PTC Creo® Parametric 3.0
J-Link User's Guide

Datecode M090

PTC Inc., 140 Kendrick Street, Needham, MA 02494 USA

Contents

About This Guide

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

Purpose

This manual describes how to use J-Link, a Java language toolkit for PTC Creo Parametric. J-Link makes possible the development of Java programs that access the internal components of a PTC Creo Parametric session, to customize PTC Creo Parametric models.

Note

J-Link is supported only with PTC Creo Parametric. It is not supported with the other PTC Creo applications.

Audience

This manual is intended for experienced PTC Creo Parametric users who are already familiar with Java or another object-oriented language.

Prerequisites

This manual assumes you have the following knowledge:

- PTC Creo Parametric
- The syntax and language structure of Java.

Documentation

The documentation for J-Link includes the following:

- *J-Link User's Guide*
- An online browser that describes the syntax of the J-Link methods and provides a link to the online version of this manual. The online version of the documentation is updated more frequently than the printed version. If there are any discrepancies, the online version is the correct one.

Conventions

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

`Convention`	**Description**
UPPERCASE	PTC Creo Parametric-type menu name (for example, PART).
Boldface	Windows-type menu name or menu or dialog box option (for example, **View**), or utility. **Boldface** font is also used for keywords, J-Link methods, names of dialog box buttons, and PTC Creo Parametric commands.
`Monospace (Courier)`	Code samples appear `in courier font like this`. Java aspects (methods, classes, data types, object names, and so on) also appear in Courier font.
Emphasis	Important information appears *in italics like this*. Italic font is also used for file names and uniform resource locators (URLs).
Choose	Highlight a menu option by placing the arrow cursor 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.
Element	An element describes redefinable characteristics of a feature in a model.
Mode	An environment in PTC Creo Parametric in which you can perform a group of closely related functions (Drawing, for example).
Model	An assembly, part, drawing, format, notebook, case study, sketch, and so on.
Option	An item in a menu or an entry in a configuration file or a setup file.
Solid	A part or an assembly.
`<creo_loadpoint>`	The location where the PTC Creo applications are installed, for example, `C:\Program Files\PTC\Creo 1.0`.
`<creo_jlink_loadpoint>`	The location where the J-Link application is installed, that is, `<creo_loadpoint>\<datecode>\Common Files\jlink`.

Note

- Important information that should not be overlooked appears in notes like this.
- All references to mouse clicks assume use of a right-handed mouse.

Software Product Concerns and Documentation Comments

For resources and services to help you with PTC software products, see the *PTC Customer Service Guide*. It includes instructions for using the World Wide Web or fax transmissions for customer support.

In regard to documentation, PTC welcomes your suggestions and comments. You can send feedback in the following ways:

- Send comments electronically to *MCAD-documentation@ptc.com*.
- Fill out and mail the PTC Documentation Survey in the customer service guide.

1

Setting Up J-Link

This chapter describes how to set up your environment so you can run J-Link.

Setting Up Your Machine

See Java Options and Debugging on page 421 for more information about supported Java Virtual Machines and how to setup PTC Creo Parametric.

Setting Up a Synchronous J-Link Program

A synchronous J-Link application is started and managed by PTC Creo Parametric. Control belongs to either PTC Creo Parametric or the application, but not both at the same time.

An asynchronous application is started independent of PTC Creo Parametric with the option to start or connect to PTC Creo Parametric processes. Refer to chapter Asynchronous Mode on page 353 for details on setting up an asynchronous application.

You can run synchronous J-Link programs as standalone applications or model-specific programs. Most of the required settings for these two programs are independent of the programs themselves. This enables you to convert an application program to a model program, or vice versa.

Standalone Applications

You can start the J-Link application independently at any time, regardless of which models are in session. A registry file contains key information regarding the execution of the program.

Using application programs you can make additions to the PTC Creo Parametric user interface, gather or change data associated with the models in session, or add session-level `ActionListener` routines. See the chapter Action Listeners on page 291 for more information on `ActionListeners`.

Registry File

A registry file contains PTC Creo Parametric-specific information about the standalone application you want to load.

The registry file called `protk.dat` is a simple text file, where each line consists of one predefined keyword followed by a value. The standard form of the `protk.dat` file is as follows:

```
name                java_demo
startup             java
java_app_class      MyJavaApp
java_app_start      start
java_app_stop       stop
allow_stop          true
delay_start         true
```

```
text_dir          <path to text directory used by
                   message and menu related commands>
end
```

The fields of the registry file are as follows:

- name—Assigns a unique name to this J-Link application. The name identifies the application when there is more than one in the protk.dat file. The maximum size of the name is 31 characters for the name, plus the end-of-string character.
- startup—Specifies the method to be used by to communicate with the application. For J-Link applications, set startup to java.
- java_app_class—Specifies the fully qualified package and name of a Java class. This class contains the J-Link application's start and stop methods (described below).
- java_app_classpath—An optional field to specify the full path to the J-Link application classes and archives (including the J-Link archive pfc.jar). Refer to the section CLASSPATH Variables on page 423 section for more information on the other available mechanisms to set the CLASSPATH. This field has a character limit of 2047 wide characters (wchar_t).
- java_app_start—Specifies the start method of your program. See the section Start and Stop Methods on page 21 for more information.
- java_app_stop—Specifies the stop method of your program. See the section Start and Stop Methods on page 21 for more information.
- allow_stop—Stops the application during the session if it is set to true. If this field is missing or set to false, you cannot stop the application, regardless of how it was started.
- delay_start—Enables you to choose when to start the J-Link application if it is set to true. PTC Creo Parametric does not start the J-Link application during startup. If this field is missing or is set to false, the J-Link application starts automatically.
- text_dir—Specifies the location of the text directory that contains the language-specific directories. The language-specific directories contain the message files, menu files, resource files and UI bitmaps in the language supported by the J-Link application. The files must be located under a directory called text or text/<language>, if localized messages are

used in the application. This field has a character limit of 2047 wide characters (`wchar_t`).

- `end`—Indicates the end of the description of the J-Link application. You can define multiple J-Link applications in the registry files. All these applications are started by PTC Creo Parametric.

Registering a J-Link Application

Registering a J-Link application means providing information about the files that form the J-Link application to PTC Creo Parametric. To do this, create a small text file, called the J-Link “registry file,” that PTC Creo Parametric will find and read.

PTC Creo Parametric searches for the registry file in the following order:

1. A file called `creotk.dat`, `protk.dat` or `prodev.dat` in the current directory
2. A file named in a `creotkdat`, `protkdat`, `prodevdat`, or `toolkit_registry_file` statement in the PTC Creo Parametric configuration file

> **Note**
>
> From Creo Parametric 1.0 onward, the file name `prodev.dat` has been replaced with `creotk.dat` or `protk.dat`. The configuration file option `prodevdat` can now be either `creotkdat`, or `protkdat`, or `toolkit_registry_file`.

3. A file called `creotk.dat`, `protk.dat`, or `prodev.dat` in the directory `<creo_loadpoint>\<datecode>\Common Files\<machine type>\text\<language>`
4. A file called `creotk.dat`, `protk.dat`, or `prodev.dat` in the directory `<creo_loadpoint>\<datecode>\Common Files\text`

In the last two options, the variables are as follows:

- `<creo_loadpoint>`—The PTC Creo Parametric loadpoint (not the J-Link loadpoint)
- `<machine type>`—The machine-specific subdirectory such as `i486_nt`
- `<language>`—The language of PTC Creo Parametric with which the J-Link application is used such as `usascii` (English), `german`, or `japanese`

If more than one registry file having the same filename exists in this search path, J-Link stops searching after finding the first instance of the file and starts all the J-Link applications specified in it. If more than one registry file having different filenames exists in this search path, PTC Creo Parametric stops searching after finding one instance of each of them and starts all the J-Link applications specified in them.

Option 1 is used normally during development, because the J-Link application is seen only if you start PTC Creo Parametric from the specific directory that contains `creotk.dat`, `protk.dat`, or `protk.dev`.

Option 2 or 4 is recommended when making an end-user installation, because it makes sure that the registry file is found irrespective of thedirectory used to start PTC Creo Parametric.

Option 3 enables you to have a different registry file for each platform, and for each PTC Creo Parametric language. This is more commonly used for J-Link applications that have a platform dependent setup.

Starting and Stopping a Standalone Application

If the `delay_start` field in the registry file is set to false, the J-Link application starts automatically when you start PTC Creo Parametric. Otherwise start the program by following these steps:

1. From the PTC Creo Parametric toolbar, select **Utilities ▸ Auxiliary Applications**.
2. Choose the name of the application.
3. Click **Start**.
    - **Start** - activates start method
    - **Stop** - activates stop method

If the `allow_stop` field in the registry file is set to true, you can click **Stop** in the **Auxiliary Applications** dialog box to stop the application. Click **Start** to restart it. If the `allow_stop` field is set to false, the program runs until the PTC Creo Parametric session ends.

Setting Up a Model Program

A model program is a J-Link program specific to a particular solid model, that is part or assembly. PTC Creo Parametric activates the start method for a program when it loads the part into memory and activates the stop method when it erases the part from memory.

Using model programs you can add programming logic to the interaction with a solid model. You can create a dialog box to drive the regeneration of a part or create model-specific utilities to generate reports or engineering information from a model. As Java programs are platform independent, the same model program runs on any Windows installation of PTC Creo Parametric.

To setupa a model program you need to:

- Associate and run a J-Link application with a model
- Create a JAR file for Model-Program Dependency

Associating and Running a J-Link Application with a Model

1. Load the solid model that you want to associate and run with the J-Link application.
2. From the **PART** or **ASSEMBLY** menu, select **Tools** ▸ **Program** ▸ **J-Link**.
3. If the application is stored in a Java archive (JAR) file, click **Add File** in the **Model Programs** dialog box and add the JAR file to the list of Java archive files. If the application is stored in a .class file proceed to the next step.
4. Click **Add** and enter the following information in the **Java Application Properties** dialog box:
   - Application Name—A unique name for this **J-Link** application. The maximum size of the name is 31 characters for the name, plus the end-of-string character.
   - Class Name—The Java class that contains the start and stop methods for the J-Link application. This class must reside in a JAR file you have added to the list or in a directory that is part of your CLASSPATH.
   - Start Method Name—The method in the Class Name class that will be called whenever the model is loaded into a session.
   - Stop Method Name—The method in the Class Name class that will be called whenever the model is erased.

The J-Link application immediately attempts to run. If it cannot start successfully an exception condition is listed in the **Status** column of the **Model Programs** dialog box.

JAR File Needed for Model-Program Dependency

Although individual class files are associated with a model, there is no dependency between the model and the program. Therefore, PTC Windchill server is not able to recognize the relationship between the class files and the model. To create a dependency, include all the class files and the source code in a Java archive file (jar file). JAR files are created through the command jar, which is a part of the standard Java Development Kit (JDK) package.

To create a JAR file use a command similar to the following command string:

```
jar cvf0 myjar.jar *.java *.class
```

Note

You must use the command-line switch 0 (zero) because JAVA cannot read classes from compressed `JAR` files.

You can add `JAR` files to, or remove them from, a model by using the buttons on the left side of the model program interface. All the `JAR` files for the model program must be placed in the PTC Creo Parametric search path.

Note

When naming a J-Link model program `JAR` file, you must use lower case.

Start and Stop Methods

All synchronous J-Link programs must have a static start and stop method regardless of whether they will run standalone or as model programs. You can give these methods any name you want because you identify them in the registry file or in the model program setup. PTC Creo Parametric automatically calls these methods upon starting or stopping a program. All methods that you want to call in a particular program must be called in the start and stop methods, or you must use the start method to register listeners to react to events in the PTC Creo Parametric interface.

For example:

```
public static void startMyProgram()
{
    runMyUtilities();
    configureMyModels();
    addMyUI();
}

public static void stop() {
    cleanupModels();
    outputToPrinterFiles();
}
```

J-Link start and stop methods must be public, static, return void and take no arguments. You can configure applications based on the PTC Creo Parametric version and build or custom command line arguments using methods described in the Session Objects on page 65 chapter.

2

Java Programming Considerations

This chapter contains a brief overview of the Java programming language. None of the information provided in this chapter is specific to J-Link.

Use of the JDK in J-Link Applications

Sun Microsystems provides a large number of objects and methods with the Java Development Kit (JDK). These objects and methods include the following:

- Utilities and methods for a large volume of common programming tasks—Java has APIs for manipulating data, creating vectors (expandable arrays), responding to events, creating queues, and creating hash tables.
- Methods for file manipulation—The `java.io` API contains objects that enable you to read and write data to and from files.
- Creation of Web-enabled applets—The `java.applet` API enables quick creation of a Java applet for use in an HTML page.
- Access to user interface components—The `java.awt` and "Swing" APIs allow creation of simple and complex user interfaces.
- Java Database Connectivity (JDBC)—JDBC provides interaction with database objects

For information on specific classes and methods in the Java API's, visit the Sun Web site at the following URL:

`http://java.sun.com/docs/index.html`

Examples in this guide usually do not use classes from the Java API beyond the ones found in the package `java.lang`. Most of the other packages can be used to improve a J-Link program, but they are not absolutely necessary to create the program.

Java Overview

Java is an object-oriented programming language that offers portability across multiple platforms.

Note

The Java Overview presented here describes technical information known to be true for Java version 1.1.5. Later Java versions may render some of this information incorrect or obsolete.

Java offers the following benefits:

- `Inheritance`—One of the fundamental concepts of an object-oriented language is `inheritance`. A subclass inherits variables and methods not only from the class above it but also from all of its ancestors. Consider the following code:

```
class A {
    public A() {
    // Constructor of class A
    }
}

class B extends A {
    public B() {
    //A is a superclass of B, B is a subclass of A.
    }
}
```

Java does not support multiple inheritance. But implementation of an interface is a way around this restriction. For example:

```
interface A {
    public void  doNothing();
    // Only the declaration of a method
}

// Constructor of B
class B implements A {
    // Implementation of this method
    public void doNothing() {
    }
}
```

- `Polymorphism`—You can substitute a derived class whenever its base class is required.
- `Method overriding and overloading`—You can redefine a superclass method with an exact signature and return type. In addition, you can define methods or constructors with different signatures.
- `Platform-independent interpreter`—Java is architecture-neutral. When you compile a Java program, the compiler translates your program into platform-independent instructions called Java bytecodes. You then use an interpreter to parse each Java bytecode and run it on the computer. Your Java program is compiled only once, but is interpreted each time you run it.

  Java bytecodes are like machine code instructions for the Java Virtual Machine (JVM).
- `High performance`—Java is a high-performance language that is dynamic —you can load Java classes into a running Java interpreter.
- `No pointers`—Java does not allow you to use pointers to directly reference memory locations. However, all object references are, in effect, pointers, because they refer to a location in memory that contains an object. Therefore, setting one object equal to another does not create a new version of the object. For example:

  ```
  String mystring = yourstring;
  ```

In this example, a new `String` object is not created.

- `Garbage collection`—Java does not require you to free memory after you are finished with it. The garbage collection routines within the JVM recognize data that is no longer used and automatically free the memory.
- `No preprocessors`—Java does not include a command preprocessor like C or C++. Therefore, you cannot declare global constants as you do in C. Instead, you can declare a field within an object to be static and final, which effectively declares that field to be constant.

Java Keywords

This section describes the Java keywords most commonly used when using J-Link.

The following keywords specify the accessibility to data:

- `public`—Accessible from the class, subclass, package, and world.
- `private`—Accessible only from the class.
- `protected`—Accessible from the class, subclass, and package.
- `package`—Accessible from the class and package. This is the default access.

The following keywords describe variables or methods:

- `static`—The method or variable is not attached to a particular object, but to an entire class.

  The advantage of a static method is that you do not need to define an instance of the class in order to use the method.
- `final`—Specifies that the class, method, or field will not be modified by another object.

  A static final declaration identifies a constant.
- `new`—Creates instances of various classes. The following statement shows an example of instantiation:

  ```
  String mystring = new String ("This is my string.");
  ```

  Except for single objects, you cannot use the `new` keyword to initialize J-Link objects. You can use `new` to construct objects that do not explicitly belong to J-Link (that is, Java API objects).
- `instanceof`—The Java `instanceof` operator is a way to determine whether a particular object can be correctly cast to a specified class. The instanceof operator produces a `Boolean` value that identifies whether the object is a member of that class. The typical use is as follows:

  ```
  if (<objectname> instanceof <classname>)
  ```

In J-Link this operator should be used to distinguish between various levels of inheritance.

Java Data Types

Java allows primitive types to be wrapped inside of classes. These wrappers are used to provide an object definition for every type of data within Java. In J-Link, certain methods take a wrapped object argument instead of a primitive type. You can convert one to the other by creating a new instance, as in the following example:

```
boolean yes_or_no = true;
Boolean yes_or_no_object = new Boolean (yes_or_no);
Integer seven = new Integer (7);
```

The following table lists the type wrappers provided by Java.

Data Type	Java Wrapper Class
int	Integer
float	Float
double	Double
char	Character
byte	Byte
boolean	Boolean
—	String

Note

Java represents character strings only as objects, not as arrays of characters. There is no corresponding primitive string type.

The following keywords specify the implementation types:

native—A method implemented in another language (usually C or C++) that can be called from Java

abstract—A method or class that has no implementation

Event Handling

Java implements listeners and adapters to notify you of certain events. There are three kinds of listeners:

- `ActionListener`—Waits for a specified action, such as clicking on a button in a dialog box
- `ItemListener`—Waits for a specified item, such as a checked check box
- `FocusListener`—Waits for a specified keyboard or mouse focus event on a component

Java exceptions enable you to test for and handle certain events. To create event handling, use the following keywords:

- `try`—Execute a control block using predeclared exception handlers.
- `catch`—Specify the exceptions to "catch" in a `try` block.
- `finally`—Specify a control block to be applied after a `try` block, regardless of whether an exception is handled by a `catch` clause within the `try` block.
- `throw`—Immediately send control to a handler for that specific exception.

Comments

Java provides three different types of comment characters: C++-style, C-style, and `javadoc`-style.

As in the C++ language, two double slashes (//) are used to specify a one-line comment. For example:

```
.
.
.
// This method retrieves the value of the dimension.
.
.
.

Java also supports C-style comment characters (/* */).
All the text within these characters is considered a comment. For example:
.
.
.
{
/* Open the file input.txt with read-only
   access. */

   inStream = new RandomAccessFile ("input.txt", "r");
.
.
.
```

Java also supports documentation comments for `javadoc`, a tool that automatically creates Web pages from your code. See the URL `http://java.sun.com/products/jdk/javadoc/index.html` for more information on `javadoc`.

Documentation comments are delimited by the characters `/**` and `*/`. For example:

```
/** The following code example shows how to create a
  * random-access file. The program reads a line from
  * one file (input.txt) and writes it to another file
  * (output.txt).
  */
.
.
.
```

3

Overview of J-Link

This chapter provides an overview of J-Link.

Class Types

J-Link is made up of a number classes in many packages. The following are the eight main classes.

- `Creo Parametric-Related Interfaces`—Contain unique methods and attributes that are directly related to the functions in PTC Creo Parametric. See the section PTC Creo Parametric-Related Interfaces on page 32 for additional information.
- `Compact Data Classes`—Classes containing data needed as arguments to some J-Link methods. See the section Compact Data Classes on page 34 for additional information.
- `Union Classes`—A class with a potential for multiple types of values. See the section Unions on page 35 for additional information.
- `Sequence Classes`—Expandable arrays of objects or primitive data types. See the section Sequences on page 35 for more information.
- `Array Classes`—Arrays that are limited to a certain size. See the section Arrays on page 37 for more information.
- `Enumeration Classes`—Defines enumerated types. See the section Enumeration Classes on page 38 for more information.
- `ActionListener Classes`—Enables you to specify programs that will run only if certain events in PTC Creo Parametric take place. See the section Action Listeners on page 40 for more information.
- `Utility Classes`—Contains static methods used to initialize certain J-Link objects. See the section Utilities on page 42 for more information.

Each class shares specific rules regarding initialization, attributes, methods, inheritance, or exceptions. The following seven sections describe these classes in detail.

PTC Creo Parametric-Related Interfaces

The PTC Creo Parametric-related interfaces contain methods that directly manipulate objects in PTC Creo Parametric. Examples of these objects include models, features, and parameters.

Initialization

You cannot construct one of these objects using the Java keyword `new`. Some objects that represent PTC Creo Parametric objects cannot be created directly but are returned by a `Get` or `Create` method.

For example, `pfcSession.Session.GetCurrentModel` returns a `Model` object set to the current model and `pfcModelItem.ParameterOwner.CreateParam` returns a newly created `Parameter` object for manipulation.

Attributes

Attributes within PTC Creo Parametric-related objects are not directly accessible, but can be accessed through `Get` and `Set` methods. These methods are of the following types:

```
Attribute name: int XYZ
Methods:    int GetXYZ();
            void SetXYZ (int i);
```

Some attributes that have been designated as read can only be accessed by the `Get` method.

Methods

You must start Methods from the object in question and you must first initialize that object. For example, the following calls are illegal:

```
Window window;

window.Activate(); // The window has not yet been
                      initialized.

Repaint();         // There is no invoking object.
```

The following calls are legal:

```
Session session = pfcGlobal.GetProESession();
Window window = session.GetCurrentWindow();
       // You have initialized the window object.
window.Activate()
window.Repaint()
```

Inheritance

All PTC Creo Parametric related objects are defined as interfaces so that they can inherit methods from other interfaces. To use these methods, call them directly (no casting is needed). For example:

```
public interface Feature
     extends jxobject,
     pfcModelItem.ParameterOwner,
     pfcObject.Parent,
     pfcObject.Object,
     pfcModelItem.ModelItem

Feature myfeature;           // Previously initialized

String name = myfeature.GetName();// GetName is in the
```

```
                                   // class ModelItem.
```

However, if you have a reverse situation, you need to explicitly cast the object. For example:

```
ModelItem item;  // You know this is a Feature -- perhaps
                 // you previously checked its type.
int number = ((Feature)item).GetNumber();
                 // GetNumber() is a Feature method.
```

Exceptions

Almost every J-Link method can throw an exception of type `com.ptc.cipjava.jxthrowable`. Surround each method you use with a `try-catch-finally` block to handle any exceptions that are generated. See the Exceptions section for more information.

