Image Registration and Segmentation Scripting API Reference Guide

Eclipse
BrachyVision

P1020793-002-B AUGUST 2025

Legal Information

Publication ID
P1020793-002-B

Publication Title
Image Registration and Segmentation Scripting API Reference Guide

Abstract
This document provides basic information for writing scripts to access the data model in the following Eclipse applications, version 16.1.

- Image Registration
- Contouring
- Smart Segmentation

This publication is the English-language original.

Notice
Information in this publication is subject to change without notice and does not represent a commitment on the part of Varian.

Other Legal Information
For other legal information refer to the instructions for use of the device.

Table of Contents

Introduction

About this Manual

The Registration and Segmentation Scripting Application Programming Interface (Registration and Segmentation Scripting API) is a programming interface and a software library for the Eclipse applications **Image Registration**, **Contouring**, and **Smart Segmentation**. It allows software developers to write scripts to access the data model in these applications. The scripts can be integrated into the application's user interfaces, or they can be run as stand-alone executables.

Compatibility / Document History

Ensure that your version of this document matches the Varian system release that you intend to deploy. All document versions are available on www.MyVarian.com.

Table 1 Document History

Varian System Release	Document Version	Date of Issue
15.5 and higher[1]	P1020793-001-A	May 2017
15.0, 15.1	P1015144-001-A	December 2015
13.5, 13.6, 13.7	P1006610B	March 2014
13.0	P1001047A	May 2013

For further information about products and applications, refer to the applicable customer release notes.

Who Should Read This Manual

This guide is written mainly for medical/technical personnel who wish to write custom scripts to be used in registration and segmentation applications. It is assumed that you are familiar with:

- Image Registration, Contouring, and Smart Segmentation applications
- Radiation oncology domain and concepts
- DICOM
- Software engineering practices
- Microsoft Visual Studio development environment
- Microsoft Visual C# programming language and object oriented development
- English

CAUTION: Before creating your own scripts, familiarize yourself with the Image Registration, Contouring, and Smart Segmentation user documentation, especially any safety-related information, cautions and warnings found throughout the documentation.

[1] higher, if this is the latest document version.

Related Documentation

- *Image Registration and Segmentation Instructions for Use*
- *Image Registration and Segmentation Algorithms Reference Guide*
- *RT and Imaging Online Help*

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.

The Registration and Segmentation Scripting API

About the Registration and Segmentation Scripting API

The Registration and Segmentation Scripting API is a Microsoft .NET class library that gives you read access to the data model and objects of the Image Registration, Contouring, and Smart Segmentation applications. It allows you to create scripts that let you retrieve data from the Varian System Database for ARIA® Radiation Therapy Management and create your own dose evaluations using this data. You can integrate the scripts into these applications, or you can run scripts as stand-alone executables.

The Registration and Segmentation Scripting API and scripting capabilities are very similar to the Eclipse and Portal Dosimetry scripting API. If you are already using one of these scripting APIs, many aspects of the Registration and Segmentation Scripting API will already be familiar to you.

WARNING: The authors of custom scripts are responsible for verifying the accuracy and correctness of the scripts.

Features

By using the Registration and Segmentation Scripting API, you can:

- Write custom scripts and integrate them into the respective application's user interface.
- Write stand-alone executable applications that leverage the Registration and Segmentation Scripting API.

Through the created scripts, you can access specific objects that are wrapped for more convenient use by the registration and segmentation applications, such as images:

- structures and structure sets
- rigid and non-rigid registrations

As all ARIA RTM applications that support scripting, the Registration and Segmentation Scripting API provides read-only access to the following RT domain base objects in the Varian System Database:

- Field and control point data
- Treatment records
- Projection and volume images
- Annotations, contours and labels on projection images
- Volumetric structures on volume images
- Rigid and non-rigid spatial registrations

The Registration and Segmentation Scripting API provides you also the following:

- A wizard that makes it easy to create new scripts
- Patient data protection that complies with HIPAA

- Support for ARIA RTM user authorization
- API documentation online help
- Example applications
- Creation of transient dose analysis

System Requirements

The basic system requirements of the API are the same as those of the related applications. To create scripts with the Registration and Segmentation Scripting API, you need one or several of the following applications.

