Parametric Technology Corporation

Creo® View 2.0
Web Toolkit Developer's Guide

March 2012

Parametric Technology Corporation, 140 Kendrick Street, Needham, MA 02494 USA

Contents

About This Guide

This section contains information about the contents of this Developer's guide and the conventions used.

Purpose

The *Creo View Web Toolkit Developer's Guide* describes how to use Creo View Web Toolkit, the JavaScript customization toolkit for Creo View from PTC (Parametric Technology Corporation). Creo View Web Toolkit provides customers and third-parties the ability to expand Creo View capabilities by writing JavaScript code and seamlessly integrating the resulting application into Creo View.

This manual introduces Creo View Web Toolkit, its features, and the techniques and background knowledge users require to use it.

Related Documentation

The documentation for Creo View Web Toolkit includes:

- Online reference documentation—Describes Creo View Web Toolkit function syntax. The reference documentation is available at `<creo_view_api_loadpoint>/documentation/web`

Conventions

The following table lists conventions and terms used throughout this book.

Convention	Description
UPPERCASE	Creo View-type menu name (for example, PART).
Boldface	Windows-type menu name or menu or dialog box option (for example, **View**), or utility (for example, **promonitor**). Function names also appear in boldface font.
`Monospace (Courier)`	File names and code samples appear `in courier font`.
SMALLCAPS	Key names appear in smallcaps (for example, ENTER).
Emphasis	Important information appears in *italics*. Italic font also indicates function arguments.

Convention	Description
Choose	Highlight a menu option by placing the pointer on the option and pressing the left mouse button.
Select	A synonym for “choose” as above, Select also describes the actions of selecting elements on a model and checking boxes.

- Important information that should not be overlooked appears in notes like this.

  Note: All references to mouse clicks assume use of a right-handed mouse.

Documentation for PTC Products

You can access PTC documentation using the following resources:

- Product CD -- All relevant PTC documentation is included on the CD set.
- Reference Documents Web Site -- All books are available from the Reference Documents link of the PTC Web site at http://www.ptc.com/appserver/cs/doc/refdoc.jsp. On the Web site, choose the product or document type.

  A Service Contract Number (SCN) is required to access the PTC documentation from the Reference Documents Web site. For more information on SCNs, see Technical Support at

  http://www.ptc.com/support/index.htm.

Technical Support

Contact PTC Technical Support via the PTC Web site, phone, fax, or e-mail if you encounter problems using Creo View Web Toolkit or the product documentation.

For complete details, refer to "Contacting Technical Support" in the *PTC Customer Service Guide*. This guide can be found on the PTC Web site at:

http://www.ptc.com/support/cs_guide/cs_guide.pdf

You must have a Service Contract Number (SCN) before you can receive technical support. If you do not have an SCN, contact PTC Maintenance Department using the instructions found in your *PTC Customer Service Guide* under "Contacting Your Maintenance Support Representative".

Comments

PTC welcomes your suggestions and comments on its documentation. You can submit your feedback through the online survey form at the following URL:

http://www.ptc.com/go/wc_pubs_feedback

Third-Party Products

Examples in this guide referencing third-party products are intended for demonstration purposes only. For additional information about third-party products, contact individual product vendors.

Code Examples

Some code examples in this guide have been reformatted for presentation purposes and, therefore, may contain hidden editing characters (such as tabs and end-of-line characters) and extraneous spaces. If you cut and paste code from this manual, check for these characters and remove them before attempting to use the example in your application.

1

Fundamentals

This chapter describes the basic concepts and installation and configuration of Creo View Web Toolkit.

Introduction to Creo View Toolkit

Creo View Toolkits enable you to embed and control Creo View technology within Web pages and other applications to share product data with customers through interactive portals.

You can use Creo View Toolkit to:

- Embed Creo View tools and data in a custom in-house application.
- Use Creo View tools and data in an extended enterprise Web portal.
- Embed Creo View tools in a custom environment and distribute that application.
- Provide custom integrations of Creo View into Web and Java applications developed by a software services company.

The types of customizations available for Creo View are:

- Creo View Web Toolkit—JavaScript (Web page) APIs
- Creo View Java Toolkit—Java (embedding in a Java application) APIs
- Creo View Office Toolkit—Visual Basic APIs

Installation Requirements

This section describes the prerequisites and system requirements for installing Creo View Web Toolkit.

Programming Language support

For information on programming language support, refer to the Creo View Toolkit Software matrix at http://www.ptc.com/appserver/cs/doc/refdoc.jsp.

System Requirements

A software matrix on the PTC Web site lists the combinations of platforms, operating systems, and third-party products that are certified for use with Creo View Toolkit on Windows. To obtain a copy of the latest software matrix, go to:

http://www.ptc.com/appserver/cs/doc/refdoc.jsp

You are directed to the PTC Online Support Web page for reference documents. For your document search criteria, select your product from the Product list, select the current release from the Release list, and select Software Matrices from the Document Type list.

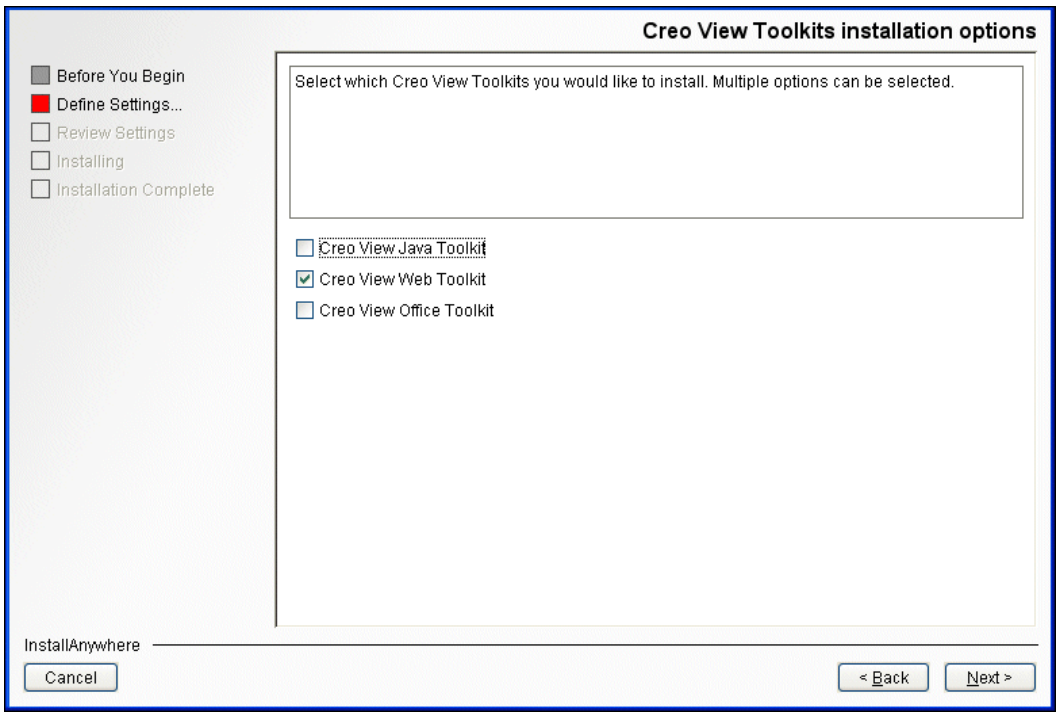
Installing Creo View Toolkit

Remember the following points during the installation:

- Click **Previous** at anytime during the installation process to revise the information that you have provided.
- Click **Cancel** at anytime to stop the installation. You are prompted for confirmation.

Perform the installation as follows:

1. Insert the Creo View Toolkit CD-ROM. If autorun is enabled for your CD-ROM drive, the *setup.vbs* file starts automatically. Otherwise, start Windows Explorer, browse to the CD-ROM drive, and double-click the icon for *setup.vbs*. The **Select Language** dialog box opens.
2. Select the required language and click **OK**. The **Before You Begin** window appears.
3. Review the information and click **Next**. The **PTC Customer License Agreement** window appears.
4. Click **I Accept the License Agreement Terms and Conditions** to proceed with the installation.
5. Click **Next**. The **Select Directory** window appears.
6. Click **Browse** to specify the location for the installation. You are prompted for confirmation if you want to create a new directory.
7. Click **Yes**. The **Creo View Toolkit installation options** window appears.



8. Select the Creo View Toolkit to be installed:
   - Creo View Java Toolkit—Java APIs for Creo View
   - Creo View Web Toolkit—Web APIs for Creo View
   - Creo View Office Toolkit—Visual Basic APIs for Creo View

   Note: You should install only the Toolkit that you have purchased.
9. Click **Next**. The **Review Settings** window appears.
10. Click **Install** to start the installation process. When complete, the **Installation Complete** window appears.
11. Click **Done**.

After the installation is complete, the following directories are created in the installation folder:

- redist
- web

- documentation
- installer
- demodata

Configuration of Creo View Web Toolkit

After installing Creo View Web Toolkit, install the Creo View client as follows:

1. Browse to `<creo_view_api_loadpoint>/redist`, where, `<creo_view_api_loadpoint>` is the location where you have installed Creo View Web Toolkit. The following files are available:

- `CreoView_32.exe`—Creo View client installer for a 32-bit platform.
- `CreoView_64.exe`—Creo View client installer for a 64-bit platform.
- `CreoView_Express_32.exe`—Creo View Express client installer for a 32-bit platform.
- `CreoView_Express_32_64.exe`—Creo View Express client installer for a 64-bit platform.

The Creo View Consumer files are located in the `<creo_view_api_loadpoint>/redist/consumer` folder. The following files are available:

- `CreoView_Consumer_32.exe`—Creo View Consumer client installer for a 32-bit platform.
- `CreoView_Consumer_32_64.exe`—Creo View Consumer client installer for a 64-bit platform.
- `consumer.cab`—Creo View Consumer cabinet file that can be placed on web servers.

2. Run `CreoView_32.exe` to install the latest version of the Creo View client available with Creo View Web Toolkit. Run `CreoView_64.exe` on a 64-bit platform.

   Note: If you install Creo View Express client not all functionality supported by Creo View Web Toolkit will be available.

Alternatively,

To install the Creo View client:

1. Browse to `<creo_view_api_loadpoint>/web`.
2. Open one of the sample HTML files provided, for example, `installTest.html`.
3. You are prompted to install a Creo View Client on your computer, if it is not already installed. Otherwise, you are prompted to upgrade to the latest version of the viewer available with Creo View Web Toolkit.