Compact Data Classes

Compact data classes are data-only classes. They are used, as needed, for arguments and return values for some J-Link methods. They do not represent actual objects in PTC Creo Parametric.

Initialization

You can create instances of these classes using a static create method.

```
Example: pfcModel.BOMExportIntructions_Create()
```

This static method usually belongs to the utility class in the specific package that the compact data class belong to.

Attributes

Attributes within compact data related classes are not directly accessible, but can be accessed through `Get` and `Set` methods. These methods are of the following types:

```
Attribute name: int XYZ
Methods:    int GetXYZ();
            void SetXYZ (int i);
```

Methods

You must start Methods from the object in question and you must first initialize that object. For example, the following calls are illegal:

```
SelectionOptions options;

options.SetMaxNumSels(); // The object has not been
                      initialized.
SetOptionsKeywords(); // There is no invoking object
```

Inheritance

Compact objects can inherit methods from other compact interfaces. To use these methods, call them directly (no casting needed).

Exceptions

Almost every J-Link method can throw an exception of type `com.ptc.cipjava.jxthrowable`. Surround each method you use with a `try-catch-finally` block to handle any exceptions that are generated.

Unions

Unions are interface-like objects. Every union has a discriminator method with the pre-defined name `Getdiscr()`. This method returns a value identifying the type of data that the union objects holds. For each union member, a pair of (`Get`/`Set`) methods is used to access the different data types. It is illegal to call any `Get` method except the one that matches the value returned from `Getdiscr()`. However, any `Set` method can be called. This switches the discriminator to the new value.

The following is an example of a J-Link union:

```
class ParamValue
{
public:
    ParamValueType                 Getdiscr ();
    String                         GetStringValue ();
    void                           SetStringValue (String value);
    int                            GetIntValue ();
    void                           SetIntValue (int value);
    boolean                        GetBoolValue ();
    void                           SetBoolValue (boolean value);
    double                         GetDoubleValue ();
    void                           SetDoubleValue (double value);
    int                            GetNoteId ();
    void                           SetNoteId (int value);
};
```

Sequences

Sequences are expandable arrays of primitive data types or objects in J-Link. All sequence classes have the same methods for adding to and accessing the array. Sequence classes are identified by a plural name, or the suffix `seq`.

Initialization

You cannot construct one of these objects using the Java keyword `new`. Static create methods for each list type are available. For example, `pfcModel.Models.create()` returns an empty `Models` sequence object for you to fill in.

Attributes

The attributes within sequence objects must be accessed using methods.

Methods

Sequence objects always contain the same methods: `get`, `set`, `getarraysize`, `insert`, `insertseq`, `removerange`, and `create`. Methods must be invoked from an initialized object of the correct type, except for the static `create` method, which is invoked from the sequence class.

Inheritance

Sequence classes do not inherit from any other J-Link classes. Therefore, you cannot cast sequence objects to any other type of J-Link object, including other sequences. For example, if you have a list of model items that happen to be features, you cannot make the following call:

```
Features features = (Features) modelitems;
```

To construct this array of features, you must insert each member of the list separately while casting it to a `Feature`.

Exceptions

If you try to get or remove an object beyond the last object in the sequence, the exception `cipjava.XNoAttribute` is thrown.

Example Code: Sequence Class

The following example code shows the sequence class `com.ptc.pfc.pfcModel.Models`.

```
package com.ptc.pfc.pfcModel;

public class Models extends jxobject_i
{
    public int getarraysize() throws jxthrowable
    public Model get (int idx) throws jxthrowable
    public void                       set (
        int                            idx,
        Model                          value
    ) throws jxthrowable
```

```
    public void                         removerange
(
        int                              frominc,
        int                              toexcl
    ) throws jxthrowable

    public void                          insert
(
        int                               atidx,
        Model                             value
    ) throws jxthrowable

    public void                         insertseq
(
        int                                 atidx,
        pfcModel.Models                     value
    ) throws jxthrowable

    public static pfcModel.Models   create() throws jxthrowable
}
```

Arrays

Arrays are groups of primitive types or objects of a specified size. An array can be one or two dimensional. The following array classes are in the `pfcBase` package: `Matrix3D`, `Point2D`, `Point3D`, `Outline2D`, `Outline3D`, `UVVector`, `UVParams`, `Vector2D`, and `Vector3D`. See the online reference documentation to determine the exact size of these arrays.

Initialization

You cannot construct one of these objects using the Java keyword `new`. Static creation methods are available for each array type. For example, the method `pfcBase.Point2D.create` returns an empty `Point2D` array object for you to fill in.

Attributes

The attributes within array objects must be accessed using methods.

Methods

Array objects always contain the same methods: `get`, `set`, and `create`. Methods must be invoked from an initialized object of the correct type, except for the `create` method, which is invoked from the name of the array class.

Inheritance

Array classes do not inherit from any other J-Link classes.

Exceptions

If you try to access an object that is not within the size of the array, the exception `cipjava.XNoAttribute` is thrown.

Example Code - Array Class

The following example code shows the array class `com.ptc.pfc.pfcBase.Point2D`.

```
package com.ptc.pfc.pfcBase;

public class Point2D extends jxobject_i
{
    public double                   get (int idx0) throws jxthrowable

    public void                     set (
        int                             idx0,
        double                          value
    ) throws jxthrowable

    public static Point2D   create() throws jxthrowable
};
```

Enumeration Classes

In J-Link, enumeration classes are used in the same way that an `enum` is used in C or C++. An enumeration class defines a limited number of static final instances which correspond to the members of the enumeration. Each static final instance has a corresponding static final integer constant. In the `FeatureType` enumeration class the static instance `FEATTYPE_HOLE` has as it's integer equivalent `_FEATTYPE_HOLE` . Enumeration classes in J-Link generally have names of the form `XYZType` or `XYZStatus`.

Enumeration instances are passes whenever a method requires you to choose among multiple options. Use the integer constants where an `int` is required (such as cases in a `switch` statement).

Initialization

You cannot construct one of these objects. You simply use the name of the static instance or static integer constant.

Attributes

An enumeration class is made up of constant integer attributes and static instances of the enumerated class type. Related integers and instances have the same name, except the integer attribute begins with an underscore (_). The names of these attributes are all uppercase and describe what the attribute represents. For example:

- `PARAM_INTEGER`—An instance of the `ParamValueType` enumeration class that is used to indicate that a parameter stores an integer value. The corresponding integer is `_PARAM_INTEGER`.
- `ITEM_FEATURE`—An instance of the `ModelItemType` enumeration class that is used to indicate that a model item is a feature. The corresponding integer is_ `ITEM_FEATURE`.

An enumeration class always has an integer attribute named "`__Last`", which is one more than the highest acceptable numerical value for that enumeration class.

Methods

Enumeration classes have one method that you are likely to use:

- `getValue`—Returns the integer value of an enumeration instance.

Inheritance

Enumeration classes do not inherit from any other J-Link classes.

Exceptions

Enumeration classes do not throw exceptions.

Example Code: Enumeration Class

The following example code shows the enumeration class `com.ptc.pfc.pfcBase.Placement`.

```
package com.ptc.pfc.pfcBase;

public final class Placement
{
    public static final int _PLACE_INSIDE = 0;

    public static final Placement PLACE_INSIDE =
           new Placement (_PLACE_INSIDE);

    public static final int _PLACE_ON_BOUNDARY = 1;

    public static final Placement PLACE_ON_BOUNDARY =
           new Placement (_PLACE_ON_BOUNDARY);
```

```
    public static final int _PLACE_OUTSIDE = 2;

    public static final Placement PLACE_OUTSIDE =
           new Placement (_PLACE_OUTSIDE);

    public static final int __Last = 3;

    public static Placement FromInt (int value)

    public int getValue()
};
```

Action Listeners

Use `ActionListeners` in J-Link to assign programmed reactions to events that occur within PTC Creo Parametric. J-Link defines a set of action listener interfaces that can be implement enabling PTC Creo Parametric to call your J-Link application when specific events occur. These interfaces are designed to respond to events from action sources in PTC Creo Parametric. Examples of action sources include the session, user-interface commands, models, solids, parameters, and features.

Initialization

For each of its defined `ActionListener` interfaces, J-Link provides a corresponding default implementation class. For example, the `SolidActionListener` interface has a corresponding `DefaultSolidActionListener` implementation. All of the default action listener classes override every listener method with an empty method.

You must use the default implementation to construct applications. You cannot directly implement the `SolidActionListener` interface, as this interface will be missing the routing used internally by J-Link.

You implement an action listener class by inheriting the appropriate default class and overriding the methods that respond to specific events. For the other events, PTC Creo Parametric calls the empty methods inherited from the default class.

Construct your `ActionListener` classes using the Java keyword `new`. Then assign your `ActionListener` to an `ActionSource` using the `AddActionListener()` method of the action source.

Attributes

Action listeners do not have any accessible attributes.

Methods

You must override the methods you need in the default class to create an `ActionListener` object correctly. The methods you create can call other methods in the `ActionListener` class or in other classes.

Inheritance

All J-Link `ActionListener` objects inherit from the interface `pfcBase.ActionListener`.

Exceptions

Action listeners cause methods to be called outside of your application start and stop methods. Therefore, you must include exception-handling code inside the `ActionListener` implementation if you want to respond to exceptions. In some methods called before an event, propagating an exception out of your method will cancel the impending event.

Example Code: Listener Class

The following example code shows part of the `SolidActionListener` interface.

```
package com.ptc.pfc.pfcSolid;

public interface SolidActionListener extends
     jxobject,
     com.ptc.pfc.pfcBase.ActionListener

{
    void                            OnBeforeRegen
(
        com.ptc.pfc.pfcSolid.Solid      Sld,
        com.ptc.pfc.pfcFeature.Feature  /* optional */ StartFeature
    ) throws jxthrowable;

    void                            OnAfterRegen (
        com.ptc.pfc.pfcSolid.Solid      Sld,
        com.ptc.pfc.pfcFeature.Feature  /* optional */ StartFeature,
        boolean                         WasSuccessful
    ) throws jxthrowable;

    void                            OnBeforeUnitConvert (
        com.ptc.pfc.pfcSolid.Solid      Sld,
        boolean                         ConvertNumbers
    ) throws jxthrowable;

    void                            OnAfterUnitConvert (
        com.ptc.pfc.pfcSolid.Solid      Sld,
```

```
        boolean                         ConvertNumbers
    ) throws jxthrowable;
};
```

Utilities

Each package in J-Link has one class that contains special static methods used to create and access some of the other classes in the package. These utility classes have the same name as the package, such as `pfcModel.pfcModel`.

Initialization

Because the utility packages have only static methods, you do not need to initialize them. Simply access the methods through the name of the class, as follows:

```
ParamValue pv = pfcModelItem.CreateStringParamValue ("my_param");
```

Attributes

Utilities do not have any accessible attributes.

Methods

Utilities contain only static methods used for initializing certain J-Link objects.

Inheritance

Utilities do not inherit from any other J-Link classes.

Exceptions

Methods in utilities can throw `jxthrowable` type exceptions.

Sample Utility Class

The following code example shows the utility class `com.ptc.pfc.pfcGlobal.pfcGlobal`.

```
package com.ptc.pfc.pfcGlobal;

public class pfcGlobal
{
public static pfcSession.Session GetProESession()
    throws jxthrowable

public static stringseq GetProEArguments()
    throws jxthrowable
```

```
public static string GetProEVersion()
    throws jxthrowable

public static string GetProEBuildCode()
    throws jxthrowable
}
```

Creating Applications

The following sections describe how to create applications. The topics are as follows:

- Importing Packages on page 43
- Application Hierarchy on page 43
- Exception Handling on page 44

Importing Packages

To use pfc code in your application you must import the necessary packages. Import each class or package with a statement similar to the following:

For the Parameter class only:

```
import com.ptc.pfc.pfcModelItem.Parameter;
```

For the package pfcBase (all classes):

```
import com.ptc.pfc.pfcBase.*;
```

You might also need to import the methods in `com.ptc.cipjava`, which contains the underlying J-Link structure, including exceptions and certain sequences.. Use the following statement:

```
import com.ptc.cipjava.*;
```

Application Hierarchy

The rules of object orientation require a certain hierarchy of object creation when you start a J-Link application. The method invoked must be `pfcGlobal.GetProESession`, which returns a handle to the current session of PTC Creo Parametric.

The application must iterate down to the level of object you want to access. For example, to list all the datum axes contained in the hole features in all models in session, do the following:

1. Get a handle to the session:
2. Get the models that are active in the session:
3. Get the feature model items in each model:

4. Filter out the features of type hole:
5. Get the subitems in each feature that are axes:

Exception Handling

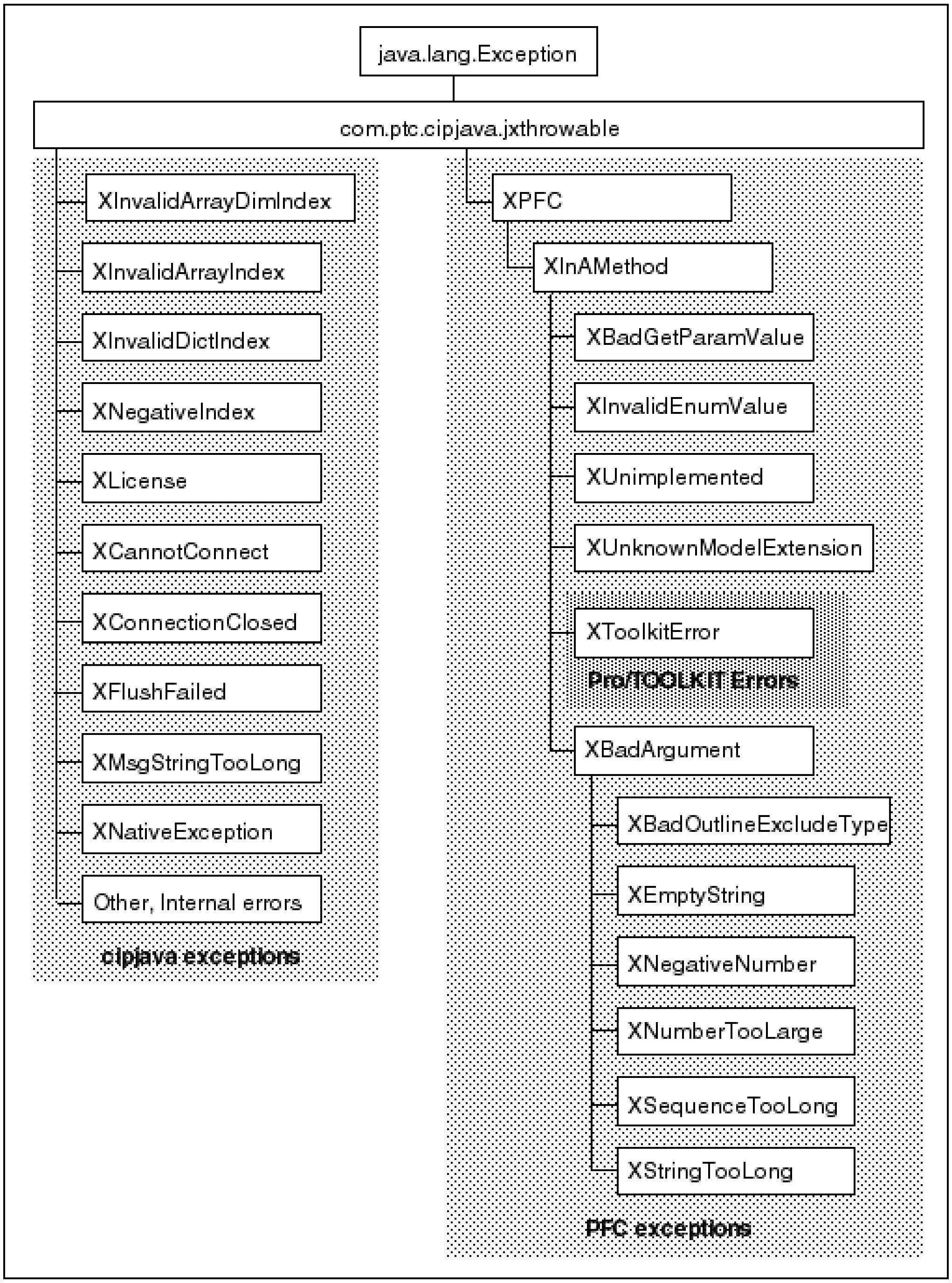
Nearly all J-Link methods are declared as throwing the `jxthrowable` exception, as shown here in the declaration of `Model.CreateLayer()`.

```
com.ptc.pfc.pfcLayer.Layer      CreateLayer (
    String                          Name
) throws jxthrowable;
```

In fact, the `jxthrowable` exception is never actually thrown. It is the parent class of all the exceptions that are thrown by J-Link methods and satisfies Java's requirement that a method's declaration include the exceptions that it throws.

The following figure organizes the J-Link exceptions into three categories to facilitate the discussion that follows.

J-Link Exception Classes



cipjava Exceptions

The `cipjava` exceptions are thrown by classes in the `com.ptc.cipjava` package. With the exception of the `intseq`, `realseq`, `boolseq`, and `stringseq` classes, these classes are only used internally.

The following table describes these exceptions.

Exception	Purpose
`XInvalidArrayDimIndex`, `XInvalidArrayIndex`, `XInvalidDictIndex`, `XNegativeIndex`	Illegal index value used when accessing a `cipjava` array, sequence, or dictionary. (The J-Link interface does not currently include any dictionary classes.)
`XLicense`	Licensing error (license does not exist, lost, server down, etc.)
`XMsgStringTooLong`	Communication synchronization problem between PTC Creo Parametric and J-Link application. Restart the J-Link application.
`XNativeException`	Unknown exception occurred in another language. This usually signals a serious error (such as division by zero or class not found) that is related to the J-Link application.
`XCannotConnect`, `XConnectionClosed`, `XFlushFailed`	Communication problems between PTC Creo Parametric and the J-Link application.
Other, internal errors	Internal assertions that should not append and which need not be caught individually.

PFC Exceptions

The PFC exceptions are thrown by the classes that make up J-Link's public interface. The following table describes these exceptions.

Exception	Purpose
`XBadExternalData`	An attempt to read contents of an external data object which has been terminated.
`XBadGetArgValue`	Indicates attempt to read the wrong type of data from the `ArgValue` union.
`XBadGetExternalData`	Indicates attempt to read the wrong type of data from the `ExternalData` union.
`XBadGetParamValue`	Indicates attempt to read the wrong type of data from the `ParamValue` union.
`XBadOutlineExcludeType`	Indicates an invalid type of item was passed to the outline calculation method.
`XCancelProEAction`	This exception type will not be thrown by J-Link methods, but you may instantiate and throw this from certain ActionListener methods to cancel the corresponding action in PTC Creo Parametric.
`XCannotAccess`	The contents of a J-Link object cannot be accessed in this situation.
`XEmptyString`	An empty string was passed to a method that does not accept this type of input.
`XInvalidEnumValue`	Indicates an invalid value for a specified enumeration class.

Exception	Purpose
XInvalidFileName	Indicates a file name passed to a method was incorrectly structured.
XInvalidFileType	Indicates a model descriptor contained an invalid file type for a requested operation.
XInvalidModelItem	Indicates that the item requested to be used is no longer usable (for example, it may have been deleted).
XInvalidSelection	Indicates that the Selection passed is invalid or is missing a needed piece of information. For example, its component path, drawing view, or parameters.
XJLinkApplicationException	Contains the details when an attempt to call code in an external J-Link application failed due to an exception.
XJLinkApplicationInactive	Unable to operate on the requested `JLinkApplication` object because it has been shut down.
XJLinkTaskExists	Indicates that a J-Link task with the given name already exists in session.
XJLinkTaskNotFound	Indicates that the J-Link task with the given name could not be found and run.
XModelNotInSession	Indicates that the model is no longer in session; it may have been erased or deleted.
XNegativeNumber	Numeric argument was negative.
XNumberTooLarge	Numeric argument was too large.
XProEWasNotConnected	The PTC Creo Parametric session is not available so the operation failed.
XSequenceTooLong	Sequence argument was too long.
XStringTooLong	String argument was too long.
XUnimplemented	Indicates unimplemented method.
XUnknownModelExtension	Indicates that a file extension does not match a known PTC Creo Parametric model type.

PTC Creo Parametric TOOLKIT Errors

The `XToolkitError` exception provides access to error codes from PTC Creo Parametric TOOLKIT functions that J-Link uses internally and to the names of the functions returning such errors. `XToolkitError` is the exception you are most likely to encounter because J-Link is built on top of PTC Creo Parametric TOOLKIT. The following table lists the integer values that can be returned by the `XToolkitError.GetErrorCode()` method and shows the corresponding PTC Creo Parametric TOOLKIT constant that indicates the cause of the error. Each specific `XToolkitError` exception is represented by an appropriately named child class, allowing you to catch specific exceptions you need to handle separately.