- Image Registration
- Contouring
- Smart Segmentation

For more information, refer to the customer release notes.

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

Supported Script Types

Two types of scripts are supported: plug-ins and executable applications.

Plug-ins

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

Two types of plug-ins are supported:

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

The application launching the script 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 application instance as an input parameter. The context contains the patient, as well as the currently active image or registration and the currently selected structure of the application that has launched the script.

Plug-in scripts work only for one patient at a time.

Users of Registration and Segmentation scripts can define favorite plug-in scripts. These scripts are directly accessible in the tools menu of the application or optionally by a user defined shortcut key. Other plug-in scripts need to be started using the **Scripts** dialog.

Executable Applications

A stand-alone executable is a .NET application that references the Registration and Segmentation 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 the calling application, the stand-alone executable can scan the database and open any patient.

Registration and Segmentation Scripting API Object Model

Registration and Segmentation Scripting API Data Model

The data model exposed in the Registration and Segmentation Scripting API is a collection of .NET classes with properties and methods. The class hierarchy is an abstraction over the ARIA RTM data model and closely resembles 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 consequence of this read-only approach is that it is not possible to create, delete or modify objects on the Varian System Database by scripts. If a script needs to store data persistently, this needs to be done by another mechanism, e.g. by using files.

The classes of the object model hide all the details of interacting with the database and creating the in-memory representations of the application 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 the script author.

Registration and Segmentation Scripting API Concepts

The most important concepts of the Registration and Segmentation Scripting API are described below.

Coordinate System and Units of Measurement

The Registration and Segmentation 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. Spatial registrations are always between two DICOM coordinate systems.

Image pixels are accessed using the pixel coordinate system as shown below, where the origin is in the center of the top left pixel of the first image plane. Remember that pixel indices are zero based and that for projection images the only valid image plane index is zero. The scripting API provides conversion methods between the DICOM and the pixel coordinate systems.

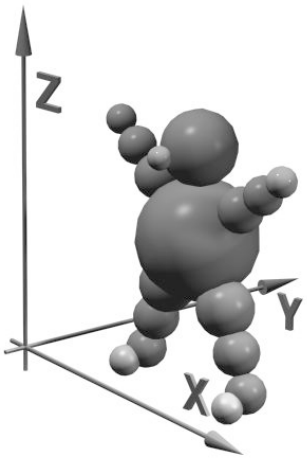

Figure 1 DICOM Coordinate System

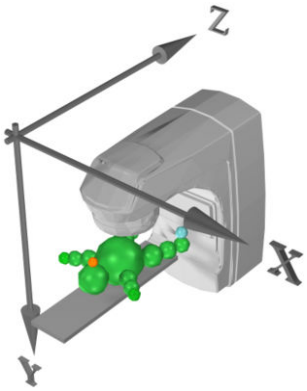

Figure 2 Standard Planning Coordinate System

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

Pixel Values

Raw integer pixel values of 2D images or image slices can be converted to the required unit representation such as HU using the following method of the Frame class:

```
public double VoxelToDisplayValue(
        int voxelValue
)
```

Treatment Unit Scales

All methods and properties of the Registration and Segmentation Scripting API return the treatment unit and accessory properties in the IEC61217 scale. This feature makes it simpler to create scripts because the differences in scale interpretation between the treatment unit vendors do not need to be taken into account.

Images and Frames

The Varian System Database supports multi-frame images. This influences also the Registration and Segmentation Scripting API. Every image consists of two parts:

- Container object of type ‘Image’. The image contains data that is constant over all frames.
- Collection of ‘Frame’ objects, aggregated to the image object. The frames contain data that can vary from frame to frame. Most of the image data is contained in the frames, for example size, resolution, and pixel data.