About the User Interface

You can launch the Creo View client using the Creo View Toolkit applications in:

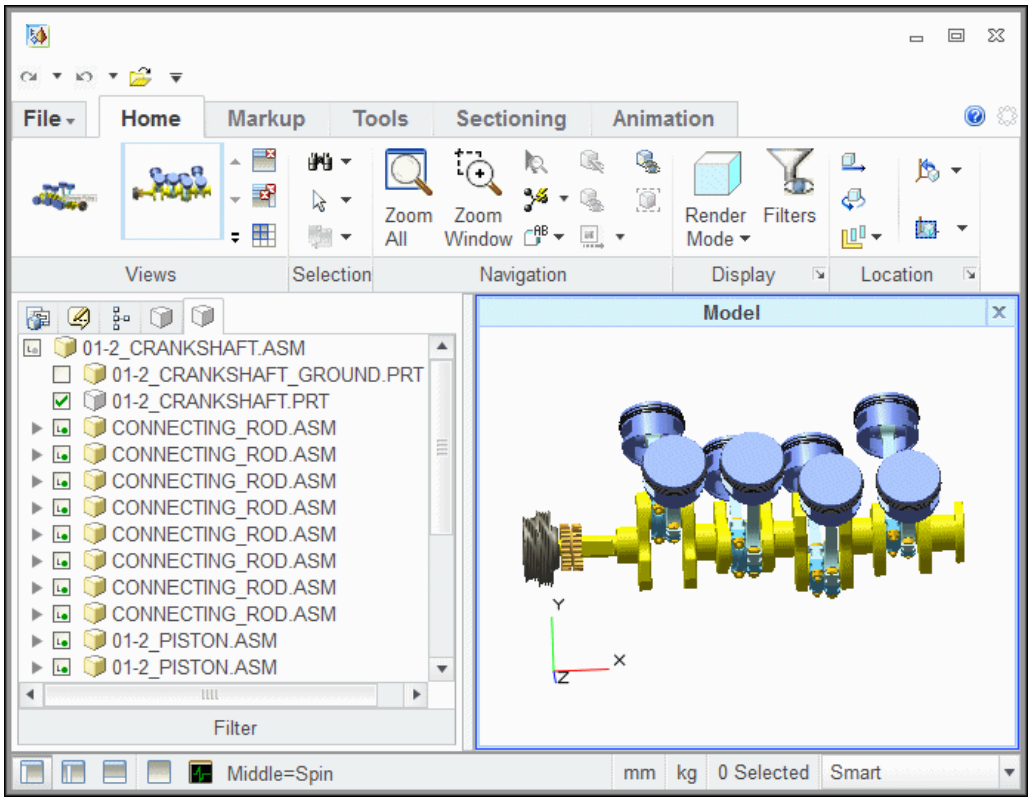
- GUI mode—Graphics User Interface mode
- Non-GUI mode—Non-graphics User Interface mode where the Graphical User Interface for the Creo View client is not shown.

This section describes these modes in detail. It also describes how you can customize the user interface using the API applications.

GUI Mode

When the application starts Creo View in this mode, the Viewing or Graphics area is displayed along with the Creo View User Interface (UI) as shown in the following figure

.



The UI consists of:

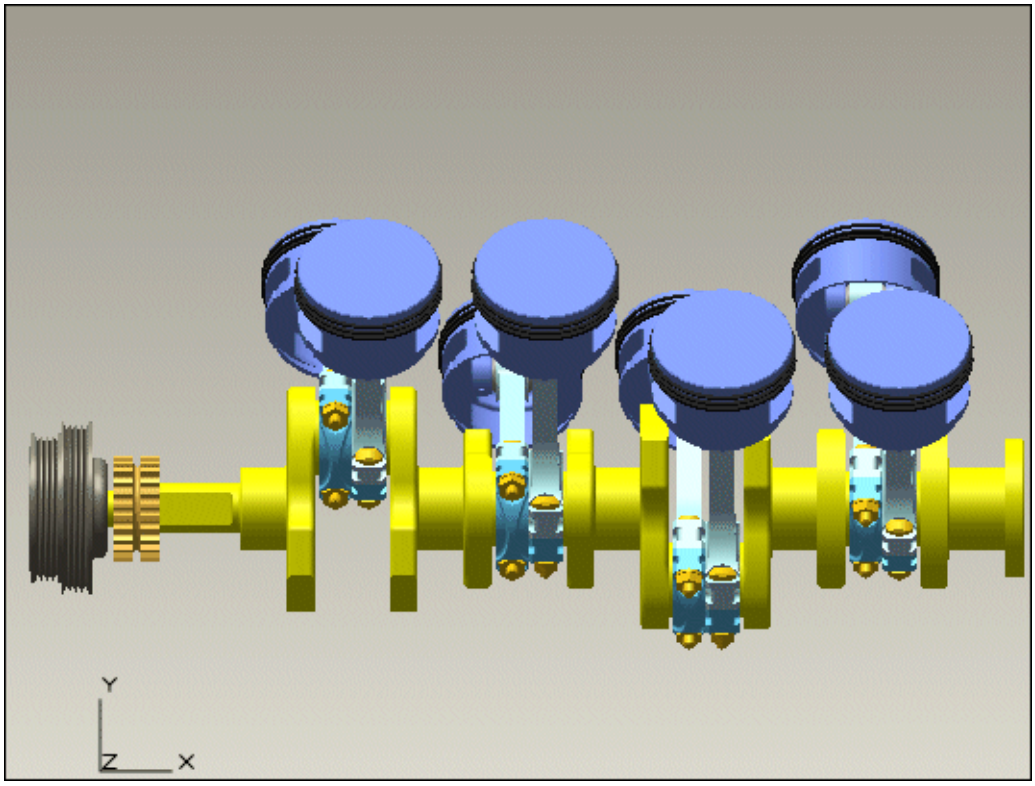
- **Top-level Cascading Menu**—Contains basic commands for using Creo View.
- **Ribbons**—Contain command groups.
- **Panels**—Display information about the product structure, such as files and annotations, as well as attributes.
- **Viewing Area**—The window where 3D models, drawings, and other files are displayed.
- **Status Bar**—Displays information about the current view, along with selection and units settings.

For more information on the Creo View user interface, refer to the online help available with your Creo View installation.

In the GUI mode, you can leverage the complete functionality that the UI offers and manipulate the loaded data beyond the control of your Creo View Toolkit application.

Non-GUI Mode

When the application starts Creo View in this mode, only the Viewing or Graphics area is displayed as shown in the following figure and the Creo View UI is not available.



In this mode, you can control the data displayed in the Viewing area only through the Creo View Toolkit method calls.

The methods that change the properties of the data in the Viewing area for a particular session or instance are available only in the non-GUI mode, for example, methods that manipulate:

- Background color of the Viewing area
- Properties of specific instances, such as color, transparency, and so on

Navigating Creo View

The Navigation tools that let you switch between different navigation modes are available in both GUI and non-GUI modes. You can use a combination of mouse and keyboard controls to switch between zoom, pan, and spin operations.

Note: For more information on the navigation controls, refer to the Creo View online help.

Support for Creo View Consumer

Creo View Consumer is a light-weight component version of Creo View Express. Creo View Consumer is used for interaction with 3D content and it does not include the user interface component of the Creo View client.

Creo View Consumer has been introduced, primarily, to work with Creo View Web Toolkit applications. Creo View Web Toolkit applications can launch Creo Consumer View, in the non-GUI mode, if the Creo View client is not installed on the system.

You can install Creo View Consumer using the `.exe` files, which are standalone installers. This installation allows you to use Creo View Consumer with Creo View Web Toolkit applications. You can also place the Creo View Consumer `.cab` file on the web server on which the Creo View Web Toolkit application is embedded.

To install Creo View Consumer, run `CreoView_Consumer_32.exe` or `CreoView_Consumer_32_64.exe` from the `<creoview_api_loadpoint>/redist/consumer` folder. The Creo View Consumer `consumer.cab` file is also located in the same folder.

To launch Creo View Consumer from a web application, specify `""` as the value of the input argument edition of the method **ProductView**.

Creo View Consumer supports the following file types:

- Creo View 3D files of type `.pvs`, `.pvz`, `.ol`, `.ed`, and `.edz` file types
- Image Files of type `.jpg`, `.tiff`, `.jpeg`, `.gif`, and `.bmp` file types
- Creo View drawing files of type .cgm and .hpgl file types

  Note: ECAD datasets (`.eda`), Creo Parametric data, and models in 2D are not supported in Creo View Consumer.

Creo View Web Toolkit

The Creo View Web Toolkit provides a Javascript programming interface to Creo View, allowing you to interact with Creo View inside your own Web pages.

Overview

The Creo View Web Toolkit is made up of a single class, `pvlaunch`. The method **ProductView** is the first API that is called when you load a Web page containing the embedded Creo View Web Toolkit APIs.

When you call the **ProductView** API, HTML code containing the correct script and object tags or embed tags with the appropriate parameters are inserted in the Web page. You can use the instance of the method **ProductView** to load different models or annotations. The input parameters of this method are:

- *edition*—Specifies if the Creo View client is started in the GUI or non-GUI mode through the Web application. The valid values are:
  - ""—Starts Creo View in the non-GUI mode.
  - "pview"—Starts Creo View in the GUI mode. This option displays the tabs that contain data related to the assembly or component. These tabs are:
    - **Files**—Displays the list of files referenced in the loaded assembly.
    - **Annotation Sets**—Displays the annotation sets and groups referenced in the loaded assembly
- *sourceUrl*—Specifies the name and path to the Creo View Structure (`.pvs`, `.ed`, `.eda`) or viewable files. You can also specify the name and path to Creo Parametric files. For more information refer to the section "Support for Creo Parametric Files."
- *markupurl*—Specifies the URL to the annotation `.etb` file. A `.etb` file lists the annotation sets for a `.pvs` or `.ed` file. It is modified every time an annotation set is created, renamed, or modified. Pass the URL to the `.etb` file when running Creo View against a Web server.
- *modifymarkupurl*—Specifies the URL where files will be posted when running Creo View against a Web server, for example, `http://localhost/web/file_upload.jsp?path=F:/tomcat/webapps/ROOT/web/demodata/Crank/&`. The URL can point to a `.jsp`, `.php`, or `.asp` file that will handle the requests to a Web server.
- *loadAnnotation*—Specifies the name of the annotation to be loaded on starting the application.