XToolkitError Child Class	PTC Creo Parametric TOOLKIT Error	#
XToolkitGeneralError	PRO_TK_GENERAL_ERROR	-1
XToolkitBadInputs	PRO_TK_BAD_INPUTS	-2
XToolkitUserAbort	PRO_TK_USER_ABORT	-3
XToolkitNotFound	PRO_TK_E_NOT_FOUND	-4
XToolkitFound	PRO_TK_E_FOUND	-5
XToolkitLineTooLong	PRO_TK_LINE_TOO_LONG	-6
XToolkitContinue	PRO_TK_CONTINUE	-7
XToolkitBadContext	PRO_TK_BAD_CONTEXT	-8
XToolkitNotImplemented	PRO_TK_NOT_IMPLEMENTED	-9
XToolkitOutOfMemory	PRO_TK_OUT_OF_MEMORY	-10
XToolkitCommError	PRO_TK_COMM_ERROR	-11
XToolkitNoChange	PRO_TK_NO_CHANGE	-12
XToolkitSuppressedParents	PRO_TK_SUPP_PARENTS	-13
XToolkitPickAbove	PRO_TK_PICK_ABOVE	-14
XToolkitInvalidDir	PRO_TK_INVALID_DIR	-15
XToolkitInvalidFile	PRO_TK_INVALID_FILE	-16
XToolkitCantWrite	PRO_TK_CANT_WRITE	-17
XToolkitInvalidType	PRO_TK_INVALID_TYPE	-18
XToolkitInvalidPtr	PRO_TK_INVALID_PTR	-19
XToolkitUnavailableSection	PRO_TK_UNAV_SEC	-20
XToolkitInvalidMatrix	PRO_TK_INVALID_MATRIX	-21
XToolkitInvalidName	PRO_TK_INVALID_NAME	-22
XToolkitNotExist	PRO_TK_NOT_EXIST	-23
XToolkitCantOpen	PRO_TK_CANT_OPEN	-24
XToolkitAbort	PRO_TK_ABORT	-25
XToolkitNotValid	PRO_TK_NOT_VALID	-26
XToolkitInvalidItem	PRO_TK_INVALID_ITEM	-27
XToolkitMsgNotFound	PRO_TK_MSG_NOT_FOUND	-28
XToolkitMsgNoTrans	PRO_TK_MSG_NO_TRANS	-29
XToolkitMsgFmtError	PRO_TK_MSG_FMT_ERROR	-30
XToolkitMsgUserQuit	PRO_TK_MSG_USER_QUIT	-31
XToolkitMsgTooLong	PRO_TK_MSG_TOO_LONG	-32
XToolkitCantAccess	PRO_TK_CANT_ACCESS	-33
XToolkitObsoleteFunc	PRO_TK_OBSOLETE_FUNC	-34
XToolkitNoCoordSystem	PRO_TK_NO_COORD_SYSTEM	-35
XToolkitAmbiguous	PRO_TK_E_AMBIGUOUS	-36
XToolkitDeadLock	PRO_TK_E_DEADLOCK	-37
XToolkitBusy	PRO_TK_E_BUSY	-38
XToolkitInUse	PRO_TK_E_IN_USE	-39
XToolkitNoLicense	PRO_TK_NO_LICENSE	-40
XToolkitBsplUnsuitableDegree	PRO_TK_BSPL_UNSUITABLE_DEGREE	-41

XToolkitError Child Class	PTC Creo Parametric TOOLKIT Error	#
XToolkitBsplNonStdEnd Knots	PRO_TK_BSPL_NON_STD_ END_ KNOTS	-42
XToolkitBsplMultiInner Knots	PRO_TK_BSPL_MULTI_ INNER_KNOTS	-43
XToolkitBadSrfCrv	PRO_TK_BAD_SRF_CRV	-44
XToolkitEmpty	PRO_TK_EMPTY	-45
XToolkitBadDimAttach	PRO_TK_BAD_DIM_ATTACH	-46
XToolkitNotDisplayed	PRO_TK_NOT_DISPLAYED	-47
XToolkitCantModify	PRO_TK_CANT_MODIFY	-48
XToolkitCheckoutCon flict	PRO_TK_CHECKOUT_ CONFLICT	-49
XToolkitCreateViewBad Sheet	PRO_TK_CRE_VIEW_BAD_ SHEET	-50
XToolkitCreateViewBadMo del	PRO_TK_CRE_VIEW_BAD_ MODEL	-51
XToolkitCreateViewBad Parent	PRO_TK_CRE_VIEW_BAD_ PARENT	-52
XToolkitCreateViewBad Type	PRO_TK_CRE_VIEW_BAD_ TYPE	-53
XToolkitCreateViewBadEx plode	PRO_TK_CRE_VIEW_BAD_ EXPLODE	-54
XToolkitUnattachedFeats	PRO_TK_UNATTACHED_FEATS	-55
XToolkitRegenerateAgain	PRO_TK_REGEN_AGAIN	-56
XToolkitDrawingCreateEr rors	PRO_TK_DWGCREATE_ERRORS	-57
XToolkitUnsupported	PRO_TK_UNSUPPORTED	-58
XToolkitNoPermission	PRO_TK_NO_PERMISSION	-59
XToolkitAuthentication Failure	PRO_TK_AUTHENTICATION_ FAILURE	-60
XToolkitAppNoLicense	PRO_TK_APP_NO_LICENSE	-92
XToolkitAppExcessCall backs	PRO_TK_APP_XS_CALLBACKS	-93
XToolkitAppStartup Failed	PRO_TK_APP_STARTUP_FAIL	-94
XToolkitAppInitializa tionFailed	PRO_TK_APP_INIT_FAIL	-95
XToolkitAppVersionMis match	PRO_TK_APP_VERSION_ MISMATCH	-96
XToolkitAppCommunica tionFailure	PRO_TK_APP_COMM_FAILURE	-97
XToolkitAppNewVersion	PRO_TK_APP_NEW_VERSION	-98

The exception `XProdevError` represents a general error that occurred while executing a Pro/DEVELOP function and is equivalent to an `XZToolkit` general Error. (PTC does not recommend the use of Pro/DEVELOP functions.)

The exception `XExternalDataError` and it's children are thrown from External Data methods. See the chapter on External Data for more information.

Approaches to J-Link Exception Handling

To deal with the exceptions generated by J-Link methods surround each method with a `try-catch-finally` block. For example:

```
try {
    JLinkObject.DoSomething()
}
catch (jxthrowable x) {
    // Respond to the exception.
}
```

Rather than catching the generic exception, you can set up your code to respond to specific exception types, using multiple `catch` blocks to respond to different situations, as follows:

```
try
{
    Object.DoSomething()
}
    catch (XToolkitError x)
    {
        // Respond based on the error code.
        x.GetErrorCode();
    }
    catch (XStringTooLong x)
    {
        // Respond to the exception.
    }
    catch (jxthrowable x)  // Do not forget to check for
                           // an unexpected error!
    {
        // Respond to the exception.
    }
```

If you do not want to surround every block of code with a `try` statement, you can declare your methods to throw the `jxthrowable` object. For example:

```
public class MyClass {

    public void MyMethod() throws jxthrowable
    {
         // Includes Pro/J-Link function calls
    }
}
```

However, you must surround any calls to `MyMethod()` with a `try` block.

4

J-Link Programming Considerations

This chapter contains information needed to develop an application using J-Link.

J-Link Thread Restrictions

When you run a synchronous J-Link program, you should configure your program so it does not interfere with the main thread of the PTC Creo Parametric program. Because the Java API allows you to run with multiple threads, you should be cautious of using certain Java routines in your program.

The most obvious restriction involves the use of Java language user interfaces. Any Java window that you create must be a dialog box that is blocking (or modal).

Creating a nonblocking frame causes a new thread to begin, which can have unexpected results when running with PTC Creo Parametric. For example, you can use the `javax.swing.JDialog` class and set `modal` true. You cannot use `javax.swing.JWindow`, however, because it starts a thread.

Optional Arguments to J-Link Methods

Many methods in J-Link are shown in the online documentation as having optional arguments.

For example, the `pfcModelItem.ModelItemOwner.ListItems` method takes an optional `Type` argument.

```
public interface ModelItemOwner extends
       jxobject

{
    com.ptc.pfc.pfcModelItem.ModelItems ListItems
(
         com.ptc.pfc.pfcModelItem.ModelItemType
             /* optional */ Type
    ) throws jxthrowable;
}
```

You can pass the Java keyword `null` in place of any such optional argument. In addition, a return value of `null` is possible for returns that are declared optional. The J-Link methods that take optional arguments provide default handling for null parameters as is described in the online documentation.

Note

- If a J-Link method takes an optional primitive type, such as an `int` or `boolean`, the argument will actually be a Java wrapper class, so that it may be set to `null`.
- You can only pass `null` in place of arguments that are shown in the documentation to be optional.

Optional Returns for J-Link Methods

Some methods in J-Link may have an optional return. Usually these correspond to lookup methods that may or may not find an object to return. For example, the `pfcSession.BaseSession.GetModel` method returns an optional model:

```
public interface Session
{
/*optional*/ com.ptc.pfc.pfcModel.ModelGetModel
(java.lang.String Name,
 com.ptc.pfc.pfcModel.ModelType Type);
}
```

J-Link might return `null` in certain cases where these methods are called. You must use appropriate error checking in your application code.

Parent-Child Relationships Between J-Link Objects

Some J-Link objects inherit from either the interface `com.ptc.pfc.pfcObject.Parent` or `com.ptc.pfc.pfcObject.Child`. These interfaces are used to maintain a relationship between the two objects. This has nothing to do with Java inheritance. In J-Link, the Child is owned by the Parent.

Methods Introduced:

- **pfcObject.Child.GetDBParent**

The method `GetDBParent` returns the owner of the child object. Because the object is returned as a `com.ptc.pfc.pfcObject.Parent` object, the application developer must down cast the return to the appropriate class. The following table lists parent/child relationships in J-Link.

Parent	Child
Session	Model
Session	Window
Model	ModelItem
Solid	Feature
Model	Parameter
Model	ExternalDataAccess
Display	DisplayList2D/3D
Part	Material
Model	View
Model2D	View2D
Solid	XSection
Session	Dll (PTC Creo Parametric TOOLKIT)
Session	Application (J-Link)

Run-Time Type Identification in J-Link

J-Link and the Java language provides several methods to identify the type of an object. Each has its adavantages and disadvantages.

Many J-Link classes provide read access to a type enum (for example, the `Feature` class has a `GetFeatType` method, returning a `FeatureType` enumeration value representing the type of the feature). Based upon the type, a user can downcast the `Feature` object to a particular subtype, such as `ComponentFeat`, which is an assembly component.

5

The J-Link Online Browser

This chapter describes how to use the online browser provided with J-Link.

Online Documentation J-Link APIWizard

J-Link provides an online browser called the J-Link APIWizard that displays detailed documentation. This browser displays information from the *J-Link User's Guide* and API specifications derived from J-Link header file data.

The J-Link APIWizard contains the following items:

- Definitions of J-Link packages and their hierarchical relationships
- Definitions of J-Link classes and interfaces
- Descriptions of J-Link methods
- Declarations of data types used by J-Link methods
- The *J-Link User's Guide*, which you can browse by topic or by class
- Code examples for J-Link methods (taken from the sample applications provided as part of the J-Link installation)

Read the Release Notes and README file for the most up-to-date information on documentation changes.

Note

The *J-Link User's Guide* is also available in PDF format. This file is located at:

<creo_jlink_loadpoint>\jlinkug.pdf

Accessing the APIWizard

You can access the APIWizard installed locally or from PTC.com. You can run the J-Link APIWizard in the following modes:

- Applet Based APIWizard—This APIWizard uses Java applets and requires that Java version lower than Java 7 Update 21 be installed on the computer.
- Non-Applet Based APIWizard—You do not need to install Java to run this APIWizard.
- PTC.com—If you do not want to install the APIWizard locally, you can access it from www.ptc.com/support/apiwizard.htm.

Installing the APIWizard

The J-Link APIWizarda are installed when you install J-Link from the product CD. The J-Link files are installed at <creo_jlink_loadpoint>\jlinkdoc.

To open the J-Link APIWizard top page point your browser to:
<creo_jlink_loadpoint>\jlinkdoc\index_jlink.html

The top page has links to:

- Applet based APIWizard
- Non-applet based APIWizard

Applet Based APIWizard

The applet based APIWizard is supported on Internet Explorer for Java plug-ins with versions lower than Java 7 Update 21.

To open the J-Link APIWizard, open Internet Explorer, and type:
<creo_jlink_loadpoint>\jlinkdoc\index.html

The APIWizard opens in the browser. See the section The Applet Based APIWizard Interface on page 59, for more information.

Note

If Java 7 Update 21 or higher is installed on your computer, then the locally installed applet based APIWizard will not open due to security issues. Use the non-applet based APIWizard in such cases.

Troubleshooting

When you open the applet based APIWizard in Internet Explorer and if, the browser detects the presence of the Java2 plug-in, though it is not be installed on the computer, the browser the browser opens index.html and waits for the Java applet to run. Since the Java2 plug-in is not installed, the Java applet is not available, and the browser remains indefinitely in this state.

The workaround for this is to enable Java console in the Internet Explorer options:

1. Launch the Internet Explorer browser.
2. Click **Tools**>**Internet Options**. The **Internet Options** dialog box opens.
3. Click the **Advanced** tab.
4. Under **Microsoft VM**, check **Java console enabled**.
5. Click **OK**.
6. Click **File**>**Close**.
7. Restart the Internet Explorer browser.

Automatic Index Tree Updating

A link in an APIWizard HTML file causes the tree in the **Selection** frame to update and scroll the tree reference that corresponds to the newly displayed page. This is automatic tree scrolling.

Non-Applet Based Version of the APIWizard

The non-applet based APIWizard supports Internet Explorer, Firefox, and Chromium browsers.

If you have installed Java 7 Update 21 or higher on your machine, the applet-based API Wizard will not open due to security issues. Use the non-applet based APIWizard in such cases.

Start the non-applet based version of the J-Link APIWizard by pointing your browser to:
<creo_jlink_loadpoint>\jlinkdoc\manual0\loadToolkitDoc.html

Non-Applet APIWizard Top Page

The top page of non-applet based APIWizard has links to the J-Link APIWizard and User's Guide. The APIWizard opens an HTML page that contains links to J-Link classes and related methods. The User's Guide opens an HTML page that displays the Table of Contents of the User's Guide, with links to the chapters, and sections under the chapters.

Click APIWizard to open the list of J-Link classes and related methods. Click a class or method name to read more about it.

You can search for specified information in the APIWizard. Use the search field at the top left pane to search for methods. You can search for information using the following criteria:

- Search by method names
- Search using wildcard character *, where * (asterisk) matches zero or more nonwhite space characters

The resulting method names are displayed in a drop down list with links to html pages.

You can also hover the mouse over  after you enter a string in the search field. The following search options are displayed:

- **Class/Methods**—Searches for classes and methods.
- **Global Methods**—Searches only for global methods.
- **Exceptions**—Searches only for exceptions.

Select an option and the search results are displayed based on this criteria.

User’s Guide

Click User’s Guide to access the *J-Link User’s Guide*.

APIWizard Available on PTC.com

The latest version of J-Link APIWizard is available at www.ptc.com/support/apiwizard.htm. This is an applet based APIWizard. However, the Java version on your system will not affect the display of the APIWizard.

The Applet Based APIWizard Interface

The APIWizard interface consists of two frames. The next sections describe how to display and use these frames in your Web browser.

Package/Class/Interface/Exception/Topic Selection Frame

This frame, located on the left of the screen, controls what is presented in the **Display** frame. Specify what data you want to view by choosing either **J-Link Packages**, **J-Link Classes**, **J-Link Interfaces**, **J-Link Exceptions**, or the **J-Link Wildfire User’s Guide**.

In **Packages** mode, this frame displays an alphabetical list of the J-Link packages. A package is a logical subdivision of functionality within J-Link; for example, the `pfcFamily` package contains classes related to family table operations. This frame can also display J-Link interfaces, classes, and methods as subnodes of the packages.

In **J-Link Classes** mode, this frame displays an alphabetical list of the J-Link Classes. It can also display J-Link methods as subnodes of the classes.

In **J-Link Interfaces** mode, this frame displays an alphabetical list of the J-Link interfaces. It can also display J-Link methods as subnodes of the interfaces.

In **J-Link Exceptions** mode, this frame displays an alphabetical list of named exceptions in the J-Link library. It can also display the methods for the exceptions as the subnodes.

In **J-Link Wildfire User’s Guide** mode, this frame displays the *J-Link User’s Guide* table of contents in a tree structure. All chapters are displayed as subnodes of the main *J-Link User’s Guide* node.

The **Package/Class/Interface/Topic** frame includes a **Find** button for data searches of the *J-Link User’s Guide* or of API specifications taken from header files. See the section The Applet Based APIWizard Search Feature (Find) on page 61 for more information on the **Find** feature.

Display Frame

This frame, located on the right of the screen, displays:

- J-Link package definitions and their hierarchial relationships.
- J-Link classes and interfaces definitions
- J-Link method descriptions
- User's Guide content
- Code examples for J-Link methods

Navigating the Package/Class/Interface/Exception/Topic Selection Tree

Access all J-Link APIWizard online documentation data for packages, classes, interfaces, methods, or the *J-Link User's Guide* from the **Package/Class/Interface** Selection frame. This frame displays a tree structure of the data. Expand and collapse the tree to navigate this data.

To expand the tree structure, first select **J-Link Packages**, **J-Link Classes**, **J-Link Interfaces**, **J-Link Exceptions**, or the **J-Link Wildfire User's Guide** at the top of the frame. The APIWizard displays the tree structure in a collapsed form. The switch icon to the far left of a node (i.e. a package, a class, a interface or chapter name) signifies that this node contains subnodes. If a node has no switch icon, it has no subnodes. Clicking the switch icon (or double-clicking on the node text) moves the switch to the down position. The APIWizard then expands the tree to display the subnodes. Select a node or subnode, and the APIWizard displays the online data in the **Display** frame.

Browsing the J-Link Packages

View the J-Link packages by choosing **J-Link Packages** at the top of the **Package/Classes/Interfaces/Topic** frame. In this mode, the APIWizard displays all the J-Link packages in the alphabetical order.

The **Display** frame for each J-Link package displays the information about all the classes and interfaces that belong to that package.

Click the switch icon next to the desired package name, or double-click the package name text to view the clasess or interfaces that belong to that package. You can also view the methods for each class or interface in the expanded tree by clicking the switch icon next to the class or interface name, or by double-clicking the name.

Browsing the J-Link User's Guide

View the *J-Link User's Guide* by clicking the **J-Link Wildfire User's Guide** at the top of the **Package/Class/Interface/Exception/Topic** frame. In this mode, the APIWizard displays the section headings of the User's Guide.

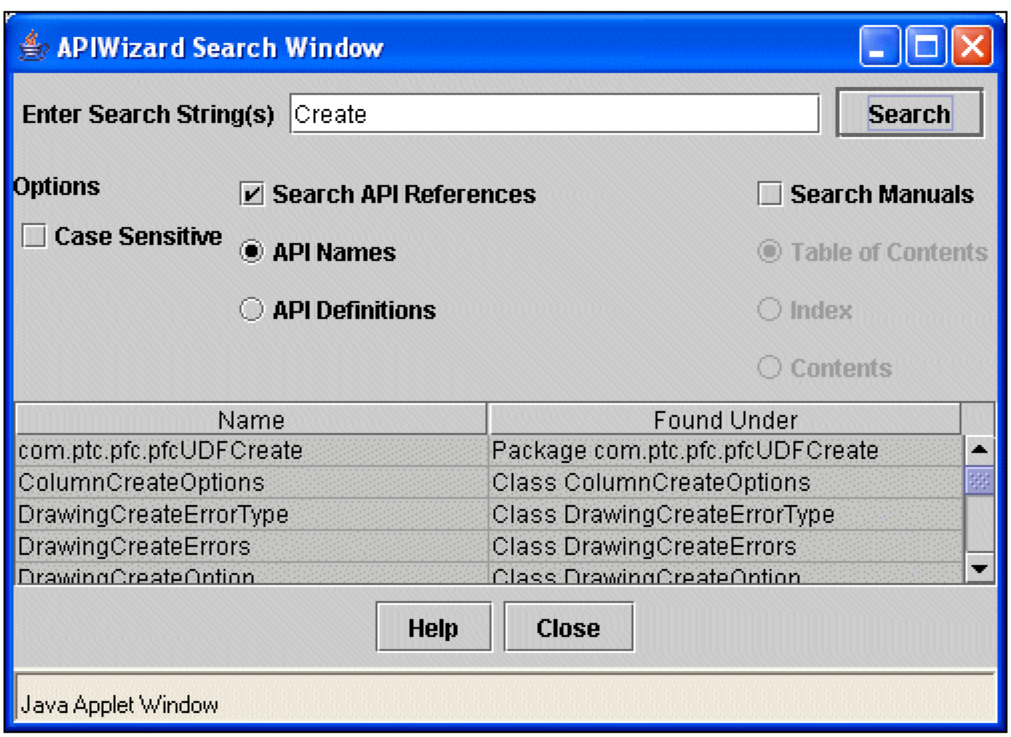
View a section by clicking the switch icon next to the desired section name or by double-clicking the section name. The APIWizard then displays a tree of subsections under the selected section. The text for the selected section and its subsections appear in the **Display** frame. Click the switch icon again (or double-click the node text) to collapse the subnodes listed and display only the main nodes.

The Applet Based APIWizard Search Feature (Find)

The APIWizard supports searches for specified strings against both the *J-Link User's Guide* and API definition files. Click the **Find** button on the **Package/Class/Interface/Exception/Topic** frame to display the APIWizard **Search** dialog.

Note

The APIWizard Search feature is slow when accessed through Internet Explorer's Default Virtual Machine. For better performance, access the APIWizard through Internet Explorer's Java2 plug-in.



The **Search** dialog box contains the following fields, buttons, and frames:

- **Enter Search String(s)**

  Enter the specific search string or strings in this field. By default, the browser performs a non-case-sensitive search.

- **Search/Stop**

  Select the **Search** button to begin a search. During a search, this button name changes to Stop. Select the **Stop** button to stop a search.

- **Help**

  Select this button for help about the APIWizard search feature. The APIWizard presents this help data in the **Display** frame.

- **Case Sensitive**

  Select this button to specify a case-sensitive search.

- **Search API References**

  Select this button to search for data on API methods. Select the **API Names** button to search for method names only. Select the **Definitions** button to search the API method names and definitions for specific strings.

- **Search Manuals**

  Select this button to search the *J-Link User's Guide* data. Select the **Table of Contents** button to search on TOC entries only. Select the **Index** button to search only the Index. Select the **Contents** button to search on all text in the *J-Link User's Guide*.

- **Name**

  This frame displays a list of strings found by the APIWizard search.

- **Found Under**

  This frame displays the location in the online help data where the APIWizard found the string.