Furthermore, for convenience in navigation, in Registration and Segmentation applications the Image objects are additionally wrapped by a MIRSImage object that have for example a property MIRSRegistrations to navigate to registrations or other images related through registrations. MIRSImage object always have images with only one frame and thus have also a Frame property to access this one and only frame.

Scripting API Layering

The Registration and Segmentation Scripting API classes can be grouped into two layers, represented by two namespaces.

- VMS.CA.Scripting namespace The classes in this namespace provide common objects and functionality for all ARIA RTM applications that support scripting. The classes and the relations between the classes are similar – but not identical - to the classes defined by the Eclipse scripting API.
- VMS.IRS.Scripting namespace The classes in this namespace extend and enrich the base object model by classes specific to the Registration and Segmentation applications. They provide a data model that is similar to the data representation in the user interface of the registration and segmentation applications.

Note: Extension classes in the VMS.IRS.Scripting namespace never duplicate existing functionality from the aggregated core class. Instead, the aggregated core object can be accessed using a property.

Example: Access pixel data of an object of type ‘MIRSImage’

```
MIRSImage mirsImage = (get the image);
 Frame frame = mirsImage. Frame;
 ushort[,] buffer = new ushort[frame.XSize, frame.YSize];
 frame.GetVoxels(0, buffer);
```

User Rights and HIPAA

The Imaging and Registration Scripting API uses the same user rights and HIPAA logging features as the corresponding applications. If you have logged into the application with certain user rights, and you execute a plug-in script, the script applies the same user rights as you have in the corresponding application.

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

Working with Several Patients

The plug-in scripts gain access to the context of the running application instance. They work only for the one patient that is selected in that context. In contrast, the 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 thrown.

Overview of the Registration and Segmentation Object Model

The following class diagram gives an overview on the Registration and Segmentation Scripting API object model, focusing on the objects specific to registration and segmentation applications:

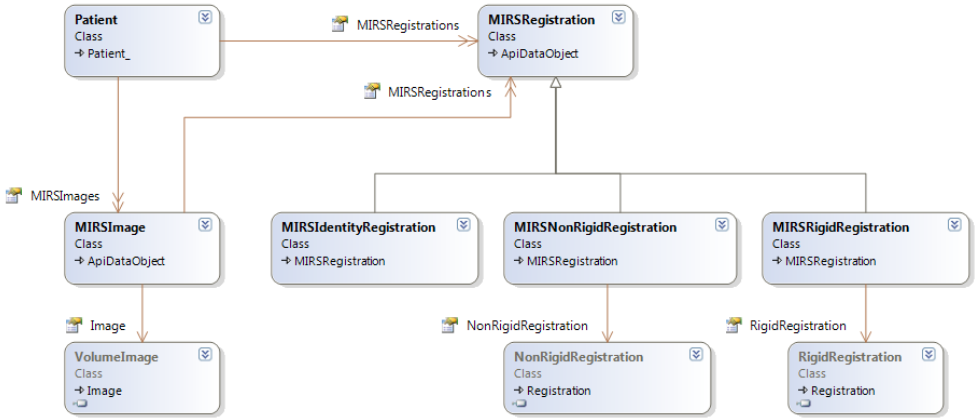

Figure 3 Registration and Segmentation Scripting API Object Model

The diagram contains the following objects:

- *Patient* having collections of MIRSImages and MIRSRegistrations
- *MIRSImage* having properties for accessing the frame of reference and the image's frames, and a collection MIRSRegistrations containing registrations to related images.
- *MIRSRegistration* having a source image, a registered image and a registration status. The source image is the image being deformed or rigidly transformed in order to fit the registered image. The registered image is the target image, sometimes refered to as static image also because it is not transformed.
- *MIRSRegistration* objects come in three concrete forms, the Identity (MIRSIdentityRegistration), the non-rigid registration (*MIRSNonRigidRegistration*, having a *VMS.CA.Scripting.NonRigidRegistration* that contains the actual vector field and pre- and post-transforms) and the rigid registration (*MIRSRigidRegistration*, having a *VMS.CA.Scripting.RigidRegistration* that contains the transformation).