- *loadViewable*—Specify the filename and extension of the Creo View viewable file to be loaded on starting the application. All file types other than `.ol` files, that is, images, drawings, documents, and PDF files present in the `.pvs` or `.ed` file are called Creo View viewable files. Used to load the viewable source associated with a `.pvs` or `.ed` file.
- *uiconfigUrl*—Specify the location to an XML file used for configuring the right mouse button in GUI mode. For more information refer to the section, "Customizing the User Interface".
- *configOptions*—Currently not available.

  Note: You can load either an annotation or a viewable file on starting the application. If you specify a value for both the parameters, the annotation is loaded.

Support for Creo Parametric Files

Creo View Web Toolkit can load the following Creo Parametric format files:

Creo Parametric File	Type
Part	`.prt`
Drawing	`.drw`
Format	`.frm`
Layout	`.lay`
Diagram	`.dgm`
Report	`.rep`
Section	`.sec`

Note: Creo View Web Toolkit does not support Creo Parametric assembly (`.asm`) files.

To be able to open a Creo Parametric file in Creo View, set the following configuration options when you save them in Creo Parametric:

- save_model_display—Specify one of the following values:
  - `shaded_lod`
  - `shaded_low`
  - `shaded_high`

- save_drawing_picture_file —Specify as `both`.
- sketcher_save_preview_image —Specify as `yes`.

The Creo Parametric format (`.frm`), layout (`.lay`), diagram (`.dgm`), report (`.rep`), and section (`.sec`) files are treated similar to drawing files in Creo View. You can navigate through the sheets of the Creo Parametric drawing file (`.drw`) using the **Page Up** and **Page Down** keys on the keyboard.

Creating an Application Using Creo View Web Toolkit

To create applications using Creo View Web Toolkit:

1. Include the `pvlaunch.js` file in your custom Web page. This file is available at `<creo_view_api_loadpoint>/web/pview_html/pvlite`.
2. Call the function **SetPvBaseUrl("pvlite/")** to specify the installation location of the Creo View and Creo View Web Toolkit files.
3. Get the object returned by the method **ProductView**.
4. Call all the Creo View Web Toolkit methods on this object

Refer to the "`Installation test`" example available as a part of the chapter, "Sample Applications" to get an idea of how to create a simple application.

Example: Installation Test

```
<html>
<head>
    <title>Basic Installation Test</title>
</head>

<script type="text/javascript" src="pvlite/pvlaunch.js">
    SetPvBaseUrl("pvlite/");
</script>

    <body>
        <p align="center">
            <b>Basic Installation Test</b>
        </p>

        <table align="center" width="600" height="500">
            <tr valign="center" width="100%" height="100%">
```

```
                <td border="3" height="100%" align="justify">
                    <script type="text/javascript">
                        try {
                            ProductView("",
                             "../demodata/Crank/01-2_crankshaft_asm.pvs");
                            }
                            catch(e)
                            {
                            alert(e);
                            }
                    </script>
                </td>
            </tr>
        </table>
    </body>
</html>
```

Note: You cannot create more than one instance of the **ProductView** method within the graphics window called by the first instance. You can allocate space elsewhere in the Web page, for example, inside two different table cells.

Product Structure

The product structure displays a hierarchical view of the contents of the `.pvs` file. Each node in the product structure tree represents a component or subassembly in the structure.

This section describes the API support for loading a product structure or viewable files in Creo View. It also describes methods to browse through the structure tree.

Methods Introduced:

- **LoadModel**
- **ListInstances**
- **OnLoadComplete**
- **OnBeginInstance**
- **OnInstance**
- **OnEndInstance**
- **SetRenderMode**
- **ShowInstanceAndDescendants**
- **HideInstanceAndDescendants**

The method **LoadModel** loads the specified viewable (`.ol`) file or the specified structure (`.pvs` or `.ed`) file depending on the argument that you provide to the **ProductView** API. You can also use this method to load Creo Parametric files. The input arguments are:

- *sourceUrl*—Specifies the path or URL of the file to be loaded. The supported file types are:
  - Viewable (`.ol`)—Files that represent the 3D model graphics of the components of an assembly.
  - Structure (`.ed`)—Pre-ProductView 9.0 files that contain product structure, component position, orientation, and metadata (part- and assembly-level parameter) information.
  - Structure (`.pvs`)—ProductView 9.0 structure files that contain product structure, component position, and orientation information.
  - Structure Package (`.pvs/.edz`)—Compressed version of the structure files.
  - ECAD datasets (`.eda`)—Files that represent the ECAD datasets, schematic, or PCB.
  - Image or Drawing file —Image files of format (`.gif`, `.jpg`, `.gif`, `.bmp`) and drawing files of format (`.dwg`, `.dxf`, `.plt`).
  - Portable Document Format (PDF) files—Adobe Acrobat PDF files.
  - You can also specify the name and path to Creo Parametric files. For more information refer to the section “Support for Creo Parametric Files.”
- *markupUrl*—Specifies the URL to the annotation `.etb` file. The `.etb` files list the existing annotations in the `.pvs` and `.ed` files and are modified when new ones are created, renamed, and deleted. Pass the URL to the `.etb` file when running Creo View against a Web server.
- *modifymarkupurl*—Specifies the URL to the `.etb` file.
- *uiconfigUrl*—Specifies the location to an XML file used for configuring the right mouse button (RMB) menu in GUI mode. For more information refer to the section, “Customizing the User Interface”.

When the structure is loaded successfully, the callback of type **OnLoadComplete** is called. In this callback, call the method **ListInstances**.

The method **ListInstances** loads the callback methods **OnBeginInstance**, **OnInstance**, and **OnEndInstance** into your JavaScript application. The method of type **OnBeginInstance** is called first and only once. Next, the method of type **OnInstance** is called. This method provides the ability to navigate the tree structure to identify component instance names. It is called as many times as there are component instances. After all the instances have been passed in the callback, the method of type **OnEndInstance** is called.

The method **SetRenderMode** allows you to specify the rendering style for the assembly in 3D view. The input parameter renderMode specifies the rendering style. The valid values of this parameter are:

- “shaded”—Renders as shaded model. This is the default setting.
- “swe”—Renders as shaded model and also displays the edges of the model.
- “wireframe”—Renders as wireframe model.
- “hlr”—Renders the model as hidden lines removed.
- “mesh”—Renders the model as a mesh.

The method **ShowInstanceAndDescendants** allows you to select an instance and see the instance and its descendants, that is, the parent and child nodes in the product structure tree.

The method **HideInstanceAndDescendants** allows you to select an instance and hide the instance and its descendants, that is, the parent and the child nodes in the product structure tree.

Instance Operations

The methods described in this section enable instance operations in the session.

Methods Introduced:

- **HideInstance**
- **ShowInstance**
- **ShowAll**
- **IsolateSelected**
- **LoadInstance**
- **UnloadInstance**
- **GetInstanceLocation**
- **SetInstanceLocation**
- **ResetInstanceLocation**
- **RestoreAllLocations**
- **GetInstanceColor**
- **SetInstanceColor**
- **SetInstanceTransparency**
- **CalculateBoundingSphere**
- **CalculateBoundingBox**
- **CancelPendingDownloads**

After you load a `.pvs` structure using the method **LoadModel**, you can choose to hide or unload a specified instance.

The method **HideInstance** temporarily turns off visibility for the specified instance in the viewer. The method **ShowInstance** makes the specified hidden instance visible in the 3D view again. To show all the hidden instances in the 3D view, use the method **ShowAll**.

The method **IsolateSelected** temporarily turns off visibility for all instances that are not selected. It displays only the specified instance in the 3D view.

The method **UnLoadInstance** removes the specified instance from the 3D view. The method **LoadInstance** loads the specified instance into the 3D view.

The method **GetInstanceLocation** returns the location of the specified instance in X, Y, and Z coordinates. Use the method **SetInstanceLocation** to set the location of the specified component instance. The instance positions are relative to the world coordinate frame.

The method **ResetInstanceLocation** restores the specified instance to its original location. Use the method **RestoreAllLocations** to restore all instances in the product structure to their original locations.

The method **GetInstanceColor** returns the color used to display the component instance in the 3D view. Use the method **SetInstanceColor** to set the color for the specified instance to be displayed in the graphics window.

The method **SetInstanceTransparency** sets the transparency for the specified component in the 3D view. You can specify the instance as opaque to completely transparent.

The method **CalculateBoundingSphere** returns a bounding sphere value for a set of component instances.

The method **CalculateBoundingBox** returns a bounding box value for a set of component instances.

The method **CancelPendingDownloads** allows you to stop the download of `.ol` types of file.

View Operations

The methods described in this section enable you to customize the viewing area. The viewing area is where you open files for viewing. This view area can display 3D models, 2D drawings, images, and documents.

Methods Introduced:

- **SetBackgroundColor**
- **GetViewLocation**
- **SetViewLocation**
- **SetViewingMode**
- **GetOrthographicWidth**
- **SetOrthographicWidth**
- **GetPerspectiveHFOV**

- **SetPerspectiveHFOV**
- **ZoomToAll**
- **ZoomToSelected**
- **SetNavMethod**
- **GetSpinCenter**
- **SetSpinCenter**
- **MoveUp**
- **MoveDown**
- **MoveRight**
- **MoveLeft**
- **MoveForward**
- **MoveBackward**
- **RotateUp**
- **RotateDown**
- **RotateLeft**
- **RotateRight**
- **RotateClockwise**
- **RotateCounterClockwise**

Use the method **SetBackgroundColor** to set the background color for the graphics display. Specify the color in the format `#XXXXXX:#XXXXXX`, for example, `0x00000000:0x00ffffff`. The first value specifies the top background color of the view, while the second value specifies the bottom background color, with a gradient fill in between.

This method is available only in the non-GUI mode. In the GUI mode, you can change the background color using the available UI options.

The method **GetViewLocation** returns the location of the graphics view. It returns a 4x4 matrix of the location. This matrix specifies the X, Y, and Z position for translation and orientation of the view.

Use the method **SetViewLocation** to set the location of the view.

The method **SetViewingMode** sets the viewing mode for the current view. The valid viewing modes are:

- `perspective`—Allows you to perceive depth and distance, as objects would appear in reality.
- `orthographic`—Allows you to view objects without any perspective effects.

The method **GetOrthographicWidth** returns the width of the view in meters for the orthographic mode.