Supported Search Types

The APIWizard Search supports the following:

- Case sensitive searches
- Search of API definitions, *J-Link User's Guide* data, or both
- Search of API data by API names only or by API names and definitions
- Search of *J-Link User's Guide* by Table of Contents only, by TOC and section titles, or on the User's Guide contents (the entire text).
- Wildcard searches—valid characters are:

- ○ `*` (asterisk) matches zero or more non-whitespace characters
- ○ `?` (question mark) matches one and only one non-whitespace character

To search for any string containing the characters Get, any number of other characters, and the characters Name

```
Get*Name
```

To search for any string containing the characters Get, one other character, and the characters Name

```
Get?Name
```

To search for any string containing the characters Get, one or more other characters, and the characters Name

```
Get?*Name
```

To search on the string Feature, followed by an `*`

```
Feature\*
```

To search on the string Feature, followed by a `?`

```
Feature\?
```

To search on the string Feature, followed by a `\`

```
Feature\\
```

- Search string containing white space— Search on strings that contain space characters (white space) by placing double- or single-quote characters around the string.

```
"family table"
'Model* methods'
```

- Search on multiple strings—Separate multiple search strings with white space (tabs or spaces). Note that the default logical relationship between multiple search strings is OR.

To return all strings matching GetName OR GetId, enter:

```
Get*Name Get*Id
```

Note

This search specification also returns strings that match both specified search targets.

For example:

```
FullName
```

returns `Model.GetName` and `ModelDescriptor.GetFullName`

If a string matches two or more search strings, the APIWizard displays only one result in the search table, for example:

```
Full* *Name
```

returns only one entry for each `FullName` property found.

Mix quoted and non-quoted strings as follows:

```
Get*Name "family table"
```

returns all instances of strings continaing Get and Name, or strings containing family table.

Performing an APIWizard Search

Follow these steps to search for information in the APIWizard online help data:

- Select the **Find** icon at the top of the **Package/Class/Interface/Exception/Topic** Selection frame.
- Specify the string or strings to be searched for in the **Enter Search String** field.
- Select **Case Sensitive** to specify a case-sensitive search. Note that the default search is non-case-sensitive.
- Select either or both of the **Search API References** and **Search User's Guide** buttons. Select the options under these buttons as desired.
- Select the **Search** button. The APIWizard turns this button red and is renames it **Stop** for the duration of the search.
- If the APIWizard finds the search string in the specified search area(s), it displays the string in the **Name** frame. In the **Where Found** frame, the APIWizard displays links to the online help data that contains the found string.
- During the search, or after the search ends, select an entry in the **Name** or **Where Found** frames to display the online help data for that string. The APIWizard first updates the **Package/Class/Interface/Exception/Topic** Selection frame tree, and then presents in the **Display** frame the online help data for the selected string.

6

Session Objects

This chapter describes how to program on the session level using J-Link .

Overview of Session Objects

The PTC Creo Parametric `Session` object (contained in the class `com.ptc.pfc.pfcSession.Session`) is the highest level object in J-Link. Any program that accesses data from PTC Creo Parametric must first get a handle to the `Session` object before accessing more specific data.

The `Session` object contains methods to perform the following operations:

- Accessing models and windows (described in the Models and Windows chapters).
- Working with the PTC Creo Parametric user interface.
- Allowing interactive selection of items within the session.
- Accessing global settings such as line styles, colors, and configuration options.

The following sections describe these operations in detail.

Directories

Methods Introduced:

- **pfcSession.BaseSession.GetCurrentDirectory**
- **pfcSession.BaseSession.ChangeDirectory**

The method `pfcSession.BaseSession.GetCurrentDirectory` returns the absolute path name for the current working directory of PTC Creo Parametric.

The method `pfcSession.BaseSession.ChangeDirectory` changes PTC Creo Parametric to another working directory.

File Handling

Methods Introduced:

- **pfcSession.BaseSession.ListFiles**
- **pfcSession.BaseSession.ListSubdirectories**

The method `pfcSession.BaseSession.ListFiles` returns a list of files in a directory, given the directory path. You can filter the list to include only files of a particular type, as specified by the file extension.

Starting with Pro/ENGINEER Wildfire 5.0 M040, the method `pfcSession.BaseSession.ListFiles` can also list instance objects when accessing PTC Windchill workspaces or folders. A PDM location (for workspace or commonspace) must be passed as the directory path. The following options have been added in the `FileListOpt` enumerated type:

- FILE_LIST_ALL—Lists all the files. It may also include multiple versions of the same file.
- FILE_LIST_LATEST—Lists only the latest version of each file.
- FILE_LIST_ALL_INST—Same as the FILE_LIST_ALL option. It returns instances only for PDM locations.
- FILE_LIST_LATEST_INST—Same as the FILE_LIST_LATEST option. It returns instances only for PDM locations.

The method pfcSession.BaseSession.ListSubdirectories returns the subdirectories in a given directory location.

Configuration Options

Methods Introduced:

- **pfcSession.BaseSession.GetConfigOptionValues**
- **pfcSession.BaseSession.SetConfigOption**
- **pfcSession.BaseSession.LoadConfigFile**

You can access configuration options programmatically using the methods described in this section.

Use the method pfcSession.BaseSession.GetConfigOptionValues to retrieve the value of a specified configuration file option. Pass the *Name* of the configuration file option as the input to this method. The method returns an array of values that the configuration file option is set to. It returns a single value if the configuration file option is not a multi-valued option. The method returns a null if the specified configuration file option does not exist.

The method pfcSession.BaseSession.SetConfigOption is used to set the value of a specified configuration file option. If the option is a multi-value option, it adds a new value to the array of values that already exist.

The method pfcSession.BaseSession.LoadConfigFile loads an entire configuration file into PTC Creo Parametric.

Macros

Method Introduced:

- **pfcSession.BaseSession.RunMacro**

The method pfcSession.BaseSession.RunMacro runs a macro string. A J-Link macro string is equivalent to a PTC Creo Parametric mapkey minus the key sequence and the mapkey name. To generate a macro string, create a mapkey in PTC Creo Parametric. Refer to the PTC Creo Parametric online help for more information about creating a mapkey.

Copy the `Value` of the generated mapkey Option from the **Tools ▸ Options** dialog box. An example `Value` is as follows:

```
$F2 @MAPKEY_LABELtest;
~ Activate `main_dlg_cur` `ProCmdModelNew.file`;
~ Activate `new` `OK`;
```

The key sequence is `$F2`. The mapkey name is `@MAPKEY_LABELtest`. The remainder of the string following the first semicolon is the macro string that should be passed to the method `pfcSession.BaseSession.RunMacro`.

In this case, it is as follows:

```
~ Activate `main_dlg_cur` `ProCmdModelNew.file`;
~ Activate `new` `OK`;
```

Note

Creating or editing the macro string manually is not supported as the mapkeys are not a supported scripting language. The syntax is not defined for users and is not guaranteed to remain constant across different datecodes of PTC Creo Parametric.

Macros are executed from synchronous mode only when control returns to PTC Creo Parametric from the J-Link program. Macros in synchronous mode are stored in reverse order (last in, first out).

Colors and Line Styles

Methods Introduced:

- **pfcSession.BaseSession.SetStdColorFromRGB**
- **pfcSession.BaseSession.GetRGBFromStdColor**
- **pfcSession.BaseSession.SetTextColor**
- **pfcSession.BaseSession.SetLineStyle**

These methods control the general display of a PTC Creo Parametric session.

Use the method `pfcSession.BaseSession.SetStdColorFromRGB` to customize any of the PTC Creo Parametric standard colors.

To change the color of any text in the window, use the method `pfcSession.BaseSession.SetTextColor`.

To change the appearance of nonsolid lines (for example, datums) use the method `pfcSession.BaseSession.SetLineStyle`.

Accessing the PTC Creo Parametric Interface

The `Session` object has methods that work with the PTC Creo Parametric interface. These methods provide access to the menu bar and message window. For more information on accessing menus, refer to the chapter Menus, Commands, and Pop-up Menus on page 97.

The Text Message File

A text message file is where you define strings that are displayed in the PTC Creo Parametric user interface. This includes the strings on the command buttons that you add to the PTC Creo Parametric number, the help string that displays when the user's cursor is positioned over such a command button, and text strings that you display in the Message Window. You have the option of including a translation for each string in the text message file.

Restrictions on the Text Message File

You must observe the following restrictions when you name your message file:

- The name of the file must be 30 characters or less, including the extension.
- The name of the file must contain lower case characters only.
- The file extension must be three characters.
- The version number must be in the range 1 to 9999.
- All message file names must be unique, and all message key strings must be unique across all applications that run with PTC Creo Parametric. Duplicate message file names or message key strings can cause PTC Creo Parametric to exhibit unexpected behavior. To avoid conflicts with the names of PTC Creo Parametric or foreign application message files or message key strings, PTC recommends that you choose a prefix unique to your application, and prepend that prefix to each message file name and each message key string corresponding to that application

Note

Message files are loaded into PTC Creo Parametric only once during a session. If you make a change to the message file while PTC Creo Parametric is running you must exit and restart PTC Creo Parametric before the change will take effect.

Contents of the Message File

The message file consists of groups of four lines, one group for each message you want to write. The four lines are as follows:

1. A string that acts as the identifier for the message. This keyword must be unique for all PTC Creo Parametric messages.
2. The string that will be substituted for the identifier.

   This string can include placeholders for run-time information stored in a `stringseq` object (shown in Writing Messages to the Message Window).
3. The translation of the message into another language (can be blank).
4. An intentionally blank line reserved for future extensions.

Writing a Message Using a Message Pop-up Dialog Box

Method Introduced:

- **pfcSession.Session.UIShowMessageDialog**

The method `pfcSession.Session.UIShowMessageDialog` displays the UI message dialog. The input arguments to the method are:

- *Message*—The message text to be displayed in the dialog.
- *Options*—An instance of the `pfcUI.MessageDialogOptions` containing other options for the resulting displayed message. If this is not supplied, the dialog will show a default message dialog with an **Info** classification and an **OK** button. If this is not to be null, create an instance of this options type with `pfcUI.pfcUI.MessageDialogOptions_Create()`. You can set the following options:
  - `Buttons`—Specifies an array of buttons to include in the dialog. If not supplied, the dialog will include only the **OK** button. Use the method `pfcUI.MessageDialogOptions.SetButtons` to set this option.
  - `DefaultButton`—Specifies the identifier of the default button for the dialog box. This must match one of the available buttons. Use the method `pfcUI.MessageDialogOptions.SetDefaultButton` to set this option.
  - `DialogLabel`—The text to display as the title of the dialog box. If not supplied, the label will be the english string `Info`. Use the method `pfcUI.MessageDialogOptions.SetDialogLabel` to set this option.
  - `MessageDialogType`—The type of icon to be displayed with the dialog box (**Info**, **Prompt**, **Warning**, or **Error**). If not supplied, an **Info** icon is

used. Use the method `pfcUI.MessageDialogOptions.SetMessageDialogType` to set this option.

Accessing the Message Window

The following sections describe how to access the message window using J-Link. The topics are as follows:

- Writing Messages to the Message Window on page 71
- Writing Messages to an Internal Buffer on page 71

Writing Messages to the Message Window

Methods Introduced:

- **pfcSession.Session.UIDisplayMessage**
- **pfcSession.Session.UIDisplayLocalizedMessage**
- **pfcSession.Session.UIClearMessage**

These methods enable you to display program information on the screen.

The input arguments to the methods `pfcSession.Session.UIDisplayMessage` and `pfcSession.Session.UIDisplayLocalizedMessage` include the names of the message file, a message identifier, and (optionally) a `stringseq` object that contains upto 10 pieces of run-time information. For `pfcSession.Session.UIDisplayMessage`, the strings in the `stringseq` are identified as `%0s`, `%1s`, ... `%9s` based on their location in the sequence. For `pfcSession.Session.UIDisplayLocalizedMessage`, the strings in the `stringseq` are identified as `%0w`, `%1w`, ... `%9w` based on their location in the sequence. To include other types of run-time data (such as integers or reals) you must first convert the data to strings and store it in the string sequence.

Writing Messages to an Internal Buffer

Methods Introduced:

- **pfcSession.BaseSession.GetMessageContents**
- **pfcSession.BaseSession.GetLocalizedMessageContents**

The methods `pfcSession.BaseSession.GetMessageContents` and `pfcSession.BaseSession.GetLocalizedMessageContents` enable you to write a message to an internal buffer instead of the PTC Creo Parametric message area.

These methods take the same input arguments and perform exactly the same argument substitution and translation as the `pfcSession.Session.UIDisplayMessage` and `pfcSession.Session.UIDisplayLocalizedMessage` methods described in the previous section.

Message Classification

Messages displayed in J-Link include a symbol that identifies the message type. Every message type is identified by a classification that begins with the characters `%C`. A message classification requires that the message key line (line one in the message file) must be preceded by the classification code.

> **Note**
>
> Any message key string used in the code should not contain the classification.

J-Link applications can now display any or all of the following message symbols:

- `Prompt`—This J-Link message is preceded by a green arrow. The user must respond to this message type. Responding includes, specifying input information, accepting the default value offered, or canceling the application. If no action is taken, the progress of the application is halted. A response may either be textual or a selection. The classification for Prompt messages is `%CP`
- `Info`—This J-Link message is preceded by a blue dot. Info message types contain information such as user requests or feedback from J-Link or PTC Creo Parametric. The classification for Info messages is `%CI`

  > **Note**
  >
  > Do not classify messages that display information regarding problems with an operation or process as `Info`. These types of messages must be classified as `Warnings`.

- `Warning`—This J-Link message is preceded by a triangle containing an exclamation point. Warning message types contain information to alert users to situations that could potentially lead to an error during a later stage of the process. Examples of warnings could be a process restriction or a suspected data problem. A Warning will not prevent or interrupt a process. Also, a Warning should not be used to indicate a failed operation. Warnings must only caution a user that the completed operation may not have been performed in a completely desirable way. The classification for Warning messages is `%CW`

- Error—This J-Link message is preceded by a a broken square. An Error message informs the user that a required task was not completed successfully. Depending on the application, a failed task may or may not require intervention or correction before work can continue. Whenever possible redress this situation by providing a path. The classification for Error messages is %CE
- Critical—This J-Link message is preceded by a red X. A Critical message type informs the user of an extremely serious situation that is usually preceeded by loss of user data. Options redressing this situation, if available, should be provided within the message. The classification for a Critical messages is %CC

Example Code: Writing a Message

The sample code in the file pfcSessionExamples.java located at <creo_jlink_loadpoint>/jlink_appls/jlinkexamples demonstrates how to write a message to the message window. The program uses the message file mymessages.txt.

Reading Data from the Message Window

Methods Introduced:

- **pfcSession.Session.UIReadIntMessage**
- **pfcSession.Session.UIReadRealMessage**
- **pfcSession.Session.UIReadStringMessage**

These methods enable a program to get data from the user.

The pfcSession.Session.UIReadIntMessage and pfcSession.Session.UIReadRealMessage methods contain optional arguments that can be used to limit the value of the data to a certain range.

The method pfcSession.Session.UIReadStringMessage includes an optional Boolean argument that specifies whether to echo characters entered onto the screen. You would use this argument when prompting a user to enter a password.

Displaying Feature Parameters

Method Introduced:

- **pfcSession.Session.UIDisplayFeatureParams**

The methodpfcSession.Session.UIDisplayFeatureParams forces PTC Creo Parametric to show dimensions or other parameters stored on a specific feature. The displayed dimensions may then be interactively selected by the user.

File Dialogs

Methods Introduced:

- **pfcSession.Session.UIOpenFile**
- **pfcUI.pfcUI.FileOpenOptions_Create**
- **pfcUI.FileOpenOptions.SetFilterString**
- **pfcUI.FileOpenOptions.SetPreselectedItem**
- **pfcUI.FileUIOptions.SetDefaultPath**
- **pfcUI.FileUIOptions.SetDialogLabel**
- **pfcUI.FileUIOptions.SetShortcuts**
- **pfcUI.pfcUI.FileOpenShortcut_Create**
- **pfcUI.FileOpenShortcut.SetShortcutName**
- **pfcUI.FileOpenShortcut.SetShortcutPath**
- **pfcSession.Session.UISaveFile**
- **pfcUI.pfcUI.FileSaveOptions_Create**
- **pfcSession.Session.UISelectDirectory**
- **pfcUI.pfcUI.DirectorySelectionOptions_Create**

The method `pfcSession.Session.UIOpenFile` opens the relevant dialog box for browsing directories and opening files. The method lets you specify several options through the input arguments `pfcUI.FileOpenOptions` and `pfcUI.FileUIOptions`.

Use the method `pfcUI.pfcUI.FileOpenOptions_Create` to create a new instance of the `pfcUI.FileOpenOptions` object. This object contains the following options:

- `FilterString`—Specifies the filter string for the type of file accepted by the dialog box. Multiple file types should be listed with wildcards and separated by commas, for example, `*.prt`, `*.asm`, `*.txt`, `*.avi`, and so on. Use the method `pfcUI.FileOpenOptions.SetFilterString` to set this option.
- `PreselectedItem`—Specifies the name of an item to preselect in the dialog box. Use the method `pfcUI.FileOpenOptions.SetPreselectedItem` to set this option.

The `pfcUI.FileUIOptions` object contains the following options:

- `DefaultPath`—Specifies the name of the path to be opened by default in the dialog box. Use the method `pfcUI.FileUIOptions.SetDefaultPath` to set this option.
- `DialogLabel`—Specifies the title of the dialog box. Use the method `pfcUI.FileUIOptions.SetDialogLabel` to set this option.
- `Shortcuts`—Specifies an array of file shortcuts of the type `pfcUI.FileOpenShortcut`. Create this object using the method `pfcUI.FileOpenShortcut_Create`. This object contains the following attributes:
  - `ShortcutName`—Specifies the name of shortcut path to be made available in the dialog box.
  - `ShortcutPath`—Specifies the string for the shortcut path.

  Use the method `pfcUI.FileUIOptions.SetShortcuts` to set the array of file shortcuts.

The method `pfcSession.Session.UIOpenFile` returns the file selected by you. The application must use other methods or techniques to perform the desired action on the file.

The method `pfcSession.Session.UISaveFile` opens the relevant dialog box for saving a file. The method accepts options similar to `pfcSession.Session.UIOpenFile` through the `pfcUI.FileSaveOptions` and `pfcUI.FileUIOptions` objects. Use the method `pfcUI.pfcUI.FileSaveOptions_Create` to create a new instance of the `pfcUI.FileSaveOptions` object. When using the **Save** dialog box, you can set the name to a non-existent file. The method `pfcSession.Session.UISaveFile` returns the name of the file selected by you; the application must use other methods or techniques to perform the desired action on the file.

The method `pfcSession.Session.UISelectDirectory` prompts the user to select a directory using the PTC Creo Parametric dialog box for browsing directories. The method accepts options through the `pfcUI.DirectorySelectionOptions` object which is similar to the `pfcUI.FileUIOptions` object (described for the method `pfcSession.Session.UIOpenFile`). Specify the default directory path, the title of the dialog box, and a set of shortcuts to other directories to start browsing. If the default path is specified as `NULL`, the current directory is used. Use the method `pfcUI.pfcUI.DirectorySelectionOptions_Create` to create a new instance of the `pfcUI.DirectorySelectionOptions` object. The method `pfcSession.Session.UISelectDirectory` returns the selected directory path; the application must use other methods or techniques to perform other relevant tasks with this selected path.

Customizing the PTC Creo Parametric Navigation Area

The PTC Creo Parametric navigation area includes the Model and Layer Tree pane, Folder browser pane, and Favorites pane. The methods described in this section enable J-Link applications to add custom panes that contain Web pages to the PTC Creo Parametric navigation area.

Adding Custom Web Pages

To add custom Web pages to the navigation area, the J-Link application must:

1. Add a new pane to the navigation area.
2. Set an icon for this pane.
3. Set the URL of the location that will be displayed in the pane.

Methods Introduced:

- **pfcSession.Session.NavigatorPaneBrowserAdd**
- **pfcSession.Session.NavigatorPaneBrowserIconSet**
- **pfcSession.Session.NavigatorPaneBrowserURLSet**

The method `pfcSession.Session.NavigatorPaneBrowserAdd` adds a new pane that can display a Web page to the navigation area. The input parameters are:

- *PaneName*—Specify a unique name for the pane. Use this name in susbsequent calls to `pfcSession.Session.NavigatorPaneBrowserIconSet` and `pfcSession.Session.NavigatorPaneBrowserURLSet`.
- *IconFileName*—Specify the name of the icon file, including the extension. A valid format for the icon file is the PTC-proprietary format used by PTC Creo Parametric `.BIF`, `.GIF`, `.JPG`, or `.PNG`. The new pane is displayed with the icon image. If you specify the value as `NULL`, the default PTC Creo Parametric icon is used.

  The default search paths for finding the icons are:

  - `<creo_loadpoint>\<datecode>\Common Files\text\resource`
  - `<Application text dir>\resource`
  - `<Application text dir>\<language>\resource`

  The location of the application text directory is specified in the registry file.
- *URL*—Specify the URL of the location to be accessed from the pane.

Use the method `pfcSession.Session.NavigatorPaneBrowserIconSet` to set or change the icon of a specified browser pane in the navigation area.

Use the method `pfcSession.Session.NavigatorPaneBrowserURLSet` to change the URL of the page displayed in the browser pane in the navigation area.

7

Selection

This chapter describes how to use Interactive Selection in J-Link.

Interactive Selection

Methods Introduced:

- **pfcSession.BaseSession.Select**
- **pfcSelect.pfcSelect.SelectionOptions_Create**
- **pfcSelect.SelectionOptions.SetMaxNumSels**
- **pfcSelect.SelectionOptions.SetOptionKeywords**

The method `pfcSession.BaseSession.Select` activates the standard PTC Creo Parametric menu structure for selecting objects and returns a `pfcSelect.Selections` sequence that contains the objects the user selected. Using the *Options* argument, you can control the type of object that can be selected and the maximum number of selections.

In addition, you can pass in a `pfcSelect.Selections` sequence to the method. The returned `pfcSelect.Selections` sequence will contain the input sequence and any new objects.

The methods `pfcSelect.pfcSelect.SelectionOptions_Create` `pfcSelect.SelectionOptions.SetOptionKeywords` take a *String* argument made up of one or more of the identifiers listed in the table below, separated by commas.

For example, to allow the selection of features and axes, the arguments would be *feature*, *axis*.