Overview of the Core Object Model

The following class diagram gives an overview of the Image related objects in the core scripting API object model:

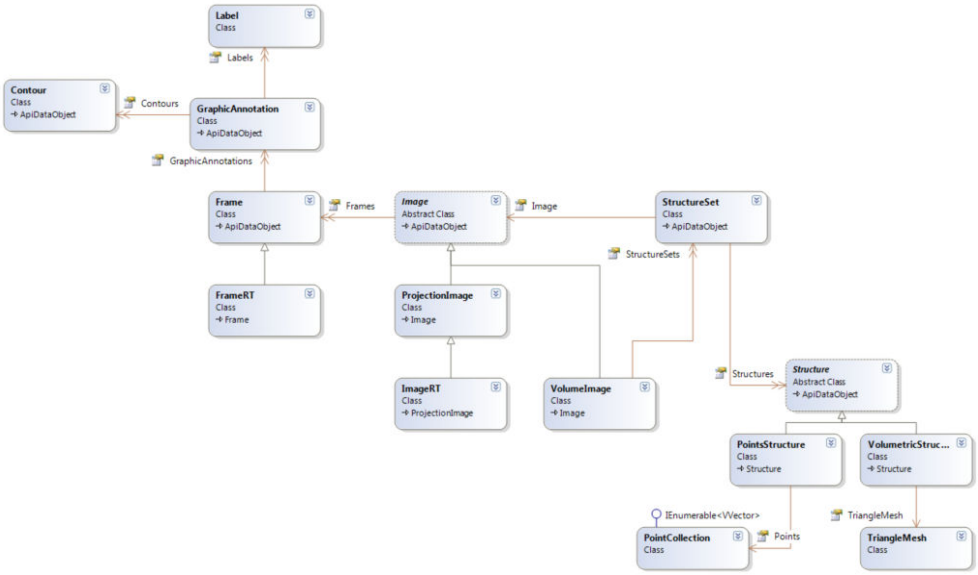

Figure 4 Image Data Model

The diagram contains the following objects:

- *Image* having a collection of frames. Represents a projection or a volumetric image.
- *ProjectionImage* representing a projection image (non-volumetric image).
- *ImageRT* representing an RT image.
- *VolumeImage* having a collection of structure sets. Represents a volumetric image or a slice of a volumetric image.
- *Frame* having a collection of graphic annotations. Represents a frame. A frame is always part of an image and contains the data that can vary from frame to frame in a multi-frame image.
- *FrameRT* representing a frame in an RT image.
- *GraphicAnnotation* having a collection of contours and labels.
- *Contour* representing a planar contour on a projection image.
- *Label* representing a text label on an image.
- *StructureSet* having a collection of structures.
- *Structure* representing a volumetric structure (segment, for example organ) or a point set.
- *PointsStructure* having a collection of points. Represents a point set – a collection of points in the volume image, for example markers.
- *VolumetricStructure* having a triangle mesh.
- *TriangleMesh* representing a triangle mesh.