Use the method **SetOrthographicWidth** to set the orthographic width. You can specify this width in meters depending on the size of the model loaded.

The method **GetPerspectiveHFOV** returns the value of the Horizontal Field of View (HFOV) which represents the angle of the view area, as if seen through a camera. Use the method **SetPerspectiveHFOV** to set the value of the HFOV for the perspective viewing mode. You can calculate this value based on the size of the object and its distance from the viewpoint.

The methods **GetOrthographicWidth**, **SetOrthographicWidth**, **GetPerspectiveHFOV**, **SetPerspectiveHFOV**, **GetViewLocation**, and **SetViewLocation** enable you to manually set up a view and save its state which can be reapplied in future.

The method **ZoomToAll** adjusts the size of the view to display all components in the view.

The method **ZoomToSelected** magnifies the view to show the selected component in more detail.

The method **SetNavMethod** lets you select the Creo View navigation mode. The valid values are:

- `Inspect`—This navigation mode is the standard Creo View navigation mode, where you can spin or pan the view.
- `Explore`—This navigation mode maximizes the fly-through viewing functionality for 3D viewing in Creo View. Explore navigation mode uses a combination of mouse button presses, keyboard modifiers, and movement of the mouse in order to view the object from different depths and angles.

For more information regarding these navigation modes refer to the Creo View Online Help.

The method **GetSpinCenter** returns a POINT3D object that represents the center of rotation, around which to spin the assembly or part.

The method **SetSpinCenter** lets you specify a center of rotation, which acts as the origin (X, Y, and Z). For example,

```
function getSpinCenter()
{

    var spinCenter = myPvApi.GetSpinCenter();
    var xValue = spinCenter.x;
    var yValue = spinCenter.y;
    var zValue = spinCenter.z;
}

function setSpinCenter()

{
    var point3d = new POINT3D(-0.0862, -0.0338, 0.0205);
    myPvApi.SetSpinCenter(point3d);
}
```

The **Move*** and **Rotate*** methods enable you to move or rotate the 3D model in the specified direction and by the specified amount in the view.

View Orientations

A view orientation is the angle at which the structure is displayed in the graphics area in Creo View. The methods described in this section enable you to set and observe orientations that are added, deleted, or modified.

You can modify view orientations in Creo View MCAD, but you can only set existing view orientations in Creo View Lite.

Methods Introduced:

- **GetOrientations**
- **OnBeginOrientation**
- **OnAddOrientation**
- **OnRemoveOrientation**
- **OnEndOrientation**
- **SetOrientations**

The method **GetOrientations** returns a scriptable object and the methods **RegisterObserver** and **SetOrientation** act on this object.

Use the method **RegisterObserver** to receive callbacks on the list of available CAD and user defined orientations. Pass an instance of the javascript object that defines the call back functions as the input argument of this method.

The method of type **OnBeginOrientation** indicates the start of a list of specified orientations.

The method of type **OnAddOrientation** is called when a new view orientation is added to the list of global orientations.

The method of type **OnRemoveOrientation** is called when a view orientation has been deleted from the list of global orientations.

Use the method **SetOrientations** to set a specified orientation to the structure in 3D view. The valid values are:

- ORIENTATION_DEFAULT—Specifies the default orientations available with the Creo View installation.
- ORIENTATION_USER_DEFINED—Specifies a user-defined orientation created through the Creo View UI.
- ORIENTATION_CAD—Specifies the orientation defined in the Creo Parametric model.

For example,

```
/* Sets the selected orientation */
function SetOrientation(name, type)
{
   var my_array = name.split(" ");
   name = my_array[0];
   type = my_array[1];
   orientations.SetOrientation(name, type);
}
function OrientationObserver()
{
    this.OnAddOrientation=OnAddOrientation;
    this.OnRemoveOrientation=OnRemoveOrientation;
    this.OnBeginOrientation=OnBeginOrientation;
    this.OnEndOrientation=OnEndOrientation;
}

/* Gets the orientation */
function observeOrientations()
{
    orientations = pvApi.GetOrientations();
    orientations.RegisterObserver(new
                                    OrientationObserver());
}
```

Screen Capture

Methods Introduced:

- **CaptureScreenshot**
- **OnSaveScreenShot**
- **CopyToClipboard**

The method **CaptureScreenshot** captures a snapshot of the current screen and saves it as an image to a file on disk. You can specify the height and width of the image. The supported image formats are `.gif` and `.bmp`. The method of type **OnSaveScreenShot** is called when the image file is saved.

The method **CopyToClipboard** copies a snapshot of the current view to the clipboard.

Selection Operations

Methods Introduced:

- **SelectInstance**
- **OnBeginSelect**
- **OnSelectInstance**
- **DeselectInstance**
- **OnDeSelectInstance**
- **OnDeSelectAll**
- **OnEndSelect**

The method **SelectInstance** selects the specified instance. The callback of type **OnBeginSelect** indicates the start of the selection.

When the instance is selected, the callback of type **OnSelectInstance** is called. This method returns the ID of the selected instance.

The method **DeselectInstance** deselects the specified instance. The callback of type **OnDeSelectInstance** is called when the specified instance is cleared in the view. Similarly, the callback of type **OnDeSelectAll** is called when the selection of all the instances in the view is cleared.

The following example code demonstrates how to use these APIs. Refer to the "Selection" example in the chapter, "Sample Applications" for more information.

```
<script type="text/javascript">
    try
    {
          myPvApi = ProductView("",
                           "../demodata/Crank/01-2_crankshaft_asm.pvs");

          myPvApi.OnBeginSelect = pvBeginSelect;
          myPvApi.OnEndSelect = pvEndSelect;
          myPvApi.OnSelectInstance = pvSelectInstance;
          myPvApi.OnDeSelectInstance = pvDeselectInstance;
          myPvApi.OnDeSelectAll = pvDeselectAll;
    }
    catch(e)
    {
    }
</script>

<script type="text/javascript">
    function pvBeginSelect()
    {
    }

    function pvSelectInstance(instanceId)
    {
    }

    function pvDeselectInstance(instanceId)
    {
    }

    function pvEndSelect()
    {
    }

    function pvDeselectAll()
    {
    }
</script>
```

Accessing Combined View States

Combined views are used to switch between customized display states of the models. The methods described in this section enable you to access the view states of the models.

Methods Introduced:

- **ListViewStates**
- **OnBeginViewState**
- **OnAddViewState**
- **OnEndViewState**
- **SetViewState**

The method **ListViewStates** triggers the activex control to search through the entire product structure and return the known view states associated with the model using the callback functions.

The method of type **OnBeginViewState** is called first and indicates that a view state has been found.

The method of type **OnAddViewState** is called when a new view state is added. This method passes the name and type of the view state which is then used by the method **SetViewState** to apply that view state.

The method of type **OnEndViewState** is called when all the available view states have been listed.

Use the method **SetViewState** to apply that view state to the current active view.

The valid values for the view states are:

- `VIEW_STATE`—Specifies a regular view state.
- `EXPLODE_STATE`—Specifies an exploded view state.
- `ALTERNATE_REPRESENTATION`—Specifies an alternate representation.
- `VIEW_STATE_SECTION_CUT`—Specifies a section cut.

Annotations

The methods in this section provide the ability to view, create, rename, and delete annotations for Creo View structure files. You can create, add, or rename annotations in a view and save them as an Annotation Set, which is stored along with the assembly. You can annotate only `.ed` and `.pvs` files.

Note: Some of the methods described in this section are not supported in Creo View Express. Therefore, not all functionality supported by applications created using these methods will be available in the Creo View Express client.

Methods Introduced:

- **GetNumOfAnnotations**
- **LoadAnnotation**
- **GetAnnotationName**
- **CreateAnnotation**
- **AddAnnotation**
- **SaveAnnotation**
- **DeleteAnnotation**
- **RenameAnnotation**
- **OnAnnotationEvent**

The method **GetNumOfAnnotations** returns the number of annotation sets stored with the current product structure.

The method **LoadAnnotation** loads the specified annotation set in the graphics view. Specify the name of the annotation as the input parameter of this method.

The method **GetAnnotationName** returns the name of the specified annotation set.

The method **CreateAnnotation** creates a new annotation set for the Creo View structure file. When you create a new annotation set, you can enter information such as the name of the annotation set, author, telephone number, e-mail address, and any other relevant comments. This method is not supported in Creo View Express.

The method **AddAnnotation** enables you to add an annotation or measurement to an existing annotation set. The valid input values are:

- `rectangle`—Provides the option to draw a rectangular annotation in the annotation set. To create a rectangle, specify this option and drag the mouse pointer to define the boundary.
- `ellipse`—Provides the option to draw an elliptical annotation.
- `rectanglemarker`—Provides the option to draw a rectangular annotation with a fill color.
- `ellipsemarker`—Provides the option to draw an elliptical annotation with a fill color.
- `leaderline`—Provides the option to draw a leaderline in the current annotation set. To draw the leaderline, click where you want to start the leader line, and drag the pointer to draw the line, and click again at the end point.
- `summary`—Provides the option to display the dimension summary of the selected object in the view. To create a summary, select an object in the viewing area and specify this option.
- `distance`—Provides the option to measure the distance between two objects selected in a view. To measure the distance, select the start point for the distance measurement in the view and call this method with the distance option. You can then select the end point for the distance measurement in the viewing area.
- `diameter`—Provides the option to measure the diameter of the selected object in view.
- `angle`—Provides the option to measure the angle between two objects selected in the view.

The method **AddAnnotationNote** creates an annotation note at the location the user clicks in the viewing area after calling this function. Specify the annotation text, font size, font colour, and background color for the annotation. This method is not supported in Creo View Express.

The method **AddAnnotation** puts Creo View in a specific mode to add annotations. When Creo View detects that you have added one annotation, it quits this mode. This method is not supported in Creo View Express.

After you add the annotation to an annotation set, you can save the annotation set using the method **SaveAnnotation**. This method is not supported in Creo View Express.

The method **DeleteSelectedAnnotations** deletes a selected annotation from an annotation set. This method is not supported in Creo View Express.

The method **DeleteAnnotation** deletes the specified annotation set. This method is not supported in Creo View Express.

The method **RenameAnnotation** saves the specified annotation under a new name. This method is not supported in Creo View Express.

You can use the callback of type **OnAnnotationEvent** to indicate the success or failure of the create, delete, rename or save operation.