PTC Creo Parametric Database Item	String Identifier	ModelItemType
Datum point	*point*	`ITEM_POINT`
Datum axis	*axis*	`ITEM_AXIS`
Datum plane	*datum*	`ITEM_SURFACE`
Coordinate system datum	*csys*	`ITEM_COORD_SYS`
Feature	*feature*	`ITEM_FEATURE`
Edge (solid or datum surface)	*edge*	`ITEM_EDGE`
Edge (solid only)	*sldedge*	`ITEM_EDGE`
Edge (datum surface only)	*qltedge*	`ITEM_EDGE`
Datum curve	*curve*	`ITEM_CURVE`
Composite curve	*comp_crv*	`ITEM_CURVE`
Surface (solid or quilt)	*surface*	`ITEM_SURFACE`
Surface (solid)	*sldface*	`ITEM_SURFACE`
Surface (datum surface)	*qltface*	`ITEM_SURFACE`
Quilt	*dtmqlt*	`ITEM_QUILT`
Dimension	*dimension*	`ITEM_DIMENSION`
Reference dimension	*ref_dim*	`ITEM_REF_DIMENSION`
Integer parameter	*ipar*	`ITEM_DIMENSION`
Part	*part*	N/A
Part or subassembly	*prt_or_asm*	N/A

PTC Creo Parametric Database Item	String Identifier	ModelItemType
Assembly component model	*component*	N/A
Component or feature	*membfeat*	`ITEM_FEATURE`
Detail symbol	*dtl_symbol*	`ITEM_DTL_SYM_INSTANCE`
Note	*any_note*	`ITEM_NOTE`, `ITEM_DTL_NOTE`
Draft entity	*draft_ent*	`ITEM_DTL_ENTITY`
Table	*dwg_table*	`ITEM_TABLE`
Table cell	*table_cell*	`ITEM_TABLE`
Drawing view	*dwg_view*	N/A

When you specify the maximum number of selections, the argument to `pfcSelect.SelectionOptions.SetMaxNumSels` must be an Integer. The code will be as follows:

```
sel_options.setMaxNumSels (new Integer (10));
```

The default value assigned when creating a `SelectionOptions` object is –1, which allows any number of selections by the user.

Accessing Selection Data

MethodsIntroduced:

- **pfcSelect.Selection.GetSelModel**
- **pfcSelect.Selection.GetSelItem**
- **pfcSelect.Selection.GetPath**
- **pfcSelect.Selection.GetParams**
- **pfcSelect.Selection.GetTParam**
- **pfcSelect.Selection.GetPoint**
- **pfcSelect.Selection.GetDepth**
- **pfcSelect.Selection.GetSelView2D**
- **pfcSelect.Selection.GetSelTableCell**
- **pfcSelect.Selection.GetSelTableSegment**

These methods return objects and data that make up the selection object. Using the appropriate methods, you can access the following data:

- For a selected model or model item use `pfcSelect.Selection.GetSelModel` or `pfcSelect.Selection.GetSelItem`.
- For an assembly component use `pfcSelect.Selection.GetPath`.
- For UV parameters of the selection point on a surface use `pfcSelect.Selection.GetParams`.

- For the T parameter of the selection point on an edge or curve use `pfcSelect.Selection.GetTParam`.
- For a three-dimensional point object that contains the selected point use `pfcSelect.Selection.GetPoint`.
- For selection depth, in screen coordinates use `pfcSelect.Selection.GetDepth`.
- For the selected drawing view, if the selection was from a drawing, use `pfcSelect.Selection.GetSelView2D`.
- For the selected table cell, if the selection was from a table, use `pfcSelect.Selection.GetSelTableCell`.
- For the selected table segment, if the selection was from a table, use `pfcSelect.Selection.GetSelTableSegment`.

Controlling Selection Display

Methods Introduced:

- **pfcSelect.Selection.Highlight**
- **pfcSelect.Selection.UnHighlight**
- **pfcSelect.Selection.Display**

These methods cause a specific selection to be highlighted or dimmed on the screen using the color specified as an argument.

The method `pfcSelect.Selection.Highlight` highlights the selection in the current window. This highlight is the same as the one used by PTC Creo Parametric when selecting an item—it just repaints the wire-frame display in the new color. The highlight is removed if you use the **View**, **Repaint** command or `pfcWindow.Window.Repaint`; it is not removed if you use `pfcWindow.Window.Refresh`.

The method `pfcSelect.Selection.UnHighlight` removes the highlight.

The method `pfcSelect.Selection.Display` causes a selected object to be displayed on the screen, even if it is suppressed or hidden.

> **Note**
>
> This is a one-time action and the next repaint will erase this display.

Example Code: Using Interactive Selection

The sample code in the file pfcSessionExamples.java located at <creo_jlink_loadpoint>/jlink_appls/jlinkexamples demonstrates how to use J-Link to allow interactive selection.

Programmatic Selection

J-Link provides methods whereby you can make your own Selection objects, without prompting the user. These Selections are required as inputs to some methods and can also be used to highlight certain objects on the screen.

Methods Introduced:

- **pfcSelect.pfcSelect.CreateModelItemSelection**
- **pfcSelect.pfcSelect.CreateComponentSelection**
- **pfcSelect.pfcSelect.CreateSelectionFromString**
- **pfcSelect.Selection.SetSelItem**
- **pfcSelect.Selection.SetPath**
- **pfcSelect.Selection.SetParams**
- **pfcSelect.Selection.SetTParam**
- **pfcSelect.Selection.SetPoint**
- **pfcSelect.Selection.SetSelTableCell**
- **pfcSelect.Selection.SetSelView2D**

The method pfcSelect.pfcSelect.CreateModelItemSelection creates a selection out of any model item object. It takes a pfcModelItem.ModelItem and optionally a pfcAssembly.ComponentPath object to identify which component in an assembly the Selection Object belongs to.

The method pfcSelect.pfcSelect.CreateComponentSelection creates a selection out of any component in an assembly. It takes a pfcAssembly.ComponentPath object. For more information about pfcAssembly.ComponentPath objects, see the section Getting a Solid Object on page 176 in theSolid on page 175chapter.

The method pfcSelect.pfcSelect.CreateSelectionFromString creates a new selection object, based on a Web.Link style selection string specified as the input.

Some J-Link methods require more information to be set in the selection object. The methods allow you to set the following:

The selected item using the method pfcSelect.Selection.SetSelItem.

The selected component path using the method `pfcSelect.Selection.SetPath`.

The selected UV parameters using the method `pfcSelect.Selection.SetParams`.

The selected T parameter (for a curve or edge), using the method `pfcSelect.Selection.SetTParam`.

The selected XYZ point using the method `pfcSelect.Selection.SetPoint`.

The selected table cell using the method `pfcSelect.Selection.SetSelTableCell`.

The selected drawing view using the method `pfcSelect.Selection.SetSelView2D`.

Selection Buffer

Introduction to Selection Buffers

Selection is the process of choosing items on which you want to perform an operation. In PTC Creo Parametric, before a feature tool is invoked, the user can select items to be used in a given tool's collectors. Collectors are like storage bins of the references of selected items. The location where preselected items are stored is called the selection buffer.

Depending on the situation, different selection buffers may be active at any one time. In Part and Assembly mode, PTC Creo Parametric offers the default selection buffer, the Edit selection buffer, and other more specialized buffers. Other PTC Creo Parametric modes offer different selection buffers.

In the default Part and Assembly buffer there are two levels at which selection is done:

- First Level Selection

  Provides access to higher-level objects such as features or components. You can make a second level selection only after you select the higher-level object.

- Second Level Selection

  Provides access to geometric objects such as edges and faces.

Note

First-level and second-level objects are usually incompatible in the selection buffer.

J-Link allows access to the contents of the currently active selection buffer. The available functions allow your application to:

- Get the contents of the active selection buffer.
- Remove the contents of the active selection buffer.
- Add to the contents of the active selection buffer.

Reading the Contents of the Selection Buffer

Methods Introduced:

- **pfcSession.Session.GetCurrentSelectionBuffer()**
- **pfcSelect.SelectionBuffer.GetContents()**

The method `pfcSession.Session.GetCurrentSelectionBuffer` returns the selection buffer object for the current active model in session. The selection buffer contains the items preselected by the user to be used by the selection tool and popup menus.

Use the method `pfcSelect.SelectionBuffer.GetContents` to access the contents of the current selection buffer. The method returns independent copies of the selections in the selection buffer (if the buffer is cleared, this array is still valid).

Removing the Items of the Selection Buffer

Methods Introduced:

- **pfcSelect.SelectionBuffer.RemoveSelection**
- **pfcSelect.SelectionBuffer.Clear**

Use the method `pfcSelect.SelectionBuffer.RemoveSelection` to remove a specific selection from the selection buffer. The input argument is the *IndexToRemove* specifies the index where the item was found in the call to the method `pfcSelect.SelectionBuffer.GetContents`.

Use the method `pfcSelect.SelectionBuffer.Clear` to clear the currently active selection buffer of all contents. After the buffer is cleared, all contents are lost.

Adding Items to the Selection Buffer

Method Introduced:

- **pfcSelect.SelectionBuffer.AddSelection**

Use the method `pfcSelect.SelectionBuffer.AddSelection` to add an item to the currently active selection buffer.

Note

The selected item must refer to an item that is in the current model such as its owner, component path or drawing view.

This method may fail due to any of the following reasons:

- There is no current selection buffer active.
- The selection does not refer to the current model.
- The item is not currently displayed and so cannot be added to the buffer.
- The selection cannot be added to the buffer in combination with one or more objects that are already in the buffer. For example: geometry and features cannot be selected in the default buffer at the same time.

8

Ribbon Tabs, Groups, and Menu Items

This chapter describes the J-Link support for the Ribbon User Interface (UI). It also describes the impact of the ribbon user interface on legacy J-Link applications and the procedure to place the commands, buttons, and menu items created by the legacy applications in the PTC Creo Parametric ribbon user interface. Refer to the PTC Creo Parametric Help for more information on the ribbon user interface and the procedure to customize the ribbon.

Creating Ribbon Tabs, Groups, and Menu Items

Customizations to the ribbon user interface using the J-Link applications are supported through the **Customize Ribbon** tab in the **Creo Parametric Options** dialog box. You can specify the user interface layout for a J-Link application and save the layout definition in a ribbon definition file, `toolkitribbonui.rbn`. Set the configuration option `tk_enable_ribbon_custom_save` to `true` before customizing the ribbon user interface using the J-Link application. When you run PTC Creo Parametric, the `toolkitribbonui.rbn` file is loaded along with the J-Link application and the commands created by the J-Link application appear in the ribbon user interface. Refer to the section About the Ribbon Definition File on page 90 for more information on the `toolkitribbonui.rbn` file.

You can customize the ribbon user interface only for a particular mode in PTC Creo Parametric. For example, if you customize the ribbon user interface and save it to the `toolkitribbonui.rbn` file in the Part mode, then on loading PTC Creo Parametric the customized user interface will be visible only in the Part mode. To view a particular tab or group in all the modes, you must customize the ribbon user interface and save the `toolkitribbonui.rbn` file in each mode. Refer to the PTC Creo Parametric Fundamentals Help for more information on customizing the ribbon.

Note

You can add a new group to an existing tab or create a new tab using the **Customize Ribbon** tab in the **Creo Parametric Options** dialog box. You will not be able to modify the tabs or groups that are defined by PTC Creo Parametric.

Workflow to Add Menu Items to the Ribbon User Interface

Note

The instructions explained below are applicable only if the application is implemented in full asynchronous mode. This is because applications in simple asynchronous mode cannot handle requests, that is, command callbacks, from PTC Creo Parametric. Refer to the chapter on Asynchronous Mode, for more information.

Set the configuration option `tk_enable_ribbon_custom_save` to `true` before customizing the ribbon user interface. The steps to add commands to the PTC Creo Parametric ribbon user interface are as follows:

1. Create a J-Link application with complete command definition, which includes specifying command label, help text, large icon name, and small icon name. Designate the command using `pfcCommand.UICommand.Designate`.
2. Start the J-Link application and ensure that it has started or connected to PTC Creo Parametric. The commands created by the J-Link application will be loaded in PTC Creo Parametric.
3. Click **File ▸ Options**. The **Creo Parametric Options** dialog box opens.
4. Click **Customize Ribbon**.
5. In the **Customize the Ribbon** list, select a tab and create a new group in it or create a new tab and a group in it.
6. In the **Choose commands from** list, select **TOOLKIT Commands**. The commands created by the J-Link application are displayed.
7. Click **Add** to add the commands to the new tab or group.
8. Click **Import/Export ▸ Save the Auxilliary Application User Interface**. The changes are saved to the `toolkitribbonui.rbn` file. The `toolkitribbonui.rbn` file is saved in the text folder specified in the J-Link application registry file. For more information refer to the section on Ribbon Definition File on page 90.

> **Note**
>
> The **Save the Auxilliary Application User Interface** button is enabled only if you set the configuration option `tk_enable_ribbon_custom_save` to `true`.

9. Click **Apply**. The custom settings are saved to the `toolkitribbonui.rbn` file.
10. Reload the J-Link application or restart PTC Creo Parametric. The `toolkitribbonui.rbn` file will be loaded along with the J-Link application.

If translated messages are available for the newly added tabs or groups, then PTC Creo Parametric displays the translated strings by searching for the same string from the list of string based messages that are loaded. For more information refer to the section on Localizing the Ribbon User Interface Created by the J-Link Applications on page 93

About the Ribbon Definition File

A ribbon definition file is a file that is created through the **Customize Ribbon** interface in PTC Creo Parametric. This file defines the containers, that is, Tabs, Group, or Cascade menus that are created by a particular J-Link application. It contains information on whether to show an icon or label. It also contains the size of the icon to be used, that is, a large icon (32X32) or a small icon (16x16).

The ribbon user interface displays the commands referenced in the ribbon definition file only if the commands are loaded and are visible in that particular PTC Creo Parametric mode. If translated messages are available for the newly added tabs or groups, then PTC Creo Parametric displays the translated strings by searching for the same string from the list of string based messages that are loaded. For more information refer to the section on Localizing the Ribbon User Interface Created by the J-Link Applications on page 93.

Set the configuration option `tk_enable_ribbon_custom_save` to `true` before customizing the ribbon user interface. To save the ribbon user interface layout definition to the `toolkitribbonui.rbn` file:

1. Click **File** ▸ **Options**. The **Creo Parametric Options** dialog box opens.
2. Click **Customize Ribbon**.
3. In the **Customize the Ribbon** list, select a tab and create a new group in it or create a new tab and a group in it.
4. In the **Choose commands from** list, select **TOOLKIT Commands**. The commands created by the J-Link application are displayed.
5. Click **Add** to add the commands to the new tab or group.
6. Click **Import/Export** ▸ **Save the Auxilliary Application User Interface**. The modified layout is saved to the `toolkitribbonui.rbn` file located in the `text` folder within the J-Link application directory, that is, <application_dir>/text

   Note

   The **Save the Auxilliary Application User Interface** button is enabled only if you set the configuration option `tk_enable_ribbon_custom_save` to `true`.

7. Click **OK**.

Note

You cannot edit the `toolkitribbonui.rbn` file manually.

To Specify the Path for the Ribbon Definition File

You can rename the `toolkitribbonui.rbn` to another filename with the `.rbn` extension. To enable the J-Link application to read the ribbon definition file having a name other than `toolkitribbonui.rbn`, it must be available at the location `<application_dir>/text/ribbon`. The method introduced in this section enables you to load the ribbon definition file from within a J-Link application.

method Introduced:

- **pfcSession.Session.RibbonDefinitionfileLoad**

The method `pfcSession.Session.RibbonDefinitionfileLoad` loads a specified ribbon definition file from a default path into the PTC Creo Parametric application. The input argument is as follows:

- *file_name* - Specify the name of the ribbon definition file including its extension. The default search path for this file is:
  - The working directory from where PTC Creo Parametric is loaded.
  - `<application_text_dir>/ribbon`
  - `<application_text_dir>/language/ribbon`

> **Note**
>
> - The location of the application text directory is specified in the J-Link registry file.
> - A J-Link application can load a ribbon definition file only once. After the application has loaded the ribbon, calls made to the method `pfcSession.Session.RibbonDefinitionfileLoad` to load other ribbon definition files are ignored.

Loading Multiple Applications Using the Ribbon Definition File

PTC Creo Parametric supports loading of multiple `.rbn` files in the same session. You can develop multiple J-Link applications that share the same tabs or groups and each application will have its own ribbon definition file. As each application is loaded, its `.rbn` file will be read and applied. When an application is unloaded, the containers and command created by its `.rbn` file will be removed.

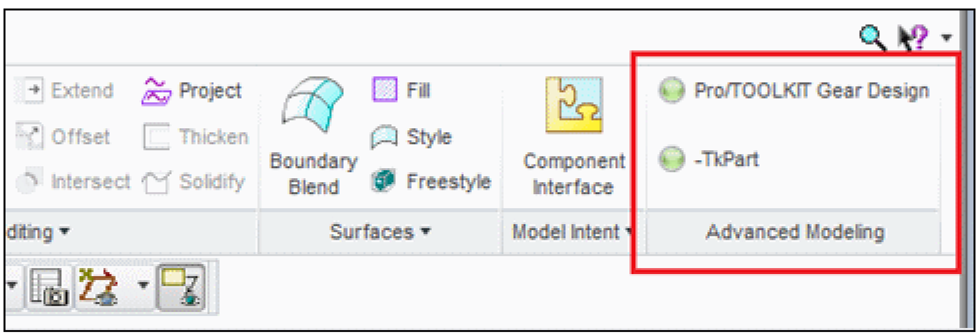
For example, consider two J-Link applications, namely, `pt_geardesign` and `pt_examples` that add commands to the same group on a tab on the Ribbon user interface. The application `pt_geardesign` adds a command **Pro/TOOLKIT Gear Design** to the **Advanced Modeling** group on the **Modeling** tab and the application pt_examples adds a command **TKPart** to the **Advanced Modeling** group on the **Model** tab. The ribbon definition file for each application will contain an instruction to create the **Advanced Modeling** group and if both the ribbon files are loaded, the group will be created only once and the two ribbon customizations will be merged into the same group.

That is, if both the applications are running in the same PTC Creo Parametric session, then the commands, **Pro/TOOLKIT Gear Design** and **TKPart** will be available under the **Advanced Modeling** group on the **Model** tab.

Note

The order in which the commands will be displayed within the group will depend on the order of loading of the .rbn file for each application.

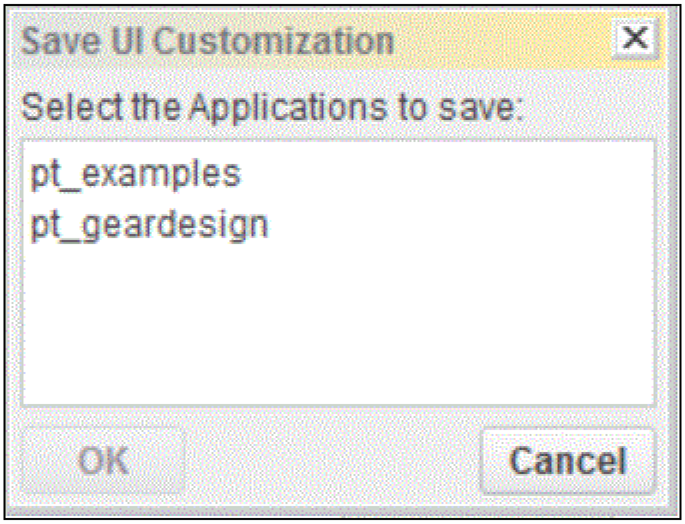
The following image displays commands added by two J-Link applications to the same group.



To save the customization when multiple applications are loaded:

1. Click **File ▸ Options**. The **Creo Parametric Options** dialog box opens.
2. Click **Customize Ribbon**.
3. In the **Customize the Ribbon** list, select a tab and create a new group in it or create a new tab and a group in it.
4. In the **Choose commands from list**, select **TOOLKIT Commands**. The commands created by the J-Link application are displayed.
5. Click **Add** to add the commands to the new tab or group.
6. Click **Import/Export ▸ Save the Auxilliary Application User Interface**. The **Save UI Customization** dialog box opens.
7. Select a J-Link application and Click **Save**. The modified layout is saved to the .rbn file of the specified J-Link application.

The **Save UI Customization** dialog box is shown in the following image:



Localizing the Ribbon User Interface Created by the J-Link Applications

The labels for the custom tabs, groups, and cascade menus belonging to the J-Link application can be translated in the languages supported by PTC Creo Parametric. To display localized labels, specify the translated labels in the ribbonui.txt file and save this file at the location <application_text_dir>/<language>. For example, the text file for German locale must be saved at the location <application_text_dir>/german/ribbonui.txt.

Create a file containing translations for each of the languages in which the J-Link application is localized. The Localized translation files must use the UTF-8 encoding with BOM character for the translated text to be displayed correctly in the user interface.

The format of the ribbonui.txt file is as shown below. Specify the following lines for each label entry in the file:

1. A hash sign (#) followed by the label, as specified in the ribbon definition file.
2. The label as specified in the ribbon definition file and as displayed in the ribbon user interface.
3. The translated label.
4. Add an empty line at the end of each label entry in the file.

For example, if the PTC Creo Parametric application creates a tab with the name TK_TAB having a group with the name TK_GROUP, then the translated file will contain the following:

```
#TK_TAB
```

```
TK_TAB
<translation for TK_TAB>
<Empty_line>
#TK_GROUP
TK_GROUP
<translation for TK_GROUP>
<Empty_line>
```

Support for Legacy J-Link Applications

The user interface for Creo Parametric 1.0 has been restructured to a ribbon user interface. This may affect the behavior of existing J-Link applications that were designed to add commands to specific Pro/ENGINEER menus or toolbars. These menus or toolbars or both have been redesigned in Creo Parametric 1.0. The commands added by the J-Link applications appear on the PTC Creo Parametric ribbon in the **Home** tab under the **Pro/TK** group. When you open a model, the **Pro/TK** group is on the **Tools** tab.

You can also arrange the commands added by the J-Link applications under a new tab or an existing tab by customizing the ribbon using the **Customize Ribbon** tab in the **Creo Parametric Options** dialog box. For a list of all the commands added by the J-Link applications, follow this procedure:

1. Click **File** ▸ **Options**. The **Creo Parametric Options** dialog box opens.
2. Click **Customize Ribbon**.
3. In the **Choose commands from list**, select **TOOLKIT Commands**. All the commands added by legacy J-Link applications are listed.