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

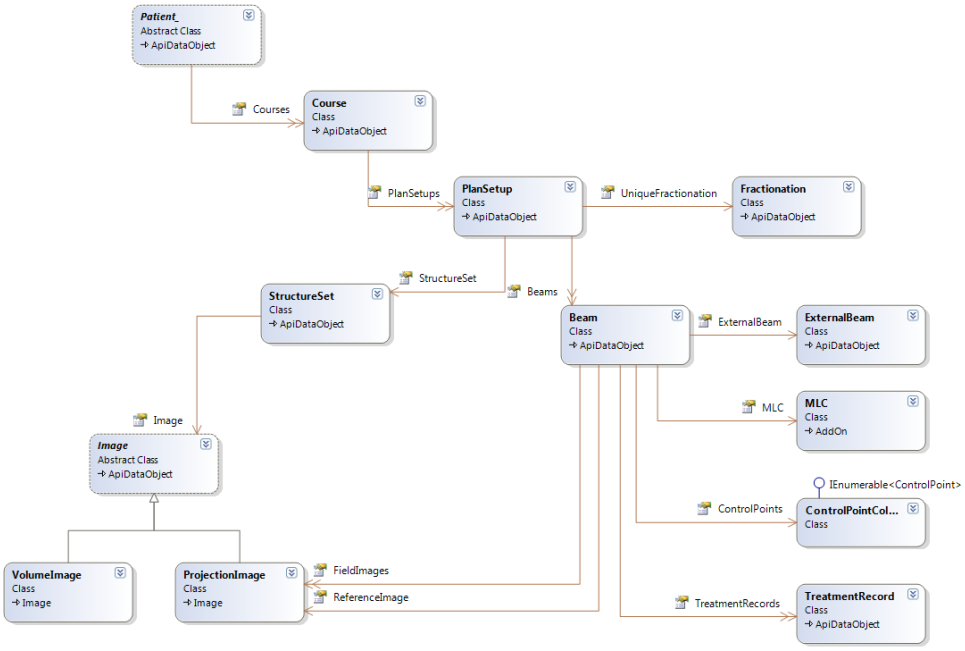

Figure 5 Plan Data Model

The diagram contains the following objects:

- *Patient* having a collection of courses.
- *Course* having a collection of plan setups.
- *PlanSetup* having a collection of beams, a structure set (representing the planning volume image if not null) and a fractionation.
- *Beam* having a collection of beam images, control points and treatment records, an assigned primary reference image, the external beam for treatment (for example a clinac) and an MLC.
- *ControlPoint* representing a point in a planned sequence of treatment beam parameters.
- *ExternalBeam* representing a machine or device for treatment.
- *MLC* representing a Multileaf Collimator.
- *TreatmentRecord* representing recorded data of a dose delivery performed.

The next diagram shows the objects related to the DICOM Patient – Study – Series model:

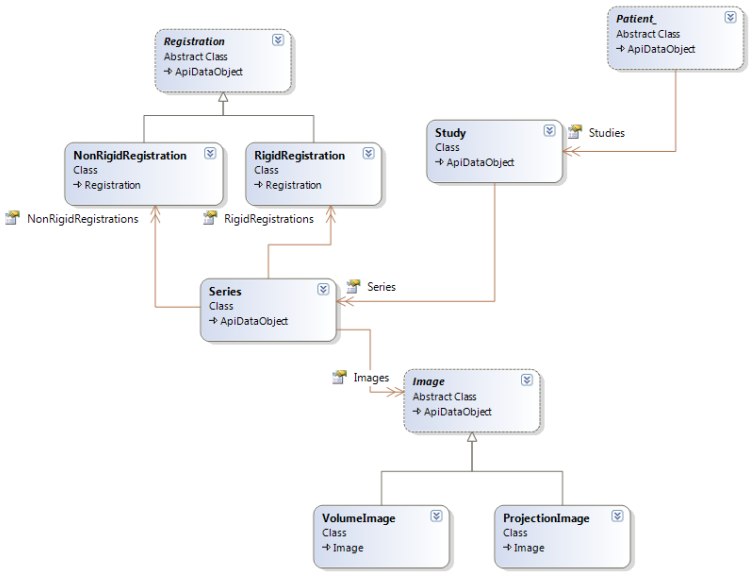

Figure 6 DICOM Patient-Study-Series Data Model

The diagram contains the following objects:

- *Patient* containing a collection of studies.
- *Study* containing a collection of series.
- *Series* containing a collection of registrations (rigid and non-rigid) and images. A series always contains one single type of objects, for example only rigid registrations or only volume images.
- *Registration* representing a rigid or non-rigid spatial registration.
- *RigidRegistration* representing a rigid spatial registration, represented by a transformation matrix.
- *NonRigidRegistration* representing a non-rigid spatial registration, represented by a vector field.