The following example code demonstrates the use of some of the methods described in this section. For more information refer to the "`Annotations`" example in the chapter, "Sample Applications".


```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
 "http://www.w3.org/TR/html4/strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Annotation example</title>
</head>

<script language="JAVASCRIPT" src="pvlite/pvlaunch.js"></script>
<script language="JAVASCRIPT" src="pvUtils.js"></script>

<script>
    var myPvApi;

    SetPvBaseUrl("pvlite/");

    /* Callback event */
    function pvAnnotationEvent(type, annoName, saved)
    {
        if (saved == true)
        {
            CheckState();
        }
    }

    /* Refreshes the annotation sets list */
    function CheckState()
    {
        if (_isIE)
        {
            var numElements =
                document.getElementById("annoSelection").children.length;
            for (i = 0; i < numElements; ++i)
```

```
        {
            document.getElementById("annoSelection").remove(0);
        }
    }
    else
    {
        var numElements =
          document.getElementById("annoSelection").childNodes.length;
        for (i = 0; i < numElements; ++i)
        {
            document.getElementById("annoSelection").remove(0);
        }
    }
    pvLoadComplete();
}

function modelSelected()
{
   myPvApi.LoadModel(document.getElementById("modelSelection").value,
                     "");
    var numElements =
             document.getElementById("annoSelection").children.length;
    for (i = 0; i < numElements; ++i)
    {
        document.getElementById("annoSelection").remove(0);
    }
}

/* Refreshes the annotation sets list */
function pvLoadComplete()
{
    var numAnno = myPvApi.GetNumOfAnnotations();
    document.getElementById("numAnnotations").value = numAnno;
    if (_isIE)
    {
        var oOption = document.createElement("OPTION");
        oOption.text = "-----";
        oOption.value = -1;
        document.getElementById("annoSelection").add(oOption);

        for (i = 0; i < numAnno; i++)
        {
            var oOption = document.createElement("OPTION");
            oOption.text = myPvApi.GetAnnotationName(i);
            oOption.value = i;
            document.getElementById("annoSelection").add(oOption);
        }
    }
    else
    {
        document.getElementById("annoSelection").options[0] =
```

```
                                new Option('-----', -1);

        for (i = 0; i < numAnno; i++)
        {
            var annoName = myPvApi.GetAnnotationName(i);
            document.getElementById("annoSelection").options[i + 1] =
                                    new Option(annoName, i);
        }
    }
}

/* Loads selected annotation set */
function annotationSelected() {
    var annoNum = document.getElementById("annoSelection").value;
    var annoName = myPvApi.GetAnnotationName(annoNum);
    if (annoNum != -1)
    {
        myPvApi.LoadAnnotation(annoName);
    }
    else
    {
        modelSelected();
    }
}

/* Deletes annotation set */
function deleteAnnotation()
{
    var anno_set_num = document.getElementById("annoSelection").value;
    if (anno_set_num == -1)
    {
        return false;
    }
    var numAnno = myPvApi.GetNumOfAnnotations();
    if (anno_set_num > numAnno || anno_set_num == "" ||
        anno_set_num < 0)
    {
        alert("Annotation set # is invalid");
        return false;
    }
   myPvApi.DeleteAnnotation(myPvApi.GetAnnotationName(anno_set_num));
}

/* Creates annotation set */
function createAnnotation()
{
    var numAnno = myPvApi.GetNumOfAnnotations();

    var name = document.getElementById("annotationName").value;
    var author = document.getElementById("annotationAuthor").value;
    var telephone = document.getElementById("annotationTelNo").value;
```

```
    var email = document.getElementById("annotationEmail").value;
    var comment = document.getElementById("annotationComment").value;

    myPvApi.CreateAnnotation(name, author, telephone, email, comment);

    clearField("annotationName");
    clearField("annotationAuthor");
    clearField("annotationTelNo");
    clearField("annotationEmail");
    clearField("annotationComment");
}

/* Saves annotation set */
function saveAnnotation()
{
    var annoNum = document.getElementById("annoSelection").value;
    if (annoNum != -1)
    {
        var anno_name = myPvApi.GetAnnotationName(annoNum);
        myPvApi.SaveAnnotation(anno_name);
    }
}

/* Rename annotation set */
function renameAnnotation()
{
    var anno_set_num = document.getElementById("annoSelection").value;
    var new_name = document.getElementById("rename0").value;
    var annoName = myPvApi.GetAnnotationName(anno_set_num);
    var numAnno = myPvApi.GetNumOfAnnotations();
    if (anno_set_num == -1)
    {
        return false;
    }

    if (anno_set_num > numAnno || anno_set_num == "" ||
        anno_set_num < 0)
    {
        alert("Annotation set # is invalid");
        return false;
    }

    if (_isIE)
    {
        var oOption = document.createElement("option");
        oOption.text = myPvApi.GetAnnotationName(anno_set_num);
        myPvApi.RenameAnnotation(oOption.text, new_name);
        document.getElementById("annoSelection").text = new_name;
    }
    else
    {
```

```
        myPvApi.RenameAnnotation(annoName, new_name);
        document.getElementById("annoSelection").text = new_name;
    }
    clearField("rename0");
}

function addAnnotation()
{
    var annotationType =
                        document.getElementById("addAnnotation").value;
    myPvApi.AddAnnotation(annotationType, "");

    document.getElementById("addAnnotation").value = "--";
}

function addAnnotationNote()
{
    var noteText = document.getElementById("noteText").value;
    var noteFontSize = document.getElementById("noteFontSize").value;
    var noteFontColor =
                        document.getElementById("noteFontColor").value;
    var noteBackgroundColor =
                document.getElementById("noteBackgroundColor").value;

    var status = isNum(noteFontSize);

    if (!status)
    {
        clearField("noteFontSize");
        return false;
    }

    status = isHex(noteFontColor);

    if (!status)
    {
        clearField("noteFontColor");
        return false;
    }

    status = isHex(noteBackgroundColor);

    if (!status)
    {
        clearField("noteBackgroundColor");
        return false;
    }

    myPvApi.AddAnnotationNote(noteText, noteFontSize, noteFontColor,
                              noteBackgroundColor);
    clearField("noteText");
```

```
        clearField("noteFontSize");
        clearField("noteFontColor");
        clearField("noteBackgroundColor");
    }

    function deleteSelectedAnnotation()
    {
        myPvApi.DeleteSelectedAnnotations();
    }
</script>

<body>
    <div align="left" style="top: 0; left: 0; border-width: 0px;
     border-style: none;">
        <table width="100%" bgcolor="#FFFFFF" bordercolorlight="#FFFFFF"
         bordercolordark="#FFFFFF" cellspacing="0" cellpadding="0"
         bordercolor="#FFFFFF">
            <tr height="58">
                <td width="100%" bgcolor="#89A2B3" bordercolor="#FFFFFF"
                 bordercolorlight="#FFFFFF" bordercolordark="#FFFFFF">
                    <p align="right">
                      <img src="bannrbmp.gif" alt="PTC Banner" width="493"
                                height="58">
                    </p>
                </td>
            </tr>
        </table>
    </div>

    <p align="center">
        <b>Annotation example</b>
    </p>

    <table width="845" >
        <tr width="550" valign="center">
            <td border="3" height="100%" align="justify" width="476">
                <table><tr><td height="700" width="476">
                <script>
                    try
                    {
                        myPvApi = ProductView("",
                        "../demodata/Crank/01-2_crankshaft_asm.pvs");
                        myPvApi.OnLoadComplete = pvLoadComplete;
                        myPvApi.OnAnnotationEvent = pvAnnotationEvent;
                    }
                    catch (e)
                    {
                        alert(e);
                    }
                </script>
                </td></tr></table>
```

```
    </td>

    <td border="3" align="justify" width="347" valign="top">
        <table><tr><td>
        <p align="left">
            Model 
           <select id="modelSelection" onchange="modelSelected()"
             name="gourl">
               <option value=
       "../demodata/Crank/01-2_crankshaft_asm.pvs">Crank</option>
              <option value="../demodata/Saw/saw.ed">Saw</option>
               <option value=
 "../demodata/Fishing_Reel/fishing_reel.pvs">Fishing Reel</option>
            </select>
             
        </p>

        <p align="left">
            # of Annotations Sets
            <input type="text" id="numAnnotations" readonly="yes"
                        name="T2" size="3"/>
             List
            <select id="annoSelection"
               onchange="annotationSelected()" name="Annotation">
            </select>
        </p>

        <hr>
        <p align="left">
            <b>Rename or Delete Annotation Set</b>
        </p>

        <p align="left">
            <input type="text" id="rename0" name="T2" size="16">
<input type="button" onclick="renameAnnotation()" value="Rename"/>
            <input type="button" onclick="deleteAnnotation()"
             value="Delete" name="B1"/>
        </p>

        <p>
            <b>Create Annotation Set</b>
        <p align="left">
            Name        
            <input type="text" id="annotationName" name="T2"
                        size="25"/>
         
        </p>

        <p align="left">
            Author       
            <input type="text" id="annotationAuthor" name="T4"
```

```
                    size="25"/>
                </p>

                <p align="left">
                    Telephone  
                    <input type="text" id="annotationTelNo" name="T6"
                    size="25"/>
                </p>

                <p align="left">

Email          
                    <input type="text" id="annotationEmail" name="T8"
                    size="25"/>
                </p>

                <p align="left">
                    Comments  
                    <textarea rows="1" id="annotationComment" name="S1"
                     cols="20" size="25"></textarea>
                </p>

                <p align="left">
                    <input type="button" onclick="createAnnotation()"
                     value="Create Annotation" name="B1">
                    <hr>
                    <p align="left">
                        <b>Add/Save/Delete Annotation</b>
                    </p>
                </p>

                <p align="left">
                    Add
                    <select id="addAnnotation" onchange="addAnnotation()"
                              name="gourl">
                       <option value='--'>--</option>
                       <option value="select">Normal mode</option>
                       <option value="rectangle">Rectangle</option>
                       <option value="ellipse">Ellipse</option>
                       <option value="rectanglemarker">Rectangle
                                                    Marker</option>
                       <option value="ellipsemarker">Ellipse
                                                   Marker</option>
                       <option value="leaderline">Leaderline</option>
                       <option value="summary">Measurement
                                                Summary</option>
                        <option value="distance">Distance
                                                 measurement</option>
                       <option value="diameter">Diamater</option>
                       <option value="angle">Angle</option>
                    </select>
```

```
                <input type="button" onclick="saveAnnotation()"
                        value="Save" name="B1"/>
                <input type="button"
                        onclick="deleteSelectedAnnotation()"
                        value="Delete" name="B1"/>
            </p>

            <p align="left">
                <b>Add Annotation Note</b>
            </p>

            <p align="left">
                Text
                <input type="text" id="noteText" name="T2" size="8"/>
                Font Size
                <input type="text" id="noteFontSize" name="T2"
                        size="4"/>
                Font Color
                <input type="text" id="noteFontColor" name="T2"
                        size="4"/>
            </p>

            <p align="left">
                 Background Color
                <input type="text" id="noteBackgroundColor" name="T2"
                        size="4"/>
                <input type="button" onclick="addAnnotationNote()"
                        value="Add Note" name="B1"/>
            </p>
            <p align="left">
                 
            </p>
             </td></tr></table>
        </td>
    </tr>
</table>
</body>
</html>
```