Note

Commands that have not been designated will not have an icon or will have a generic icon.

Refer to the PTC Creo Parametric Help for more information on customizing the Ribbon.

Migration of Legacy J-Link Applications

To migrate existing J-Link applications to the PTC Creo Parametric Ribbon user interface without compiling the source code:

1. Load the J-Link applications in PTC Creo Parametric so that the commands created in these applications are available in the **Customize Ribbon** user interface.
2. Modify the ribbon user interface layout and save the changes to the `toolkitribbonui.rbn`.
3. Copy the `toolkitribbonui.rbn` to the location <application_text_dir>/ribbon.
4. Reload J-Link application or restart PTC Creo Parametric. The `.rbn` file will be loaded along with the J-Link application and the commands will be visible in the ribbon user interface if its accessibility will be visible.

9

Menus, Commands, and Pop-up Menus

This chapter describes the methods provided by J-Link to create and modify menus, buttons, and pop-up menus in the PTC Creo Parametric user interface.

Refer to the chapter Ribbon Tabs, Groups, and Menu Items on page 87 for more information. Also, refer to the PTC Creo Parametric Help for more information on customizing the ribbon user interface.

Introduction

The J-Link classes enable you to supplement the PTC Creo Parametric ribbon user interface.

Once the J-Link application is loaded, you can add a new group to an existing tab or create a new tab using the **Customize Ribbon** tab in the **Creo Parametric Options** dialog box in PTC Creo Parametric. You will not be able to modify the groups that are defined by PTC Creo Parametric. If the translated messages are available for the newly added tabs or groups, then PTC Creo Parametric will use them by searching for the same string in the list of sting based messages loaded.

You can customize the ribbon user interface only for a particular mode in PTC Creo Parametric. For example, if you customize the ribbon user interface and save it to the `toolkitribbonui.rbn` file in the Part mode, then on loading PTC Creo Parametric the customized user interface will be visible only in the Part mode. To view a particular tab or group in all the modes, you must customize the ribbon user interface and save the `toolkitribbonui.rbn` file in each mode. Refer to the PTC Creo Parametric Fundamentals Help for more information on customizing the ribbon.

Menu Bar Definitions

- Menu—A menu, such as the **File** menu, or a sub-menu, such as the **Manage File** menu under the **File** menu.
- Menu button—A named item in a group or menu that is used to launch a set of instructions.
- Pop-up menu—A menu invoked by selection of an item in the PTC Creo Parametric graphics window.
- Command—A procedure in PTC Creo Parametric that may be activated from a button.

Menus Buttons and Menus

The following methods enable you to add new menu buttons in any location on the Ribbon user interface.

Methods Introduced:

- **pfcSession.Session.UICreateCommand**
- **pfcSession.Session.UICreateMaxPriorityCommand**
- **pfcCommand.UICommandActionListener.OnCommand**

The method `pfcSession.Session.UICreateCommand` creates a `pfcUICommand` object that contains a `pfcCommand.UICommandActionListener`. You should override the `pfcCommand.UICommandActionListener.OnCommand` method with the code that you want to execute when the user clicks a button.

The method `pfcSession.Session.UICreateMaxPriorityCommand` creates a `pfcUICommand` object having maximum command priority. The priority of the action refers to the level of precedence the added action takes over other PTC Creo Parametric actions. Maximum priority actions dismiss all other actions except asynchronous actions.

Maximum command priority should be used only in commands that open and activate a new model in a window. Create all other commands using the method `pfcSession.Session.UICreateCommand`.

The listener method `pfcCommand.UICommandListener.OnCommand` is called when the command is activated in PTC Creo Parametric by pressing a button.

Designate the command using the function `pfcCommand.UICommand.Designate` and add a button to the ribbon user interface using the using the **Customize Ribbon** tab in the **Creo Parametric Options** dialog box. This operation binds the command to the button.

Finding PTC Creo Parametric Commands

This method enables you to find existing PTC Creo Parametric commands in order to modify their behavior.

Method Introduced:

- **pfcSession.Session.UIGetCommand**

The method `pfcSession.Session.UIGetCommand` returns a `pfcCommand.UICommand` object representing an existing PTC Creo Parametric command. The method allows you to find the command ID for an existing command so that you can add an access function or bracket function to the command. You must know the name of the command in order to find its ID.

To find the name of an action command, click the corresponding icon on the ribbon user interface and then check the last entry in the trail file. For example, for the Save icon, the trail file will have the corresponding entry:

```
~ Command `ProCmdModelSave`
```

The action name for the `Save` icon is `ProCmdModelSave`. This string can be used as input to `pfcSession.Session.UIGetCommand` to get the command ID.

You can determine a command ID string for an option without an icon by searching through the resource files located in the <creo_loadpoint>\<datecode>\Common Files\text\resource directory. If you search for the menu button name, the line will contain the corresponding action command for the button.

Access Listeners for Commands

These methods allow you to apply an access listener to a command. The access listener determines whether or not the command is visible at the current time in the current session.

Methods Introduced:

- **pfcBase.ActionSource.AddActionListener**
- **pfcCommand.UICommandAccessListener.OnCommandAccess**

Use the method pfcBase.ActionSource.AddActionListener to register a new pfcCommand.UICommandAccessListener on any command (created either by an application or PTC Creo Parametric). This listener will be called when buttons based on the command might be shown.

The listener method pfcCommand.UICommandAccessListener.OnCommandAccess allows you to impose an access function on a particular command. The method determines if the action or command should be available, unavailable, or hidden.

The potential return values are listed in the enumerated type CommandAccess and are as follows:

- ACCESS_REMOVE—The button is not visible and if all of the menu buttons in the containing menu possess an access function returning ACCESS_REMOVE, the containing menu will also be removed from the PTC Creo Parametric user interface..
- ACCESS_INVISIBLE—The button is not visible.
- ACCESS_UNAVAILABLE—The button is visible, but gray and cannot be selected.
- ACCESS_DISALLOW—The button shows as available, but the command will not be executed when it is chosen.
- ACCESS_AVAILABLE—The button is not gray and can be selected by the user. This is the default value.

Example 1: Command Access Listeners

The sample code in the file pfcCommandExamples.java located at <creo_jlink_loadpoint>/jlink_appls/jlinkexamples demonstrates the usage of the access listener method for a particular command. The CommandAccessOnCommandAccess function returns ACCESS_UNAVAILABLE that disables button associated with the command, if the model is not present or if it is not of type PART. Else, the function returns ACCESS_AVAILABLE that enables button associated with the command.

Bracket Listeners for Commands

These methods allow you to apply a bracket listener to a command. The bracket listener is called before and after the command runs, which allows your application to provide custom logic to execute whenever the command is selected.

Methods Introduced:

- **pfcBase.ActionSource.AddActionListener**
- **pfcCommand.UICommandBracketListener.OnBeforeCommand**
- **pfcCommand.UICommandBracketListener.OnAfterCommand**

Use the method pfcBase.ActionSource.AddActionListener to register a new pfcCommand.UICommandBracketListener on any command (created either by an application or PTC Creo Parametric. This listener will be called when the command is selected by the user.

The listener methods pfcCommand.UICommandBracketListener.OnBeforeCommand and pfcCommand.UICommandBracketListener.OnAfterCommand allow the creation of functions that will be called immediately before and after execution of a given command. These methods are used to add the business logic to the start or end (or both) of an existing PTC Creo Parametric command.

The method pfcCommand.UICommandBracketListener.OnBeforeCommand could also be used to cancel an upcoming command. To do this, throw a pfcExceptions.XCancelProEAction exception from the body of the listener method using pfcExceptions.XCancelProEAction.Throw.

Example 2: Bracket Listeners

The sample code in the file pfcCommandExamples.java located at <creo_jlink_loadpoint>/jlink_appls/jlinkexamples demonstrates the usage of the bracket listener methods that are called before and after when the user tries to rename the model. If the model contains a certain parameter, the rename attempt will be aborted by this listener.

Designating Commands

Using J-Link you can designate PTC Creo Parametric commands. These commands can later appear in the PTC Creo Parametric ribbon user interface.

To add a command you must:

1. Define or add the command to be initiated on clicking the icon in the J-Link application.
2. Optionally designate an icon button to be used with the command.
3. Designate the command to appear in the **Customize Ribbon** tab in the **Creo Parametric Options** dialog box.

> **Note**
>
> Refer to the chapter on Ribbon Tabs, Groups, and Menu Items on page 87 for more information. Also, refer to the PTC Creo Parametric Parametic Help for more information on customizing the Ribbon User Interface.

4. Save the configuration in PTC Creo Parametric so that changes to the ribbon user interface appear when a new session of PTC Creo Parametric is started.

Command Icons

Method Introduced:

- **pfcCommand.UICommand.SetIcon**

The method `pfcCommand.UICommand.SetIcon` allows you to designate an icon to be used with the command you created. The method adds the icon to the PTC Creo Parametric command. Specify the name of the icon file, including the extension as the input argument for this method. A valid format for the icon file is a standard `.GIF`, `.JPG`, or `.PNG`. PTC recommends using `.PNG` format. All icons in the `Creo Parametric` ribbon are either 16x16 (small) or 32x32 (large) size. The naming convention for the icons is as follows:

- Small icon—`<iconname.ext>`
- Large icon—`<iconname_large.ext>`

Note

- The legacy naming convention for icons `<icon_name_icon_size.ext>` will not be supported in future releases of PTC Creo Parametric. The icon size was added as a suffix to the name of the icon. For example, the legacy naming convention for small icons was `iconname16X16.ext`. It is recommended to use the standard naming conventions for icons, that is, `iconname.ext` or `iconname_large.ext`.
- While specifying the name of the icon file, do not specify the full path to the icon names.

The application searches for the icon files in the following locations:

- `<creo_loadpoint>\<datecode>\Common Files\text\resource`
- `<Application text dir>\resource`
- `<Application text dir>\<language>\resource`

  The location of the application text directory is specified in the registry file.

Commands that do not have an icon assigned to them display the button label.

You may also use this method to assign a small icon to a button. The icon appears to the left of the button label.

Designating the Command

Method Introduced:

- **pfcCommand.UICommand.Designate**

This method allows you designate the command as available in the **Customize Ribbon** tab in the **Creo Parametric Options** dialog of PTC Creo Parametric. After a J-Link application has used the method `pfcCommand.UICommand.Designate` on a command, can add the button associated with this command into the PTC Creo Parametric ribbon user interface.

If this method is not called, the button will not be visible in the **Toolkit Commands** list in the **Customize Ribbon** tab in the **Creo Parametric Options** dialog of PTC Creo Parametric.

The arguments to this method are:

- *Label*—The message string that refers to the icon label. This label (stored in the message file) identifies the text seen when the button is displayed. If the command is not assigned an icon, the button label string appears on the toolbar button by default.

- *Help*—The one-line Help for the icon. This label (stored in the message file) identifies the help line seen when the mouse moves over the icon.
- *Description*—The message appears in the **Customize Ribbon** tab in the **Creo Parametric Options** dialog box and also when **Description** is clicked in PTC Creo Parametric.
- *MessageFile*—The message file name. All the labels including the one-line Help labels must be present in the message file.

> **Note**
>
> This file must be in the directory `<text_path>/text` or `<text_path>/text/<language>`.

Placing the Button

Once the button has been created using the methods discussed, place the button on the PTC Creo Parametric ribbon user interface. Refer to the chapter on Ribbon Tabs, Groups, and Menu Items on page 87 for more information. Also, refer to the PTC Creo Parametric Parametic Help for more information on customizing the Ribbon User Interface.

Pop-up Menus

PTC Creo Parametric provides shortcut menus that contain frequently used commands appropriate to the currently selected items. You can access a shortcut menu by right-clicking a selected item. Shortcut menus are accessible in:

- Graphics window
- Model Tree
- Some dialog boxes
- Any area where you can perform an object-action operation by selecting an item and choosing a command to perform on the selected item.

The methods described in this section allow you to add menus to a graphics window pop-up menu.

Adding a Pop-up Menu to the Graphics Window

You can activate different pop-up menus during a given session of PTC Creo Parametric. Every time the PTC Creo Parametric context changes when you open a different model type, enter different tools or special modes such as **Edit**, a different pop-up menu is created. When PTC Creo Parametric moves to the next context, the pop-up menu may be destroyed.

As a result of this, J-Link applications must attach a button to the pop-up menu during initialization of the pop-up menu. The J-Link application is notified each time a particular pop-up menu is created, which then allows the user to add to the pop-up menu.

Use the following procedure to add items to pop-up menus in the graphics window:

1. Obtain the name of the existing pop-up menus to which you want to add a new menu using the trail file.
2. Create commands for the new pop-up menu items.
3. Implement access listeners to provide visibility information for the items.
4. Add an action listener to the session to listen for pop-up menu initialization.
5. In the listener method, if the pop-up menu is the correct menu to which you wish to add the button, then add it.

The following sections describe each of these steps in detail. You can add push buttons and cascade menus to the pop-up menus. You can add pop-up menu items only in the active window. You cannot use this procedure to remove items from existing menus.

Using the Trail File to Determine Existing Pop-up Menu Names

The trail file in PTC Creo Parametric contains a comment that identifies the name of the pop-up menu if the configuration option, `auxapp_popup_menu_info` is set to `yes`.

For example, the pop-up menu, **Edit Properties**, has the following comment in the trail file:

```
~ Close `rmb_popup` `PopupMenu`
~ Activate `rmb_popup` `EditProperties`
!Command ProCmdEditPropertiesDtm was pushed from the software.
!Item was selected from popup menu 'popup_mnu_edit'
```

Listening for Pop-up Menu Initialization

Methods Introduced:

- **pfcBase.ActionSource.AddActionListener**
- **pfcUI.PopupmenuListener.OnPopupmenuCreate**

Use the method `pfcBase.ActionSource.AddActionListener` to register a new `pfcUI.PopupmenuListener` to the session. This listener will be called when pop-up menus are initialized.

The method `pfcUI.PopupmenuListener.OnPopupmenuCreate` is called after the pop-up menu is created internally in PTC Creo Parametric and may be used to assign application-owned buttons to the pop-up menu.

Accessing the Pop-up Menus

The method described in this section provides the name of the pop-up menus used to access these menus while using other methods.

Method Introduced:

- **pfcUI.Popupmenu.GetName**

The method `pfcUI.Popupmenu.GetName()` returns the name of the pop-up menu.

Adding Content to the Pop-up Menus

Methods Introduced:

- **pfcUI.Popupmenu.AddButton**
- **pfcUI.Popupmenu.AddMenu**

Use `pfcUI.Popupmenu.AddButton` to add a new item to a pop-up menu. The input arguments are:

- *Command*—Specifies the command associated with the pop-up menu.
- *Options* – A `pfcUI.PopupmenuOptions` object containing other options for the method. The options that may be included are:
  - `PositionIndex`—Specifies the position in the pop-up menu at which to add the menu button. Pass null to add the button to the bottom of the menu. Use the method `pfcUI.PopupmenuOptions.SetPositionIndex` to set this option.
  - `Name`—Specifies the name of the added button. The button name is placed in the trail file when the user selects the menu button. Use the method `pfcUI.PopupmenuOptions.SetName` to set this option.

- SetLabel—Specifies the button label. This label identifies the text displayed when the button is displayed. Use the method pfcUI.PopupmenuOptions.SetLabel to set this option.
- Helptext—Specifies the help message associated with the button. Use the method pfcUI.PopupmenuOptions.SetHelptext to set this option.

Use the method pfcUI.Popupmenu.AddMenu to add a new cascade menu to an existing pop-up menu.

The argument for this method is a pfcUI.PopupmenuOptions object, whose members have the same purpose as described for newly added buttons. This method returns a new pfcUI.Popupmenu object to which you may add new buttons.

Example 3: Creating a Pop-up Menu

The sample code in the file pfcPopupExamples.java located at <creo_jlink_loadpoint>/jlink_appls/jlinkexamples demonstrates the usage of UI functions to add a new model tree pop-up menu.

10

Models

This chapter describes how to program on the model level using J-Link.

Overview of Model Objects

Models can be any PTC Creo Parametric file type, including parts, assemblies, drawings, sections, and notebook. The classes and methods in the package `com.ptc.pfc.pfcModel` provide generic access to models, regardless of their type. The available methods enable you to do the following:

- Access information about a model.
- Open, copy, rename, and save a model.

Getting a Model Object

Methods Introduced:

- **pfcFamily.FamilyTableRow.CreateInstance**
- **pfcSelect.Selection.GetSelModel**
- **pfcSession.BaseSession.GetModel**
- **pfcSession.BaseSession.GetCurrentModel**
- **pfcSession.BaseSession.GetActiveModel**
- **pfcSession.BaseSession.ListModels**
- **pfcSession.BaseSession.GetByRelationId**
- **pfcWindow.Window.GetModel**

These methods get a model object that is already in session.

The method `pfcSelect.Selection.GetSelModel` returns the model that was interactively selected.

The method `pfcSession.BaseSession.GetModel` returns a model based on its name and type, whereas `pfcSession.BaseSession.GetByRelationId` returns a model in an assembly that has the specified integer identifier.

The method `pfcSession.BaseSession.GetCurrentModel` returns the current active model.

The method `pfcSession.BaseSession.GetActiveModel` returns the active PTC Creo Parametric model.

Use the method `pfcSession.BaseSession.ListModels` to return a sequence of all the models in session.

For more methods that return solid models, refer to the chapter Solid on page 175.

Model Descriptors

Methods Introduced:

- **pfcModel.pfcModel.ModelDescriptor_Create**
- **pfcModel.pfcModel.ModelDescriptor_CreateFromFileName**
- **pfcModel.ModelDescriptor.SetGenericName**
- **pfcModel.ModelDescriptor.SetInstanceName**
- **pfcModel.ModelDescriptor.SetType**
- **pfcModel.ModelDescriptor.SetHost**
- **pfcModel.ModelDescriptor.SetDevice**
- **pfcModel.ModelDescriptor.SetPath**
- **pfcModel.ModelDescriptor.SetFileVersion**
- **pfcModel.ModelDescriptor.GetFullName**
- **pfcModel.Model.GetFullName**

Model descriptors are data objects used to describe a model file and its location in the system. The methods in the model descriptor enable you to set specific information that enables PTC Creo Parametric to find the specific model you want.

The static utility method `pfcModel.pfcModel.ModelDescriptor_Create` allows you to specify as data to be entered a model type, an instance name, and a generic name. The model descriptor constructs the full name of the model as a string, as follows:

```
String FullName = InstanceName+"<"+GenericName+">";
                                    // As long as the
                                    // generic name is
                                    // not an empty
                                    // string ("")
```

If you want to load a model that is not a family table instance, pass an empty string as the generic name argument so that the full name of the model is constructed correctly. If the model is a family table interface, you should specify both the instance and generic names.

Note

You are allowed to set other fields in the model descriptor object, but they may be ignored by some methods.

The static utility method `pfcModel.pfcModel.ModelDescriptor_CreateFromFileName` allows you to create a new model descriptor from a given a file name. The file name is a string in the form <name>.<extension>.

Retrieving Models

Methods Introduced:

- **pfcSession.BaseSession.RetrieveModel**
- **pfcSession.BaseSession.RetrieveModelWithOpts**
- **pfcSession.BaseSession.OpenFile**
- **pfcSolid.Solid.HasRetrievalErrors**

These methods cause PTC Creo Parametric to retrieve the model that corresponds to the `ModelDescriptor` argument.

The method `pfcSession.BaseSession.RetrieveModel` retrieves the specified model into the PTC Creo Parametric session given its model descriptor from a standard directory. This method ignores the path argument specified in the model descriptor. But this method does not create a window for it, nor does it display the model anywhere.

The method `pfcSession.BaseSession.RetrieveModelWithOpts` retrieves the specified model into the PTC Creo Parametric session based on the path specified by the model descriptor. The path can be a disk path, a workspace path, or a commonspace path. The *Opts* argument (given by the `Session.RetrieveModelOptions` object) provides the user with the option to specify simplified representations.

The method `pfcSession.BaseSession.OpenFile` brings the model into memory, opens a new window for it (or uses the base window, if it is empty), and displays the model.

Note

`pfcSession.BaseSession.OpenFile` actually returns a handle to the window it has created.

To get a handle to the model you need, use the method `pfcWindow.Window.GetModel`.

The method `pfcSolid.Solid.HasRetrievalErrors` returns a true value if the features in the solid model were suppressed during the RetrieveModel or OpenFile operations. This method must be called immediately after the `pfcSession.BaseSession.RetrieveModel` method or an equivalent retrieval method.

Model Information

Methods Introduced:

- **pfcModel.Model.GetFileName**
- **pfcModel.Model.GetCommonName**
- **pfcModel.Model.IsCommonNameModifiable**
- **pfcModel.Model.GetFullName**
- **pfcModel.Model.GetGenericName**
- **pfcModel.Model.GetInstanceName**
- **pfcModel.Model.GetOrigin**
- **pfcModel.Model.GetRelationId**
- **pfcModel.Model.GetDescr**
- **pfcModel.Model.GetType**
- **pfcModel.Model.GetIsModified**
- **pfcModel.Model.GetVersion**
- **pfcModel.Model.GetRevision**
- **pfcModel.Model.GetBranch**
- **pfcModel.Model.GetReleaseLevel**
- **pfcModel.Model.GetVersionStamp**
- **pfcModel.Model.ListDependencies**
- **pfcModel.Model.CleanupDependencies**
- **pfcModel.Model.ListDeclaredModels**
- **pfcModel.Model.CheckIsModifiable**
- **pfcModel.Model.CheckIsSaveAllowed**

The method `pfcModel.Model.GetFileName` retrieves the model file name in the "name"."type" format.

The method `pfcModel.Model.GetCommonName` retrieves the common name for the model. This name is displayed for the model in PTC Windchill PDMLink.

Use the method `pfcModel.Model.GetIsCommonNameModifiable` to identify if the common name of the model can be modified. You can modify the name for models that are not yet owned by PTC Windchill PDMLink, or in certain situations if the configuration option `let_proe_rename_pdm_objects` is set to `yes`.