Getting Started

Getting Started with the Registration and Segmentation Scripting API

To get quickly started with the Registration and Segmentation Scripting API, you can:

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

```
using System;
using System.Linq;
using System.Text;
using System.Windows;
using System.Collections.Generic;
using VMS.CA.Scripting;

namespace VMS.IRS.Scripting {

  public 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");
      }
    }
  }
}
```

3. In the application, select **Tools** > **Scripts**.
4. To locate the script that you created, click **Change Directory**.
5. 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 the application.

Note: It is recommended to use the Script Wizard to create a skeleton for a script. Refer to Creating Scripts on page 19 on instructions how to use the script wizard.

Using Examples

Copying Example Scripts

The Registration and Segmentation Scripting API includes example scripts for each of the supported script types. You can use the Script Wizard to copy the example scripts:

1. From the **Tools** menu in the application, select **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.
5. Open the Visual Studio project files, compile the examples, and launch them.

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.

Use the following command line to compile the examples:

```
C:\Windows\Microsoft.NET\Framework\[version]\MSBuild.exe
VisualizeVectorField.csproj /p:Platform=x64
```

Note: It is recommended to install *Microsoft Visual Studio* for writing scripts. Visual Studio provides code completion and online help on methods and parameters as you type the code (IntelliSense). Visual C# is available free of charge with *Microsoft Visual Studio Express Editions* from the Microsoft home page.

Creating Scripts

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 Registration and Segmentation 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 **Tools** menu in the application, select **Script Wizard**.
2. Enter a name for the new script.
3. Select the **Single-file plug-in** option.
4. Select the **user interface library** to use in the script.
5. To select the location for storing the script, click **Browse**. By default, the script is stored in the *Documents/Registration and Segmentation Scripting API* folder.
6. Click **Create**.
7. 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 is stored. This project file is needed to provide IntelliSense support when the script code is edited in Microsoft Visual Studio.
   - *Plugins folder:* Contains the source code file for the single-file plug-in.

   The Script Wizard launches Microsoft Visual Studio if desired. After the project is loaded in Visual Studio, double click on the script file in the Visual Studio solution explorer to open the file.
8. 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.
9. You do not have to compile the plug-in, because the application 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 *Tools* menu in the application, select *Script Wizard*.
2. Enter a name for the new script.
3. Select the **Binary plug-in** option.
4. Select the *user interface library* to use in the script.
5. To select the location for storing the script, click **Browse**. By default, the script is stored in the *Documents/Registration and Segmentation Scripting API* folder.

6. Click **Create**.
7. 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 the application.

   The Script Wizard launches Visual Studio.
8. Edit the source code file according to your needs.
9. 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 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 Imaging and Registration 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.CA.Scripting`
4. Add a namespace called `VMS.IRS.Scripting`.
5. To the `VMS.IRS.Scripting` 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 instance. The parameter type is `VMS.IRS.Scripting.ScriptContext`.
   - A reference to the child window that the application 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 the application 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. Create the source code in the same way as for a single-file plug-in. For instructions, see Creating Single-File Plug-ins Manually on page 20.
2. Add references to the following class libraries of the Registration and Segmentation Scripting API:
   - `VMS.CA.Scripting.dll`

- `VMS.IRS.Scripting.dll`

  On the basis of this information, the dll can access the Registration and Segmentation Scripting API. These files are located in the installation directory of the corresponding application.