Editing Annotations From a Web Server

To access annotations from a Web server:

1. Setup the Web server to support a `.jsp`, `.php`, or `.asp` interface to handle uploading of annotations to the Web server.
2. Specify the URL to this interface as the value of the `modifyMarkupurl` parameter in the method **ProductView** in your application HTML page, as follows,

```
myPvApi =
ProductView("","../demodata/Crank/01-2_crankshaft_asm.pvs","","http://loc
alhost:8080/M-13/web/file_upload.jsp?path=D:/apache-tomcat-6.0.13/webapps
/ROOT/M-13/web/demodata/Crank/&","","","");
```

3. When you create, add, or delete the annotations using your JavaScript application, the changes to the annotations are saved on the Web server.

   Note: The server will need to be configured, so that it indicates to the web browser not to use the previously cached version of the `ast`, `gif`, and `etb` files, and possibly the geometry files (`.ol`). If you do not configure the server, then successfully saved annotations to the server would not be retrieved and thus, these annotations will not be displayed in the Creo View.

Sample JSP File

The following code for the `file_upload.jsp` file is available at `<creo_view_api_loadpoint>/web`. Use this file as an interface to the Web server.

```
<%@ page language='java' contentType="text/html; charset=UTF-8"%>
<%@ page import="java.io.FileOutputStream, java.util.*, java.io.File,
java.lang.Exception, java.io.PrintWriter" %>
<%!
    /**
        Utility to aid decoding the ProductView annotation files
    */
    int raiseToPower(int i, int j)
    {
      if ( j <= 0 )
      {
        return 0;
      }

      int ret = i;
      for ( int k = 1; k < j; k++)
      {
        ret = ret * i;
      }
      return ret;
    }

    /**
        Decode the ast, gif, etb files saved from ProductView
    */
    byte[] decodeString(byte[] inBytes)
    {
      boolean v = false;
```

```
int START_CHAR = 48;

int inLen = inBytes.length;

if (inLen <= 0 )
{
  return inBytes;
}

int outLen = (inLen / 4) * 3;
int remainder = inLen % 4;
if ( remainder > 1 )
{
  outLen = outLen + remainder - 1;
}

byte[] outBytes = new byte[outLen];

for ( int ic = 0; ic < outLen; ic++)
{
  outBytes[ic] = 0;
}

int j = 6;
int inCount = 0;

for ( int i = 0; i < outLen; i++)
{
  int oneChar = inBytes[inCount] - START_CHAR;

  int power = raiseToPower(2,j);
  int r = oneChar % power;

  int nextChar = inBytes[inCount + 1] - START_CHAR;

  if ( j == 2 )
  {
    r = r + nextChar * power;
    inCount = inCount + 1;
  }
  else
  {
    r = r + (nextChar / (raiseToPower(2,(j - 2)))) * power;
  }
  outBytes[i] = (byte)r;
  inCount = inCount + 1;
  if ( j == 2 )
  {
    j = 6;
  }
  else
```

```
        {
          j = j - 2;
        }
      }
      return outBytes;
    }
%>
<%
    System.out.println("file_upload.jsp called");

    if(request.getParameter("delete") != null)
    {
        String fileName = request.getParameter("file");
        String path     = request.getParameter("path");

        System.out.println("Request to delete annotation: " + path +
                           fileName);

        File deleteFile = new File(path + fileName);
        if(deleteFile.exists())
        {
            boolean status = deleteFile.delete();
            System.out.println("File delete");
        }
        else
        {
            System.out.println("Failed to delete file");
        }
    }
    else if(request.getParameter("save") != null)
    {
        String fileName = request.getParameter("file");
        String path = request.getParameter("path");

        System.out.println("Request to save annotation: " + path +
                           fileName);

        int byteCount = request.getContentLength();

        ServletInputStream is = request.getInputStream();
        byte[] bytes = new byte[byteCount];

        int c = is.readLine(bytes, 0, byteCount);

        File saveTo = new File(path + fileName);
        FileOutputStream outputStream = new FileOutputStream(saveTo);

        byte[] decodedData = decodeString(bytes);
        outputStream.write(decodedData);

        outputStream.close();
```

```
    }
    else if (request.getParameter("create") != null)
    {
        String fileName = request.getParameter("file");
        String path = request.getParameter("path");

        System.out.println("Request to create annotation: " + path +
                           fileName);

        int byteCount = request.getContentLength();

        ServletInputStream is = request.getInputStream();
        byte[] bytes = new byte[byteCount];

        int c = is.readLine(bytes, 0, byteCount);

        File saveTo = new File(path + fileName);
        FileOutputStream outputStream = new FileOutputStream(saveTo);

        byte[] decodedData = decodeString(bytes);
        outputStream.write(decodedData);
        outputStream.close();
    }

    PrintWriter printWriter = response.getWriter();
    printWriter.println("ok");
    printWriter.println("HTTP/1.1 200 OK");
    printWriter.println("Content-Type: text/plain");
%>
```

Animation Sequences

The methods described in this section allow you to start, stop, and play animation sequences. The animation methods work with animation sequences of:

- Annotation sets loaded using the method **LoadAnnotation**.
- Viewable source files loaded using the method **LoadViewable**.

Methods Introduced:

- **HasAnimation**
- **StartAnimation**
- **StopAnimation**
- **SetAnimationOffset**

The method **HasAnimation** specifies whether the active annotation and illustration view has an animation sequence.

The method **StartAnimation** specifies whether to play the animation from the beginning or to continue playing it from the point where it had stopped. This method uses a Boolean value to indicate the play options. If you set the input parameter *fromBeginning* as `True`, the animation is played from the beginning.

The method **StopAnimation** stops a playing animation sequence.

The method **SetAnimationOffset** allows you to move the starting point of an animation sequence to any position. You can specify the starting point offset value in percentage.

Manipulation Modes

Creo View provides the translation, rotation and spinning modes to manipulate models in the view. The methods described in this section provides access to these modes.

Method Introduced:

- **SetMode**

The method **SetMode** enables you to rotate, translate and spin the 3D shapeview in the viewing area. The input parameter *mode* has the following values:

- `translate`—Specifies the option to move selected parts in 3D space, so that parts in an assembly can be visually moved out of their regular position.
- `rotate`—Specifies the option to change the orientation of the selected parts.
- `spincenter`—Allows you to select a point on a part which will be used as the spin centre position when you spin the model.
- `select`—Specifies the option to quit the translation, rotation, or spinning mode.

Accessing Viewable Files

The methods in this section provide the ability to access the viewables in the product structure file.

Note: The viewable files can be directly opened without them having to be a part of a `.pvs` or `.ed` file. In such a scenario, these files should be treated as read-only type of files.

- **GetNumOfViewables**
- **LoadViewable**
- **GetViewableName**
- **GetCurrentSheet**
- **GetNumberOfSheets**
- **SetCurrentSheet**

The method **GetNumOfViewables** returns the number of viewables in the product structure file.

The method **LoadViewable** loads the specified viewable from the product structure. The value of the input parameter for this method ranges from 0 to the number returned by the method **GetNumOfViewables**.

The method **GetViewableName** returns the name of the specified viewable file.

The method **GetNumberOfSheets** returns the number of sheets in a multi-sheet drawing file.

The methods **GetCurrentSheet** and **SetCurrentSheet** enable you to navigate multi-sheet drawings programatically. The method **GetCurrentSheet** returns the sheet number of the sheet currently displayed in the view. The method **SetCurrentSheet** enables you to view a specified sheet. The value of the input parameter for this method ranges from 1 to the number returned by the method **GetNumberOfSheets**.

Sphere Operations

Methods Introduced:

- **CreateSphere**
- **UpdateSphere**
- **OnSphereUpdate**
- **DeleteSphere**

The method **CreateSphere** creates a sphere at a location specified by the values of the X, Y, and Z coordinates. You also need to specify the radius, color, transparency and a Boolean value that specifies whether the created sphere can be dragged to another location.

Use the method **UpdateSphere** to update the properties such as the X, Y, and Z coordinates, radius, color and transparency of a sphere. The callback of type **OnSphereUpdate** indicates that the sphere is updated.

Use the method **DeleteSphere** to delete a created sphere.

Bounding Box Operations

Methods Introduced:

- **CreateBoundingBox**
- **UpdateBoundingBox**
- **OnBoundingBoxUpdate**
- **DeleteBoundingBox**

The method **CreateBoundingBox** creates a bounding box based on the minimum and maximum values of the X, Y, and Z coordinates that you specify. You also need to specify the color, transparency and a Boolean value that specifies whether the created bounding box can be dragged to another location.

Use the method **UpdateBoundingBox** to update the properties such as the minimum and maximum values of the X, Y, and Z coordinates, color and transparency of a bounding box. The callback of type **OnBoundingBoxUpdate** indicates that the bounding box is updated.

Use the method **DeleteBoundingBox** to delete a created bounding box.

Accessing Properties

A property exists within a component and is used to store meta information. This can consist of text information or references to other files or locations that store information. Properties are grouped into any of several property groups. A property group is a classification of properties. A component may contain two or more of the same property as long as they exist in different property groups.

The methods described in this section enable you to access the properties or metadata associated with the selected part.

Methods Introduced:

- **ListPropertyGroups**
- **OnBeginGroupProperties**
- **OnPropertyGroup**
- **OnEndGroupProperties**
- **LoadPropertyGroup**
- **OnPropertyGroupLoaded**
- **GetPropertyValue**
- **FindInstancesWithProperty**
- **OnBeginFindInstance**
- **OnFindInstance**
- **OnEndFindInstance**

The method **ListPropertyGroups** lists all the property groups associated with the selected part. This method loads the callbacks **OnStartGroupProperties**, **OnPropertyGroup**, and **OnEndGroupProperties**.