The method `pfcModel.Model.GetFullName` retrieves the full name of the model in the instance <generic> format.

The method `pfcModel.Model.GetGenericName` retrieves the name of the generic model. If the model is not an instance, this name must be `NULL` or an empty string.

The method pfcModel.Model.GetInstanceName retrieves the name of the model. If the model is an instance, this method retrieves the instance name.

The method pfcModel.Model.GetOrigin returns the complete path to the file from which the model was opened. This path can be a location on disk from a PTC Windchill workspace, or from a downloaded URL.

The method pfcModel.Model.GetRelationId retrieves the relation identifier of the specified model. It can be NULL.

The method pfcModel.Model.GetDescr returns the descriptor for the specified model. Model descriptors can be used to represent models not currently in session.

Note

From Pro/ENGINEER Wildfire 4.0 onwards, the methodsproperties pfcModel.Model.GetFullName, pfcModel.Model.GetGenericName, and pfcModel.Model.GetDescr throw an exception pfcExceptions.XtoolkitCantOpen if called on a model instance whose immediate generic is not in session. Handle this exception and typecast the model as pfcSolid.Solid, which in turn can be typecast as pfcFamily.FamilyMember, and use the method pfcFamily.FamilyMember.GetImmediateGenericInfo to get the model descriptor of the immediate generic model. The model descriptor can be used to derive the full name or generic name of the model. If you wish to switch off this behavior and continue to run legacy applications in the pre-Wildfire 4.0 mode, set the configuration option retrieve_instance_dependencies to instance_and_generic_deps.

The method pfcModel.Model.GetType returns the type of model in the form of the pfcModel.ModelType object. The types of models are as follows:

- MDL_ASSEMBLY—Specifies an assembly.
- MDL_PART—Specifies a part.
- MDL_DRAWING—Specifies a drawing.
- MDL_2D_SECTION—Specifies a 2D section.
- MDL_LAYOUT—Specifies a notebook.
- MDL_DWG_FORMAT—Specifies a drawing format.
- MDL_MFG—Specifies a manufacturing model.
- MDL_REPORT—Specifies a report.
- MDL_MARKUP—Specifies a drawing markup.

- `MDL_DIAGRAM`—Specifies a diagram
- `MDL_CE_SOLID`—Specifies a Layout model.

Note

J-Link methods will only be able to read models of type Layout, but will not be able to pass Layout models as input to other methods. PTC recommends that you review all J-Link applications that use the object `pfcModel.ModelType` and modify the code as appropriate to ensure that the applications work correctly.

The method `pfcModel.Model.GetIsModified` identifies whether the model has been modified since it was last saved.

The method `pfcModel.Model.GetVersion` returns the version of the specified model from the PDM system. It can be `NULL`, if not set.

The method `pfcModel.Model.GetRevision` returns the revision number of the specified model from the PDM system. It can be `NULL`, if not set.

The method `pfcModel.Model.GetBranch` returns the branch of the specified model from the PDM system. It can be `NULL`, if not set.

The method `pfcModel.Model.GetReleaseLevel` returns the release level of the specified model from the PDM system. It can be `NULL`, if not set.

The method `pfcModel.Model.GetVersionStamp` returns the version stamp of the specified model. The version stamp is a PTC Creo Parametric specific identifier that changes with each change made to the model.

The method `pfcModel.Model.ListDependencies` returns a list of the first-level dependencies for the specified model in the PTC Creo Parametric workspace in the form of the `pfcModel.Dependencies` object.

Use the method `pfcModel.Model.CleanupDependencies` to clean the dependencies for an object in the PTC Creo Parametric workspace.

Note

Do not call the method `pfcModel.Model.CleanupDependencies` during operations that alter the dependencies, such as, restructuring components and creating or redefining features.

The method `pfcModel.Model.ListDeclaredModels` returns a list of all the first-level objects declared for the specified model.

The method `pfcModel.Model.CheckIsModifiable` identifies if a given model can be modified without checking for any subordinate models. This method takes a boolean argument *ShowUI* that determines whether the PTC Creo Parametric conflict resolution dialog box should be displayed to resolve conflicts, if detected. If this argument is false, then the conflict resolution dialog box is not displayed, and the model can be modified only if there are no conflicts that cannot be overridden, or are resolved by default resolution actions. For a generic model, if *ShowUI* is true, then all instances of the model are also checked.

The method `pfcModel.Model.CheckIsSaveAllowed` identifies if a given model can be saved along with all of its subordinate models. The subordinate models can be saved based on their modification status and the value of the configuration option `save_objects`. This method also checks the current user interface context to identify if it is currently safe to save the model. Thus, calling this method at different times might return different results. This method takes a boolean argument *ShowUI*. Refer to the previous method for more information on this argument.

Model Operations

Methods Introduced:

- **pfcModel.Model.Backup**
- **pfcModel.Model.Copy**
- **pfcModel.Model.CopyAndRetrieve**
- **pfcModel.Model.Rename**
- **pfcModel.Model.Save**
- **pfcModel.Model.Erase**
- **pfcModel.Model.EraseWithDependencies**
- **pfcModel.Model.Delete**
- **pfcModel.Model.Display**
- **pfcModel.Model.SetCommonName**

These model operations duplicate most of the commands available in the PTC Creo Parametric **File** menu.

The method `pfcModel.Model.Backup` makes a backup of an object in memory to a disk in a specified directory.

The method `pfcModel.Model.Copy` copies the specified model to another file.

The method `pfcModel.Model.CopyAndRetrieve` copies the model to another name, and retrieves that new model into session.

The method `pfcModel.Model.Rename` renames a specified model.

The method `pfcModel.Model.Save` stores the specified model to a disk.

The method `pfcModel.Model.Erase` erases the specified model from the session. Models used by other models cannot be erased until the models dependent upon them are erased.

The method `pfcModel.Model.EraseWithDependencies` erases the specified model from the session and all the models on which the specified model depends from disk, if the dependencies are not needed by other items in session.

Note

However, while erasing an active model, `pfcModel.Model.Erase` and `pfcModel.Model.EraseWithDependencies` only clear the graphic display immediately, they do not clear the data in the memory until the control returns to PTC Creo Parametric from the J-Link application. Therefore, after calling them the control must be returned to PTC Creo Parametric before calling any other function, otherwise the behavior of PTC Creo Parametric may be unpredictable.

The method `pfcModel.Model.Delete` removes the specified model from memory and disk.

The method `pfcModel.Model.Display` displays the specified model. You must call this method if you create a new window for a model because the model will not be displayed in the window until you call `pfcModel.Display`.

The method `pfcModel.Model.SetCommonName` modifies the common name of the specified model. You can modify this name for models that are not yet owned by PTC Windchill PDMLink, or in certain situations if the configuration option `let_proe_rename_pdm_objects` is set to `yes`.

Running PTC Creo ModelCHECK

PTC Creo Modelcheck is an integrated application that runs transparently within PTC Creo Parametric. PTC Creo Modelcheck uses a configurable list of company design standards and best modeling practices. You can configure PTC Creo Modelcheck to run interactively or automatically when you regenerate or save a model.

Methods Introduced:

- **pfcSession.BaseSession.ExecuteModelCheck**
- **pfcModelCheck.pfcModelCheck.ModelCheckInstructions_Create**
- **pfcModelCheck.ModelCheckInstructions.SetConfigDir**
- **pfcModelCheck.ModelCheckInstructions.SetMode**

- **pfcModelCheck.ModelCheckInstructions.SetOutputDir**
- **pfcModelCheck.ModelCheckInstructions.SetShowInBrowser**
- **pfcModelCheck.ModelCheckResults.GetNumberOfErrors**
- **pfcModelCheck.ModelCheckResults.GetNumberOfWarnings**
- **pfcModelCheck.ModelCheckResults.GetWasModelSaved**

You can run PTC Creo Modelcheck from an external application using the method `pfcSession.BaseSession.ExecuteModelCheck`. This method takes the model *Model* on which you want to run PTC Creo Modelcheck and instructions in the form of the object `ModelCheckInstructions` as its input parameters. This object contains the following parameters:

- `ConfigDir`—Specifies the location of the configuration files. If this parameter is set to `NULL`, the default PTC Creo Modelcheck configuration files are used.
- `Mode`—Specifies the mode in which you want to run PTC Creo Modelcheck. The modes are:
  - `MODELCHECK_GRAPHICS`—Interactive mode
  - `MODELCHECK_NO_GRAPHICS`—Batch mode
- `OutputDir`—Specifies the location for the reports. If you set this parameter to `NULL`, the default PTC Creo Modelcheck directory, as per `config_init.mc`, will be used.
- `ShowInBrowser`—Specifies if the results report should be displayed in the Web browser.

The method `pfcModelCheck.pfcModelCheck.ModelCheckInstructions_Create` creates the `ModelCheckInstructions` object containing the PTC Creo Modelcheck instructions described above.

Use the methods `pfcModelCheck.ModelCheckInstructions.SetConfigDir`, `pfcModelCheck.ModelCheckInstructions.SetMode`, `pfcModelCheck.ModelCheckInstructions.SetOutputDir`, and `pfcModelCheck.ModelCheckInstructions.SetShowInBrowser` to modify the PTC Creo Modelcheck instructions.

The method `pfcSession.BaseSession.ExecuteModelCheck` returns the results of the PTC Creo Modelcheck run in the form of the `ModelCheckResults` object. This object contains the following parameters:

- `NumberOfErrors`—Specifies the number of errors detected.
- `NumberOfWarnings`—Specifies the number of warnings found.
- `WasModelSaved`—Specifies whether the model is saved with updates.

Use the methods`pfcModelCheck.ModelCheckResults.GetNumberOfErrors`, `pfcModelCheck.ModelCheckResults.GetNumberOfWarning`, and `pfcModelCheck.ModelCheckResults.GetWasModelSaved` to access the results obtained.

11

Drawings

This chapter describes how to program drawing functions using J-Link.

Overview of Drawings in J-Link

This section describes the functions that deal with drawings. You can create drawings of all PTC Creo Parametric models using the functions in J-Link. You can annotate the drawing, manipulate dimensions, and use layers to manage the display of different items.

Unless otherwise specified, J-Link functions that operate on drawings use world units.

Creating Drawings from Templates

Drawing templates simplify the process of creating a drawing using J-Link. PTC Creo Parametric can create views, set the view display, create snap lines, and show the model dimensions based on the template. Use templates to:

- Define layout views
- Set view display
- Place notes
- Place symbols
- Define tables
- Show dimensions

Method Introduced:

- **pfcSession.BaseSession.CreateDrawingFromTemplate**

Use the method `pfcSession.BaseSession.CreateDrawingFromTemplate` to create a drawing from the drawing template and to return the created drawing. The attributes are:

- New drawing name
- Name of an existing template
- Name and type of the solid model to use while populating template views
- Sequence of options to create the drawing. The options are as follows:
  - `DRAWINGCREATE_DISPLAY_DRAWING`—display the new drawing.
  - `DRAWINGCREATE_SHOW_ERROR_DIALOG`—display the error dialog box.
  - `DRAWINGCREATE_WRITE_ERROR_FILE`—write the errors to a file.
  - `DRAWINGCREATE_PROMPT_UNKNOWN_PARAMS`—prompt the user on encountering unknown parameters

Drawing Creation Errors

Methods Introduced:

- **pfcExceptions.XToolkitDrawingCreateErrors.GetErrors**
- **pfcExceptions.DrawingCreateError.GetType**
- **pfcExceptions.DrawingCreateError.GetViewName**
- **pfcExceptions.DrawingCreateError.GetObjectName**
- **pfcExceptions.DrawingCreateError.GetSheetNumber**
- **pfcExceptions.DrawingCreateError.GetView**

The exception `XToolkitDrawingCreateErrors` is thrown if an error is encountered when creating a drawing from a template. This exception contains a list of errors which occurred during drawing creation.

> **Note**
>
> When this exception type is encountered, the drawing is actually created, but some of the contents failed to generate correctly.

The error structure contains an array of drawing creation errors. Each error message may have the following elements:

- *Type*—The type of error as follows:
  - `DWGCREATE_ERR_SAVED_VIEW_DOESNT_EXIST`—Saved view does not exist.
  - `DWGCREATE_ERR_X_SEC_DOESNT_EXIST`—Specified cross section does not exist.
  - `DWGCREATE_ERR_EXPLODE_DOESNT_EXIST`—Exploded state did not exist.
  - `DWGCREATE_ERR_MODEL_NOT_EXPLODABLE`—Model cannot be exploded.
  - `DWGCREATE_ERR_SEC_NOT_PERP`—Cross section view not perpendicular to the given view.
  - `DWGCREATE_ERR_NO_RPT_REGIONS`—Repeat regions not available.
  - `DWGCREATE_ERR_FIRST_REGION_USED`—Repeat region was unable to use the region specified.
  - `DWGCREATE_ERR_NOT_PROCESS_ASSEM`— Model is not a process assembly view.

- ○ DWGCREATE_ERR_NO_STEP_NUM—The process step number does not exist.
- ○ DWGCREATE_ERR_TEMPLATE_USED—The template does not exist.
- ○ DWGCREATE_ERR_NO_PARENT_VIEW_FOR_PROJ—There is no possible parent view for this projected view.
- ○ DWGCREATE_ERR_CANT_GET_PROJ_PARENT—Could not get the projected parent for a drawing view.
- ○ DWGCREATE_ERR_SEC_NOT_PARALLEL—The designated cross section was not parallel to the created view.
- ○ DWGCREATE_ERR_SIMP_REP_DOESNT_EXIST—The designated simplified representation does not exist.

- *ViewName*—Name of the view where the error occurred.
- *SheetNumber*—Sheet number where the error occurred.
- *ObjectName*—Name of the invalid or missing object.
- *View*—2D view in which the error occurred.

Use the method pfcExceptions.XToolkitDrawingCreateErrors.GetErrors to obtain the preceding array elements from the error object.

Example: Drawing Creation from a Template

The sample code in the file pfcDrawingExamples.java located at <creo_jlink_loadpoint>/jlink_appls/jlinkexamples creates a new drawing using a predefined template.

Obtaining Drawing Models

This section describes how to obtain drawing models.

Methods Introduced:

- **pfcSession.BaseSession.RetrieveModel**
- **pfcSession.BaseSession.GetModel**
- **pfcSession.BaseSession.GetModelFromDescr**
- **pfcSession.BaseSession.ListModels**
- **pfcSession.BaseSession.ListModelsByType**

The method pfcSession.BaseSession.RetrieveModel retrieves the drawing specified by the model descriptor. Model descriptors are data objects used to describe a model file and its location in the system. The method returns the retrieved drawing.

The method pfcSession.BaseSession.GetModel returns a drawing based on its name and type, whereas pfcSession.BaseSession.GetModelFromDescr returns a drawing specified by the model descriptor. The model must be in session.

Use the method pfcSession.BaseSession.ListModels to return a sequence of all the drawings in session.

Drawing Information

Methods Introduced:

- **pfcModel2D.Model2D.ListModels**
- **pfcModel2D.Model2D.GetCurrentSolid**
- **pfcModel2D.Model2D.ListSimplifiedReps**
- **pfcModel2D.Model2D.GetTextHeight**

The method pfcModel2D.Model2D.ListModels returns a list of all the solid models used in the drawing.

The method pfcModel2D.Model2D.GetCurrentSolid returns the current solid model of the drawing.

The method pfcModel2D.Model2D.ListSimplifiedReps returns the simplified representations of a solid model that are assigned to the drawing.

The method pfcModel2D.Model2D.GetTextHeight returns the text height of the drawing.

Drawing Operations

Methods Introduced:

- **pfcModel2D.Model2D.AddModel**
- **pfcModel2D.Model2D.DeleteModel**
- **pfcModel2D.Model2D.ReplaceModel**
- **pfcModel2D.Model2D.SetCurrentSolid**
- **pfcModel2D.Model2D.AddSimplifiedRep**
- **pfcModel2D.Model2D.DeleteSimplifiedRep**
- **pfcModel2D.Model2D.Regenerate**
- **pfcModel2D.Model2D.SetTextHeight**
- **pfcModel2D.Model2D.CreateDrawingDimension**
- **pfcModel2D.Model2D.CreateView**

The method pfcModel2D.Model2D.AddModel adds a new solid model to the drawing.

The method pfcModel2D.Model2D.DeleteModel removes a model from the drawing. The model to be deleted should not appear in any of the drawing views.

The method pfcModel2D.Model2D.ReplaceModel replaces a model in the drawing with a related model (the relationship should be by family table or interchange assembly). It allows you to replace models that are shown in drawing views and regenerates the view.

The method pfcModel2D.Model2D.SetCurrentSolid assigns the current solid model for the drawing. Before calling this method, the solid model must be assigned to the drawing using the method pfcModel2D.Model2D.AddModel. To see the changes to parameters and fields reflecting the change of the current solid model, regenerate the drawing using the method pfcSheet.SheetOwner.RegenerateSheet.

The method pfcModel2D.Model2D.AddSimplifiedRep associates the drawing with the simplified representation of an assembly

The method pfcModel2D.Model2D.DeleteSimplifiedRep removes the association of the drawing with an assembly simplified representation. The simplified representation to be deleted should not appear in any of the drawing views.

Use the method pfcModel2D.Model2D.Regenerate to regenerate the drawing draft entities and appearance.

The method pfcModel2D.Model2D.SetTextHeight ets the value of the text height of the drawing.

The method pfcModel2D.Model2D.CreateDrawingDimension creates a new drawing dimension based on the data object that contains information about the location of the dimension. This method returns the created dimension. Refer to the section Drawing Dimensions on page 135.

The method pfcModel2D.Model2D.CreateView creates a new drawing view based on the data object that contains information about how to create the view. The method returns the created drawing view. Refer to the section Creating Drawing Views on page 130.

Example: Replace Drawing Model Solid with its Generic

The sample code in the file pfcDrawingExamples.java located at <creo_jlink_loadpoint>/jlink_appls/jlinkexamples replaces all solid model instances in a drawing with its generic. Models are not replaced if the generic model is already present in the drawing.

Drawing Sheets

A drawing sheet is represented by its number. Drawing sheets in J-Link are identified by the same sheet numbers seen by a PTC Creo Parametric user.

Note

These identifiers may change if the sheets are moved as a consequence of adding, removing or reordering sheets.

Drawing Sheet Information

Methods Introduced

- **pfcSheet.SheetOwner.GetSheetTransform**
- **pfcSheet.SheetOwner.GetSheetInfo**
- **pfcSheet.SheetOwner.GetSheetScale**
- **pfcSheet.SheetOwner.GetSheetFormat**
- **pfcSheet.SheetOwner.GetSheetFormatDescr**
- **pfcSheet.SheetOwner.GetSheetBackgroundView**
- **pfcSheet.SheetOwner.GetNumberOfSheets**
- **pfcSheet.SheetOwner.GetCurrentSheetNumber**
- **pfcSheet.SheetOwner.GetSheetUnits**

Superseded Method:

- **pfcSheet.SheetOwner.GetSheetData**

The method `pfcSheet.SheetOwner.GetSheetTransform` returns the transformation matrix for the sheet specified by the sheet number. This transformation matrix includes the scaling needed to convert screen coordinates to drawing coordinates (which use the designated drawing units).

The method `pfcSheet.SheetOwner.GetSheetInfo` returns sheet data including the size, orientation, and units of the sheet specified by the sheet number.

The method `pfcSheet.SheetOwner.GetSheetData` and the `pfcSheet.SheetData` have been deprecated. Use the method `pfcSheet.SheetOwner.GetSheetInfo` and the `pfcSheet.SheetInfo` instead.

The method pfcSheet.SheetOwner.GetSheetScale returns the scale of the drawing on a particular sheet based on the drawing model used to measure the scale. If no models are used in the drawing then the default scale value is 1.0.

The method pfcSheet.SheetOwner.GetSheetFormat returns the drawing format used for the sheet specified by the sheet number. It returns a null value if no format is assigned to the sheet.

The method pfcSheet.SheetOwner.GetSheetFormatDescr returns the model descriptor of the drawing format used for the specified drawing sheet.

The method pfcSheet.SheetOwner.GetSheetBackgroundView returns the view object representing the background view of the sheet specified by the sheet number.

The method pfcSheet.SheetOwner.GetNumberOfSheets returns the number of sheets in the model.

The method pfcSheet.SheetOwner.GetCurrentSheetNumber returns the current sheet number in the model.

Note

The sheet numbers range from 1 to n, where n is the number of sheets.

The method pfcSheet.SheetOwner.GetSheetUnits returns the units used by the sheet specified by the sheet number.

Drawing Sheet Operations

Methods Introduced:

- **pfcSheet.SheetOwner.AddSheet**
- **pfcSheet.SheetOwner.DeleteSheet**
- **pfcSheet.SheetOwner.ReorderSheet**
- **pfcSheet.SheetOwner.RegenerateSheet**
- **pfcSheet.SheetOwner.SetSheetScale**
- **pfcSheet.SheetOwner.SetSheetFormat**
- **pfcSheet.SheetOwner.SetCurrentSheetNumber**

The method pfcSheet.SheetOwner.AddSheet adds a new sheet to the model and returns the number of the new sheet.

The method pfcSheet.SheetOwner.DeleteSheet removes the sheet specified by the sheet number from the model.

Use the method pfcSheet.SheetOwner.ReorderSheet to reorder the sheet from a specified sheet number to a new sheet number.

Note

The sheet number of other affected sheets also changes due to reordering or deletion.

The method pfcSheet.SheetOwner.RegenerateSheet regenerates the sheet specified by the sheet number.

Note

You can regenerate a sheet only if it is displayed.

Use the method pfcSheet.SheetOwner.SetSheetScale to set the scale of a model on the sheet based on the drawing model to scale and the scale to be used. Pass the value of the *DrawingModel* parameter as null to select the current drawing model.

Use the method pfcSheet.SheetOwner.SetSheetFormat to apply the specified format to a drawing sheet based on the drawing format, sheet number of the format, and the drawing model.

The sheet number of the format is specified by the *FormatSheetNumber* parameter. This number ranges from 1 to the number of sheets in the format. Pass the value of this parameter as null to use the first format sheet.

The drawing model is specified by the *DrawingModel* parameter. Pass the value of this parameter as null to select the current drawing model.