3. 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.

Debugging Binary Plug-in Scripts

To debug plug-in scripts, Microsoft Visual Studio is required. Make the following configuration in Visual Studio to debug a plug-in script:

1. Load the plug-in script project in Visual Studio.
2. In the solution explorer of Visual Studio, right click the plug-in project and select the **Properties** menu entry in the context menu. The project property pages are shown.
3. In the **Debug** tab, section **Start Action** choose the **Start external program** option and enter the full installation path of the application.
4. In the **Start Options** section, enter `/standalone` as **command line argument** and enter the installation path of the application as **working directory**.

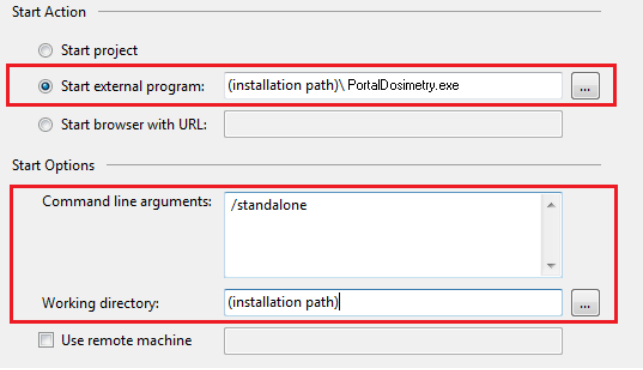

Figure 7 Debugging Properties

5. Set breakpoints in the script as desired and start debugging mode with the F5 key. The applications starts up. In the **Tools-Scripts** dialog, select the script being debugged and click **Run**. Script execution stops at the defined breakpoints and the Visual Studio debugger is activated.

For more information on how to debug C# code, refer to Microsoft Visual Studio documentation.

Creating Stand-alone Executable Applications

The following sections give you step-by-step instructions on creating stand-alone executables supported by the Registration and Segmentation 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 **Tools** menu in the application, select **Script Wizard**.
2. Enter a name for the new script.
3. Select the **Standalone executable** option.
4. Select the **user interface library** to use in the script.
5. To select the location for storing the script, click **Browse**.
6. Click **Create**.
7. 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.
8. 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. Create a new project file for the executable.
2. In the main method of the executable file, use the static *CreateApplication* method to create an instance of the *VMS.IRS.Scripting.Application* class. This class represents the root object of the data model. The *CreateApplication* method also initializes the Registration and Segmentation Scripting API.
3. Call the Dispose() method of the instance when the stand-alone executable exits to free the resources in the Registration and Segmentation Scripting API. For more information on disposing of objects, refer to Microsoft documentation of the IDisposable interface.
4. 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.
5. Use a single-threaded apartment (STA) as the COM threading model of the executable. The Registration and Segmentation 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.CA.Scripting;
using VMS.IRS.Scripting;

namespace StandaloneExample {

  static 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();
    }
  }
}
```

6. Insert an application configuration file (*app.config*) to the project. The application configuration file defines the location of the registration and segmentation application's binary files. The following shows the required settings. Remember to change the path according to the installation location of the registration and segmentation application on your workstation:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <appSettings>
    <add key="AssemblyPath"
        value="C:\Program Files
(x86)\Varian\ProductLine\Workspaces\VMS.IRS.Workspace\"/>
  </appSettings>
</configuration>
```

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.

Launching Scripts

Launching Scripts

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

Launching Plug-in Scripts

To launch a plug-in script:

1. In the application, select **Tools** > **Scripts**.

   The Scripts dialog box opens.
2. To locate the script that you want to run, select it from the list of scripts available in the scripts directory. To change the script directory:

   - Click **Change Directory** and select a folder. All files with the .cs extension or with extension *.dll* become available on the list.
3. In the Scripts dialog, select the script file on the list.
4. Click **Run**.

Launching Stand-alone Executable Applications

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

Index