The method of type **OnStartGroupProperties** is called first indicating that the property groups are being returned. The method of type **OnPropertyGroup** is called for each property group found passing the name of the group and its load status. The method of type **OnEndGroupProperties** is called when all the property groups have been listed.

The method **LoadPropertyGroup** loads the specified property group for the selected part. This method loads the callback **OnPropertyGroupLoaded** when the property group is loaded.

The method **GetPropertyValue** returns the value for the specified attribute.

The method **FindInstancesWithProperty** finds all the instances that match the specified group and property. The method of type **OnStartFindInstance** is called when the start of the instances have been found. **OnFoundInstance** is called by passing the ID of the instances matching the group and property. **OnEndFindInstance** is called when all the instance IDs have been returned.

Illustration List Items

Creo View allows you to view and mark up technical illustrations created with Creo Illustrate. You can create an item list from an illustration in Creo Illustrate and further export it to Creo View. The methods described in this section allow you to retrieve information about the items from the item list of an illustration.

Methods Introduced:

- **OnViewableLoaded**
- **GetNumberOfItems**
- **GetItemNumber**
- **GetItemNameTag**
- **GetItemQty**
- **GetItemFromCalloutId**
- **GetItemFromInstance**
- **SelectItemsListItem**
- **SelectCallout**

Use the callback method **OnViewableLoaded** to check if the illustration view is successfully loaded. After the view is successfully loaded, you can use the below described methods to retrieve information about the items from the item list.

The method **GetNumberOfItems** gets the number of items from the item list available in the active view.

Use the method **GetItemNumber** to get the item number from the item list. The item index number is passed as the input parameter.

The method **GetItemNameTag** returns the name of the item from the item list. The item index number is passed as the input parameter.

The method **GetItemQty** returns a number that indicates the number of times the item is present in the item list.

The method **GetItemFromCalloutId** returns the index of the item for the specified callout ID.

The method **GetItemFromInstance** returns the index of the item for the specified instance ID.

Use the method **SelectItemsListItem** to select the specified list item.

Use the method **SelectCallout** to select the callout associated with the specified list item.

Customizing the User Interface

You can customize the right mouse button (RMB) menu using an external XML file only in the GUI mode. To customize the RMB menu:

1. Create an XML file that contains the logic required to add a customization for your application. You can add as many customization options as you want to this file.
2. Pass the location of this `.xml` file as the input to the parameter *uiconfigUrl* of the method **ProductView**. This adds a new command to the RMB menu in the 3D Viewing area.

The following example shows a sample `.xml` file and the syntax for the method **ProductView**. For more information refer to the "`UI Configuration`" example in the chapter, "Sample Applications".

XML File

```
<?xml version="1.0" encoding="UTF-8"?>
<uiconfig>

    <menu ui="rmb" view="all" select="any">menu string
      <action target="browser">
      </action>
      <icon>icon_name.extension<icon>
    </menu>
</uiconfig >
```

The `.xml` file contains the following components:

- `uiconfig`—This is the root element.
- action—Defines an extension to the right mouse button menu. This element contains the following attributes:
  - target—Specifies the target for the action. The valid target values are `browser` or `this`. For example, if you specify the target="`browser`", the target could be a browser or a new window. If you specify target="`this`", the action will return information to the client application.

You can specify URL in the target file only if the target value is set as `browser`. Specify the full URL to the target file in the following format:

- `file:///local host:/`—Specifies the location of a file on disk.
- `http:// or https://`—Specifies the location of a file on a server.

  You can also specify a relative path. The relative path is the base URL on which Creo View is running. The base URL in a relative path has its protocol as `http://`, `https://` or `file://`.

The body of the action element contains the definition of the action. The format of the action element follows the World Wide Web Consortium's (W3C) Uniform Resource Identifier (URI) definition. For example, `protocol:data`, where `protocol` is:

`http:`—Defines a web-targeted URL. For example, `http://www.ptc.com`.

The syntax for URL must:

- Support the basic URL template features.
- Allow the client to substitute values for parameters. For example, consider a URL, `http://XXXXX<!param!>`, where `<! !>` encloses the substitution argument.

- `saveViewToAnnoSet`—Specifies that the current view must be saved as an annotation set, if the value is set to `True`. A new annotation set is created if the view is not an annotation set.
- `annoSetNameSeed`—Specifies a name for the seed annotation set. An annotation set with this name is automatically created when `saveViewToAnnoSet` is set to `True`.
- `annoSetNameSuffixUser`—Specify `True` to append the current user's name to the name of the automatically created annotation set.
- `annoSetNameSuffixDate`—Specify `True` to append the current date to the name of the automatically created annotation set.

Note: If `annoSetNameSuffixUser` is set to `True` then the current date is appended after the user name.

- copyViewToClipboard—Specify `True` to copy the current view to the clipboard. If the view state is an annotation set, then the link and image are both copied. If you paste the contents in Microsoft Excel, Microsoft PowerPoint, or Microsoft Word and click the image link Creo View is opened in the corresponding Windchill page and displays the annotation set as the first page.

- `icon`—Specifies an icon image. You must specify the icon name with its extension in the `<icon>` element. You need not specify the path of the icon image in the `<icon>` element. For the icon to be displayed:
  - Creo View must be running on Windchill.
  - The specified icons must be up on the Windchill server.

  You can upload the icons in Windchill in the same way as the `uiconfig.xml` file.

- `menu`—Specifies the string to be displayed in the menu item or the tooltip for dashboard icons.
  - `ui`—Specifies the location where the action will take place. The options are `rmb` or `dash`.
  - `view`—Specifies the objects on which the action can take place in the user interface. The options are `2d` (2D object), `3d` (3D object), `image`, `doc` , `all`, and `none`. You can specify one of the options or an OR list of these options. For example, `view="2d|3d"`.
  - `pane`—Specifies a set of panes where the action will take place in the user interface. The options are viewcontent, landmark, file, modelannotations, all, and none. You can specify one of the options or an OR list of these options. For example, pane="viewcontent|landmark". The default value is all. If this attribute is not set, then by default the customized menu is shown on all the panes.
  - `select`—Specifies the number of selected objects required to activate this action. The options are `0`, `1`, `any`, or `many`.

- `visible`—Specifies a set of filter conditions using the functions listed below. Each function returns a boolean value that can be used to specify the filter conditions. You can specify more than one function by using the AND operator (&&).

Function Name	Description
`getSelectedCount ">" or "==" or ">=" number`	Returns `True` if the number of selected parts is "greater than", or "equal to", or "greater than or equal to" the specified number.
`getPropertyExists ('property_name')`	• Returns `True` if the specified property exists and is available for selection. • If the name specified in the first parameter of the function is enclosed in the substitution parameters `<!and!>`, then the substitution argument in the URL of action is substituted with this selected property value.

Function Name	Description
`getAnnotationSetId ('oid')`	• This method is supported only when Creo View is running on Windchill. • Returns `True` when the current view state is set to annotation. This function also returns `True` if `saveViewToAnnoSet` is set to True in `<action>`, though the view state is yet to be saved to annotation. • You can define a string for the parameter `'oid'`. • Do not set the property name of the part as oid. • If a URL is specified in the `<action>` tag, then the annotation set id is substituted for `<!oid!>` in the URL of action.
`getSelectedPartProperty( 1, 'part_objectid','poid1')`	• Returns `True` when the first selected object contains a property with the name in the second parameter of the function. In the above example `part_objectid` is the first selected object. • The order of selection is random, so the selection does not necessarily mean the first selection made by the user. This function ensures that each selection made by the user is unique. • The property value specified in the second parameter of this function will be substituted for `<!poid!>` in the URL specified in the `<action>` tag.

Function Name	Description
`getSelectedPartProperty( 2, 'part_objectid','poid2')`	• "Returns True when the second selected object contains a property with the name in the second parameter in the function. In the above example part_objectid is the second selected object. • The order of selection is random, so the selection does not necessarily mean the second selection made by the user. This function ensures that each selection made by the user is unique.

Sample XML File

Example

The following example shows a sample .xml file and the syntax for the method ProductView. For more information refer to the "UI Configuration" example in the chapter, "Sample Applications".

```
<?xml version="1.0" encoding="UTF-8"?>

<uiconfig>

    <menu ui="rmb" view="all" visible="getSelectedCount()==1
&amp;&amp;getPropertyExists('DESCRIPTION') ">
        <label>API RMB Example</label>
        <action
target="browser">http://localhost/web/propertyValue.html?cost=&lt;!COST!&
gt;&amp;bankangle=&lt;!BANK_ANG!&gt;&amp;description=&lt;!DESCRIPTION!&gt
;</action>

    </menu>

</uiconfig>
```

The `.xml` file that you create should contain the following:

- Calls to the **getSelectedCount()** and **getPropertyExists()** methods. Use these methods exactly as shown in the sample code above. You can change only the value of the input parameter of the method **getPropertyExists()**.
- The full URL to the target file with the following format:
  - `file:///D:/`—Specifies the location of a file on disk.
  - `http:// or https://`—Specifies the location of a file on server.

Creo View Method Syntax

```
script type="text/javascript">
    try
    {
        myPvApi =
          ProductView("pview","../demodata/Crank/01-2_crankshaft_asm.pvs",
                      "","","","","rmbMenu.xml");

        myPvApi.OnLoadComplete = pvLoadComplete;
        myPvApi.OnBeginGroupProperties = pvBeginGroupProperties;
        myPvApi.OnPropertyGroup = pvPropertyGroup;
        myPvApi.OnEndGroupProperties = pvEndGroupProperties;
        myPvApi.OnPropertyGroupLoaded = pvPropertyGroupLoaded;
    }
    catch(e)
    {
    }
</script>
```

Localizing the XML File

The XML file can be localized in different languages using resource bundles.

In the example below, the custom menu name is set as "Windchill Part Details".

```
<uiconfig>
   <menu …>
      <label rb="com.ptc.wvs.server.ui.uiResource"
key="PVCFG_PART_DETAILS">Part</label>
      <action…….. </action>
   </menu>
</uiconfig>

<uiResource.java>
@RBEntry("Windchill Part Details")
@RBComment("Creo View right menu action to display part details page")
public static final String PVCFG_PART_DETAILS = "1401";
```

Problem Report Workflow

To launch an external Problem Report workflow, follow the steps described below:

1. Create the command **Create Problem Report** in the RMB menu by customizing the RMB menu using an external XML file. See the section Customizing the User Interface for more information.
2. Click **Create Problem Report** to launch the external Problem Report workflow. When you click the command, the following actions take place in Creo View:
    - The current viewstate is saved as an annotation set (if not already saved).
    - The information is packaged together to launch the Windchill Problem Report workflow. The information is passed from Creo View to Windchill by customizing the RMB menu using an external XML file.

The Problem Report workflow is a Windchill web based task. The command launches a browser page and puts the user into the Problem Report workflow. The Problem Report is pre-populated with the saved annotation set. At this point, Creo View completes the Problem Report workflow launch task, and returns back to the view that was being annotated. You can continue working on Project Report in Windchill.

3

Sample Applications

This section describes the sample applications installed with your Creo View Web Toolkit installation.

Installing Sample Applications

When you install Creo View Web Toolkit from the CD-ROM, the Creo View Toolkit loadpoint directory contains all the libraries example applications, and documentation specific to Creo View Web Toolkit. The following are the directories under `<creo_view_api_loadpoint>`:

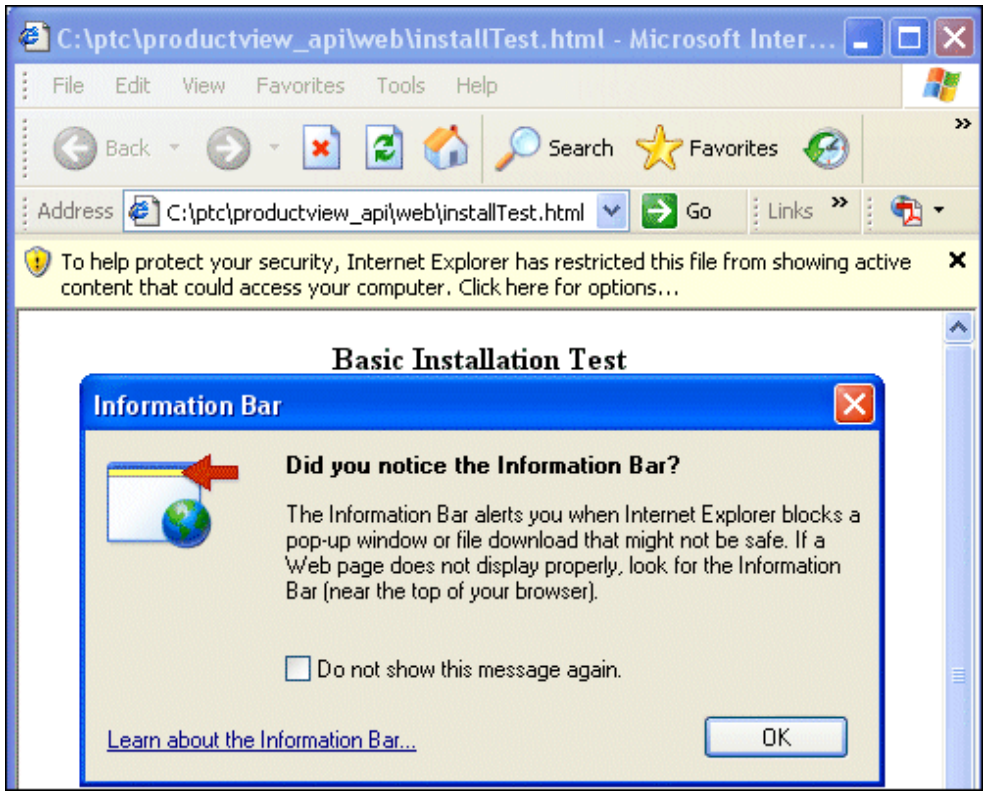
Directory	Description
`<creo_view_api_loadpoint>/web`	Contains the source files of the sample applications.
`<creo_view_api_loadpoint>/demodata`	Contains the data used by the sample applications.
`<creo_view_api_loadpoint>/documentation`	Contains the documentation specific to Creo View Web Toolkit.

To access the sample applications:

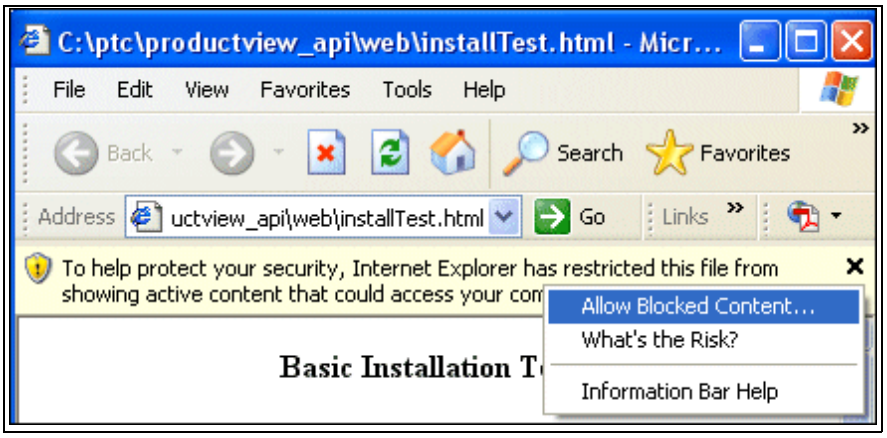
1. Open the file `sampleApplications.html` from `<creo_view_api_loadpoint>/web` to get a complete list of the sample applications.
2. Click on the link to the sample application that you want to access.

If you are using Internet Explorer to access the HTML files:

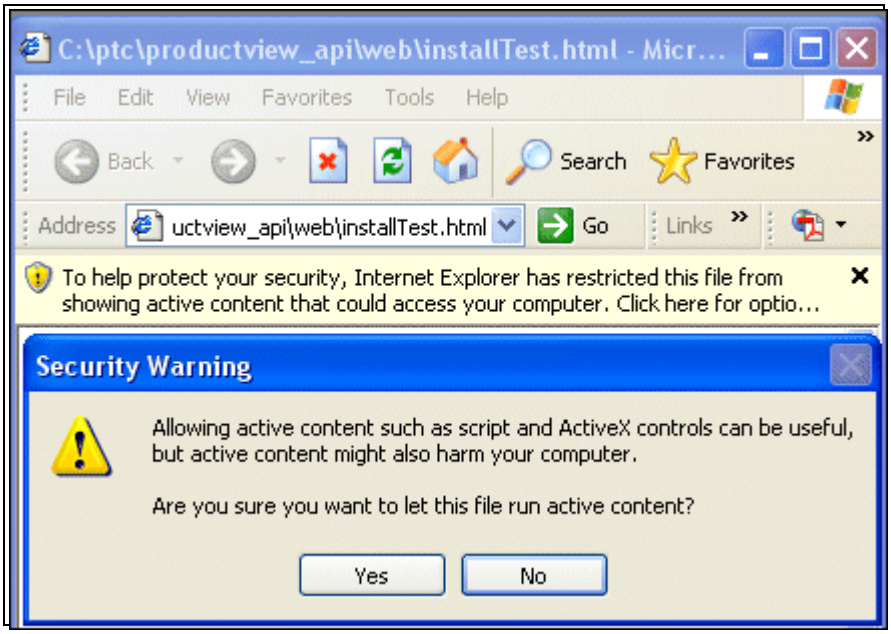
1. You may get the following warning depending on the security settings for your browser. **Click OK**.



2. Right-click the message at the top of the display window and click **Allow Blocked Content**.



3. The **Security Warning** dialog box opens. Click **Yes** and the sample application opens.



Details of Sample Applications

The following table provides more details on the sample applications.

Note: If you move or copy the sample applications to a web server or some other location, you must copy the corresponding `demodata` folder to that location too.

Sample Application	Filename	Description
Installation test	`installTest.html`	This example serves as a basic test to check if Creo View Web Toolkit is installed properly.
Load Model	`LoadModel.html`	This example loads a `.pvs` file. It also contains a drop-down list from which you can load different types of Creo View compatible files such as `.pvs`, `.ed`, `.ol`, and so on.
Selection	`selectionPvs.html`	This example allows the user to select or clear instances.
Generic APIs	`genericApi.html`	This example uses all the APIs that are generic in nature, that is, the APIs that can be used to change the way the user wants to look at the opened model.
Annotations	`annotation.html`	This example demonstrates the use of all APIs related to annotations. Not all functionality demonstrated by this example will be available if you use the Creo View Express client.
Animation Sequences	`animation.html`	This example demonstrates the use of all APIs related to animation sequences.
Bounding Box	`boundingbox.html`	This example demonstrates the use of APIs related to the bounding box.
Bounding Sphere	`boundingsphere.html`	This example demonstrates the use of APIs related to the bounding sphere.

Sample Application	Filename	Description
Instance	`instance.html`	This example uses all the APIs related to instance operations.
Viewables	`viewable.html`	This example lists all the viewables for a particular model in a drop-down list.

Loading models using the Creo View API	`productview1.html`
	`productview2.html`
	`productview3.html`
	`productview4.html`
	`productview5.html`
		This example shows the different ways to use the Creo View API to load either `.pvs`, `.ed`, or `.ol` files or annotations or viewables. It also shows how to use the Creo View API to load Creo Parametric files. You can open the ECAD datasets (PCB and Schematic).

UI Configuration	`uiconfig.html`	This example shows you how to add a command to the right mouse button menu.
Accessing Annotations from a Web server	`file_upload.jsp`	Use this file to create your own application to access annotations from a Web server. Refer to section, “Editing Annotations From a Web Server”, in the chapter, “Creo View Web Toolkit” for more details.
View States	`viewstate.html`	The example lists all the view states and the orientations of the model in the 3D view. It allows you to select and set any of the orientations and view states.

4

Summary of Technical Changes

This chapter contains a list of new and enhanced capabilities for Creo View 2.0 Web Toolkit.

New Methods

The following section describes the new methods for Creo View 2.0 Web Toolkit.

Component Operations

New Method	Description
OnViewableLoaded GetNumberOfItems GetItemNumber GetItemNameTag GetItemQty GetItemFromCalloutId GetItemFromInstance SelectItemsListItem SelectCallout	Retrieves information about illustration list items.
StartAnimation	Specifies whether the active annotation and illustration view has an animation sequence..

Index

V