The method pfcSheet.SheetOwner.SetCurrentSheetNumber sets the current sheet to the sheet number specified.

Example: Listing Drawing Sheets

The sample code in the file pfcDrawingExamples.java located at <creo_jlink_loadpoint>/jlink_appls/jlinkexamples shows how to list the sheets in the current drawing. The information is placed in an external browser window.

Drawing Views

A drawing view is represented by the pfcView2D.View2D. All model views in the drawing are associative, that is, if you change a dimensional value in one view, the system updates other drawing views accordingly. The model automatically reflects any dimensional changes that you make to a drawing. In addition, corresponding drawings also reflect any changes that you make to a model such as the addition or deletion of features and dimensional changes.

Creating Drawing Views

Method Introduced:

- **pfcModel2D.Model2D.CreateView**

The method pfcModel2D.Model2D.CreateView reates a new view in the drawing. Before calling this method, the drawing must be displayed in a window.

The pfcView2D.View2DCreateInstructions contains details on how to create the view. The types of drawing views supported for creation are:

- DRAWVIEW_GENERAL—General drawing views
- DRAWVIEW_PROJECTION—Projected drawing views

General Drawing Views

The pfcView2D.GeneralViewCreateInstructions contains details on how to create general drawing views.

Methods Introduced:

- **pfcView2D.pfcView2D.GeneralViewCreateInstructions_Create**
- **pfcView2D.GeneralViewCreateInstructions.SetViewModel**
- **pfcView2D.GeneralViewCreateInstructions.SetLocation**
- **pfcView2D.GeneralViewCreateInstructions.SetSheetNumber**
- **pfcView2D.GeneralViewCreateInstructions.SetOrientation**
- **pfcView2D.GeneralViewCreateInstructions.SetExploded**
- **pfcView2D.GeneralViewCreateInstructions.SetScale**

The method pfcView2D.pfcView2D.GeneralViewCreateInstructions_Create creates the pfcView2D.GeneralViewCreateInstructions ata object used for creating general drawing views.

Use the method pfcView2D.GeneralViewCreateInstructions.SetViewModel to assign the solid model to display in the created general drawing view.

Use the method `pfcView2D.GeneralViewCreateInstructions.SetLocation` to assign the location in a drawing sheet to place the created general drawing view.

Use the method `pfcView2D.GeneralViewCreateInstructions.SetSheetNumber` to set the number of the drawing sheet in which the general drawing view is created.

The method `pfcView2D.GeneralViewCreateInstructions.SetOrientation` assigns the orientation of the model in the general drawing view in the form of the `pfcBase.Transform3D` data object. The transformation matrix must only consist of the rotation to be applied to the model. It must not consist of any displacement or scale components. If necessary, set the displacement to {0, 0, 0} using the method `pfcBase.Transform3D.SetOrigin`, and remove any scaling factor by normalizing the matrix.

Use the method `pfcView2D.GeneralViewCreateInstructions.SetExploded` to set the created general drawing view to be an exploded view.

Use the method `pfcView2D.GeneralViewCreateInstructions.SetScale` to assign a scale to the created general drawing view. This value is optional, if not assigned, the default drawing scale is used.

Projected Drawing Views

The `pfcView2D.ProjectionViewCreateInstructions` contains details on how to create general drawing views.

Methods Introduced:

- **pfcView2D.pfcView2D.ProjectionViewCreateInstructions_Create**
- **pfcView2D.ProjectionViewCreateInstructions.SetParentView**
- **pfcView2D.ProjectionViewCreateInstructions.SetLocation**
- **pfcView2D.ProjectionViewCreateInstructions.SetExploded**

The method `pfcView2D.pfcView2D.ProjectionViewCreateInstructions_Create` creates the `pfcView2D.ProjectionViewCreateInstructions` data object used for creating projected drawing views.

Use the method `pfcView2D.ProjectionViewCreateInstructions.SetParentView` to assign the parent view for the projected drawing view.

Use the method `pfcView2D.ProjectionViewCreateInstructions.SetLocation` to assign the location of the projected drawing view. This location determines how the drawing view will be oriented.

Use the method `pfcView2D.ProjectionViewCreateInstructions.SetExploded` to set the created projected drawing view to be an exploded view.

Example: Creating Drawing Views

The sample code in the file `pfcDrawingExamples.java` located at <creo_jlink_loadpoint>/jlink_appls/jlinkexamples adds a new sheet to a drawing and creates three views of a selected model.

Obtaining Drawing Views

Methods Introduced:

- **pfcSelect.Selection.GetSelView2D**
- **pfcModel2D.Model2D.List2DViews**
- **pfcModel2D.Model2D.GetViewByName**
- **pfcModel2D.Model2D.GetViewDisplaying**
- **pfcSheet.SheetOwner.GetSheetBackgroundView**

The method `pfcSelection.Selection.GetSelView2D` returns the selected drawing view (if the user selected an item from a drawing view). It returns a null value if the selection does not contain a drawing view.

The method `pfcModel2D.Model2D.List2DViews` lists and returns the drawing views found. This method does not include the drawing sheet background views returned by the method `pfcSheet.SheetOwner.GetSheetBackgroundView`.

The method `pfcModel2D.Model2D.GetViewByName` returns the drawing view based on the name. This method returns a null value if the specified view does not exist.

The method `pfcModel2D.Model2D.GetViewDisplaying` returns the drawing view that displays a dimension. This method returns a null value if the dimension is not displayed in the drawing.

Note

This method works for solid and drawing dimensions.

The method pfcSheet.SheetOwner.GetSheetBackgroundView returns the drawing sheet background views.

Drawing View Information

Methods Introduced:

- **pfcObject.Child.GetDBParent**
- **pfcView2D.View2D.GetSheetNumber**
- **pfcView2D.View2D.GetIsBackground**
- **pfcView2D.View2D.GetModel**
- **pfcView2D.View2D.GetScale**
- **pfcView2D.View2D.GetIsScaleUserdefined**
- **pfcView2D.View2D.GetOutline**
- **pfcView2D.View2D.GetLayerDisplayStatus**
- **pfcView2D.View2D.GetIsViewdisplayLayerDependent**
- **pfcView2D.View2D.GetDisplay**
- **pfcView2D.View2D.GetTransform**
- **pfcView2D.View2D.GetName**
- **pfcView2D.View2D.GetSimpRep**

The inherited method pfcObject.Child.GetDBParent, when called on a View2D object, provides the drawing model which owns the specified drawing view. The return value of the method can be downcast to a Model2D object.

The method pfcView2D.View2D.GetSheetNumber returns the sheet number of the sheet that contains the drawing view.

The method pfcView2D.View2D.GetIsBackground returns a value that indicates whether the view is a background view or a model view.

The method pfcView2D.View2D.GetModel returns the solid model displayed in the drawing view.

The method pfcView2D.View2D.GetScale returns the scale of the drawing view.

The method pfcView2D.View2D.GetIsScaleUserdefined specifies if the drawing has a user-defined scale.

The method pfcView2D.View2D.GetOutline returns the position of the view in the sheet in world units.

The method pfcView2D.View2D.GetLayerDisplayStatus returns the display status of the specified layer in the drawing view.

The method `pfcView2D.View2D.GetDisplay` returns an output structure that describes the display settings of the drawing view. The fields in the structure are as follows:

- *Style*—Whether to display as wireframe, hidden lines, no hidden lines, or shaded
- *TangentStyle*—Linestyle used for tangent edges
- *CableStyle*—Linestyle used to display cables
- *RemoveQuiltHiddenLines*—Whether or not to apply hidden-line-removal to quilts
- *ShowConceptModel*—Whether or not to display the skeleton
- *ShowWeldXSection*—Whether or not to include welds in the cross-section

The method `pfcView2D.View2D.GetTransform` returns a matrix that describes the transform between 3D solid coordinates and 2D world units for that drawing view. The transformation matrix is a combination of the following factors:

- The location of the view origin with respect to the drawing origin.
- The scale of the view units with respect to the drawing units
- The rotation of the model with respect to the drawing coordinate system.

The method `pfcView2D.View2D.GetName` returns the name of the specified view in the drawing.

The simplified representations of assembly and part can be used as drawing models to create general views. Use the method `pfcView2D.View2D.GetSimpRep` to retrieve the simplified representation for the specified view in the drawing.

Example: Listing the Views in a Drawing

The sample code in the file `pfcDrawingExamples.java` located at `<creo_jlink_loadpoint>/jlink_appls/jlinkexamples` creates an information window about all the views in a drawing. The information is placed in an external browser window.

Drawing Views Operations

Methods Introduced:

- **pfcView2D.View2D.SetScale**
- **pfcView2D.View2D.Translate**
- **pfcView2D.View2D.Delete**
- **pfcView2D.View2D.Regenerate**

- **pfcView2D.View2D.SetLayerDisplayStatus**
- **pfcView2D.View2D.SetDisplay**

The method `pfcView2D.View2D.SetScale` sets the scale of the drawing view.

The method `pfcView2D.View2D.Translate` moves the drawing view by the specified transformation vector.

The method `pfcView2D.View2D.Delete` deletes a specified drawing view. Set the *DeleteChildren* parameter to `true` to delete the children of the view. Set this parameter to false or null to prevent deletion of the view if it has children.

The method `pfcView2D.View2D.Regenerate` erases the displayed view of the current object, regenerates the view from the current drawing, and redisplays the view.

The method `pfcView2D.View2D.SetLayerDisplayStatus` sets the display status for the layer in the drawing view.

The method `pfcView2D.View2D.SetDisplay` sets the value of the display settings for the drawing view.

Drawing Dimensions

This section describes the J-Link methods that give access to the types of dimensions that can be created in the drawing mode. They do not apply to dimensions created in the solid mode, either those created automatically as a result of feature creation, or reference dimension created in a solid. A drawing dimension or a reference dimension shown in a drawing is represented by the `com.ptc.pfc.pfcDimension2D.Dimension2D`.

Obtaining Drawing Dimensions

Methods Introduced:

- **pfcModelItem.ModelItemOwner.ListItems**
- **pfcModelItem.ModelItemOwner.GetItemById**
- **pfcSelect.Selection.GetSelItem**

The method `pfcModelItem.ModelItemOwner.ListItems` returns a list of drawing dimensions specified by the parameter *Type* or returns null if no drawing dimensions of the specified type are found. This method lists only those dimensions created in the drawing.

The values of the parameter *Type* for the drawing dimensions are:

- `ITEM_DIMENSION`—Dimension
- `ITEM_REF_DIMENSION`—Reference dimension

Set the parameter *Type* to the type of drawing dimension to retrieve. If this parameter is set to null, then all the dimensions in the drawing are listed.

The method `pfcModelItem.ModelItemOwner.GetItemById` returns a drawing dimension based on the type and the integer identifier. The method returns only those dimensions created in the drawing. It returns a null if a drawing dimension with the specified attributes is not found.

The method `pfcSelect.Selection.GetSelItem` returns the value of the selected drawing dimension.

Creating Drawing Dimensions

Methods Introduced:

- **pfcDimension2D.pfcDimension2D.DrawingDimCreateInstructions_Create**
- **pfcModel2D.Model2D.CreateDrawingDimension**
- **pfcDimension2D.pfcDimension2D.EmptyDimensionSense_Create**
- **pfcDimension2D.pfcDimension2D.PointDimensionSense_Create**
- **pfcDimension2D.pfcDimension2D.SplinePointDimensionSense_Create**
- **pfcDimension2D.pfcDimension2D.TangentIndexDimensionSense_Create**
- **pfcDimension2D.pfcDimension2D.LinAOCTangentDimensionSense_Create**
- **pfcDimension2D.pfcDimension2D.AngleDimensionSense_Create**
- **pfcDimension2D.pfcDimension2D.PointToAngleDimensionSense_Create**

The method`pfcDimension2D.pfcDimension2D.DrawingDimCreateInstructions_Create` creates an instructions object that describes how to create a drawing dimension using the method `pfcModel2D.Model2D.CreateDrawingDimension`.

The parameters of the instruction object are:

- *Attachments*—The entities that the dimension is attached to. The selections should include the drawing model view.
- *IsRefDimension*—True if the dimension is a reference dimension, otherwise null or false.
- *OrientationHint*—Describes the orientation of the dimensions in cases where this cannot be deduced from the attachments themselves.

- *Senses*—Gives more information about how the dimension attaches to the entity, i.e., to what part of the entity and in what direction the dimension runs. The types of dimension senses are as follows:
  - DIMSENSE_NONE
  - DIMSENSE_POINT
  - DIMSENSE_SPLINE_PT
  - DIMSENSE_TANGENT_INDEX
  - DIMSENSE_LINEAR_TO_ARC_OR_CIRCLE_TANGENT
  - DIMSENSE_ANGLE
  - DIMSENSE_POINT_TO_ANGLE
- *TextLocation*—The location of the dimension text, in world units.

The method `pfcModel2D.Model2D.CreateDrawingDimension` creates a dimension in the drawing based on the instructions data object that contains information needed to place the dimension. It takes as input an array of `pfcSelection` objects and an array of `pfcDimensionSense` structures that describe the required attachments. The method returns the created drawing dimension.

The method `pfcDimension2D.pfcDimension2D.EmptyDimensionSense_Create` creates a new dimension sense associated with the type `DIMSENSE NONE`. The `sense` field is set to *Type* In this case no information such as location or direction is needed to describe the attachment points. For example, if there is a single attachment which is a straight line, the dimension is the length of the straight line. If the attachments are two parallel lines, the dimension is the distance between them.

The method `pfcDimension2D.pfcDimension2D.PointDimensionSense_Create` creates a new dimension sense associated with the type `DIMSENSE POINT` which specifies the part of the entity to which the dimension is attached. The `sense` field is set to the value of the parameter *PointType*.

The possible values of *PointType* are:

- `DIMPOINT_END1`— The first end of the entity
- `DIMPOINT_END2`—The second end of the entity
- `DIMPOINT_CENTER`—The center of an arc or circle
- `DIMPOINT_NONE`—No information such as location or direction of the attachment is specified. This is similar to setting the *PointType* to `DIMSENSE NONE`.
- `DIMPOINT_MIDPOINT`—The mid point of the entity

The method pfcDimension2D.pfcDimension2D.SplinePointDimensionSense_Create creates a dimension sense associated with the type DIMSENSE_SPLINE_PT. This means that the attachment is to a point on a spline. The sense field is set to *SplinePointIndex* i.e., the index of the spline point.

The method pfcDimension2D.pfcDimension2D.TangentIndexDimensionSense_Create creates a new dimension sense associated with the type DIMSENSE_TANGENT_INDEX. The attachment is to a tangent of the entity, which is an arc or a circle. The sense field is set to *TangentIndex*, i.e., the index of the tangent of the entity.

The method pfcDimension2D.pfcDimension2D.LinAOCTangentDimensionSense_Create creates a new dimension sense associated with the type DIMSENSE_LINEAR_TO_ARC_OR_CIRCLE_TANGENT. The dimension is the perpendicular distance between the a line and a tangent to an arc or a circle that is parallel to the line. The sense field is set to the value of the parameter *TangentType*.

The possible values of *TangentType* are:

- DIMLINAOCTANGENT_LEFT0—The tangent is to the left of the line, and is on the same side, of the center of the arc or circle, as the line.
- DIMLINAOCTANGENT_RIGHT0—The tangent is to the right of the line, and is on the same side, of the center of the arc or circle, as the line.
- DIMLINAOCTANGENT_LEFT1—The tangent is to the left of the line, and is on the opposite side of the line.
- DIMLINAOCTANGENT_RIGHT1— The tangent is to the right of the line, and is on the opposite side of the line.

The method pfcDimension2D.pfcDimension2D.AngleDimensionSense_Create creates a new dimension sense associated with the type DIMSENSE_ANGLE. The dimension is the angle between two straight entities. The sense field is set to the value of the parameter *AngleOptions*.

The possible values of *AngleOptions* are:

- IsFirst—Is set to TRUE if the angle dimension starts from the specified entity in a counterclockwise direction. Is set to FALSE if the dimension ends at the specified entity. The value is TRUE for one entity and FALSE for the other entity forming the angle.
- ShouldFlip—If the value of ShouldFlip is FALSE, and the direction of the specified entity is away from the vertex of the angle, then the dimension

attaches directly to the entity. If the direction of the entity is away from the vertex of the angle, then the dimension is attached to the a witness line. The witness line is in line with the entity but in the direction opposite to the vertex of the angle. If the value of `ShouldFlip` is `TRUE` then the above cases are reversed.

The method `pfcDimension2D.pfcDimension2D.PointToAngleDimensionSense_Create` creates a new dimension sense associated with the type `DIMSENSE_POINT_TO_ANGLE`. The dimension is the angle between a line entity and the tangent to a curved entity. The curve attachment is of the type `DIMSENSE_POINT_TO_ANGLE` and the line attachment is of the type `DIMSENSE_POINT`. In this case both the `angle` and the `angle_sense` fields must be set. The field `sense` shows which end of the curve the dimension is attached to and the field `angle_sense` shows the direction in which the dimension rotates and to which side of the tangent it attaches.

Drawing Dimensions Information

Methods Introduced:

- **pfcDimension2D.Dimension2D.GetIsAssociative**
- **pfcDimension2D.Dimension2D.GetIsReference**
- **pfcDimension2D.Dimension2D.GetIsDisplayed**
- **pfcDimension2D.Dimension2D.GetAttachmentPoints**
- **pfcDimension2D.Dimension2D.GetDimensionSenses**
- **pfcDimension2D.Dimension2D.GetOrientationHint**
- **pfcDimension2D.Dimension2D.GetBaselineDimension**
- **pfcDimension2D.Dimension2D.GetLocation**
- **pfcDimension2D.Dimension2D.GetView**
- **pfcDimension2D.Dimension2D.GetTolerance**
- **pfcDimension2D.Dimension2D.GetIsToleranceDisplayed**

The method `pfcDimension2D.Dimension2D.GetIsAssociative` returns whether the dimension or reference dimension in a drawing is associative.

The method `pfcDimension2D.Dimension2D.GetIsReference` determines whether the drawing dimension is a reference dimension.

The method `pfcDimension2D.Dimension2D.GetIsDisplayed` determines whether the dimension will be displayed in the drawing.

The method `pfcDimension2D.Dimension2D.GetAttachmentPoints` returns a sequence of attachment points. The dimension senses array returned by the method `pfcDimension2D.Dimension2D.GetDimensionSenses` gives more information on how these attachments are interpreted.

The method `pfcDimension2D.Dimension2D.GetDimensionSenses` returns a sequence of dimension senses, describing how the dimension is attached to each attachment returned by the method `pfcDimension2D.Dimension2D.GetAttachmentPoints`.

The method `pfcDimension2D.Dimension2D.GetOrientationHint` returns the orientation hint for placing the drawing dimensions. The orientation hint determines how PTC Creo Parametric will orient the dimension with respect to the attachment points.

Note

This methods described above are applicable only for dimensions created in the drawing mode. It does not support dimensions created at intersection points of entities.

The method `pfcDimension2D.Dimension2D.GetBaselineDimension` returns an ordinate baseline drawing dimension. It returns a null value if the dimension is not an ordinate dimension.

Note

The method updates the display of the dimension only if it is currently displayed.

The method `pfcDimension2D.Dimension2D.GetLocation` returns the placement location of the dimension.

The method `pfcDimension2D.Dimension2D.GetView` returns the drawing view in which the dimension is displayed. This method applies to dimensions stored in the solid or in the drawing.

The method `pfcDimension2D.Dimension2D.GetTolerance` retrieves the upper and lower tolerance limits of the drawing dimension in the form of the `DimTolerance` object. A null value indicates a nominal tolerance.

Use the method `pfcDimension2D.Dimension2D.GetIsToleranceDisplayed` determines whether or not the dimension's tolerance is displayed in the drawing.

Drawing Dimensions Operations

Methods Introduced:

- **pfcDimension2D.Dimension2D.ConvertToLinear**
- **pfcDimension2D.Dimension2D.ConvertToOrdinate**
- **pfcDimension2D.Dimension2D.ConvertToBaseline**
- **pfcDimension2D.Dimension2D.SetLocation**
- **pfcDimension2D.Dimension2D.SwitchView**
- **pfcDimension2D.Dimension2D.SetTolerance**
- **pfcDimension2D.Dimension2D.EraseFromModel2D**
- **pfcModel2D.Model2D.SetViewDisplaying**

The method `pfcDimension2D.Dimension2D.ConvertToLinear` converts an ordinate drawing dimension to a linear drawing dimension. The drawing containing the dimension must be displayed.

The method `pfcDimension2D.Dimension2D.ConvertToOrdinate` converts a linear drawing dimension to an ordinate baseline dimension.

The method `pfcDimension2D.Dimension2D.ConvertToBaseline` converts a location on a linear drawing dimension to an ordinate baseline dimension. The method returns the newly created baseline dimension.

Note

The method updates the display of the dimension only if it is currently displayed.

The method `pfcDimension2D.Dimension2D.SetLocation` sets the placement location of a dimension or reference dimension in a drawing.

The method `pfcDimension2D.Dimension2D.SwitchView` changes the view where a dimension created in the drawing is displayed.

The method `pfcDimension2D.Dimension2D.SetTolerance` assigns the upper and lower tolerance limits of the drawing dimension.

The method `pfcDimension2D.Dimension2D.EraseFromModel2D` permanently erases the dimension from the drawing.

The method `pfcModel2D.Model2D.SetViewDisplaying` changes the view where a dimension created in a solid model is displayed.

Example: Command Creation of Dimensions from Model Datum Points

The sample code in the file `pfcDrawingExamples.java` located at <creo_jlink_loadpoint>/jlink_appls/jlinkexamples shows a command which creates vertical and horizontal ordinate dimensions from each datum point in a model in a drawing view to a selected coordinate system datum.

Drawing Tables

A drawing table in J-Link is represented by the `com.ptc.pfc.pfcTable.Table`. It is a child of the `ModelItem` .

Some drawing table methods operate on specific rows or columns. The row and column numbers in J-Link begin with 1 and range up to the total number of rows or columns in the table. Some drawing table methods operate on specific table cells. The `com.ptc.pfc.pfcTable.TableCell` is used to represent a drawing table cell.

Creating Drawing Cells

Method Introduced:

- **pfcTable.pfcTable.TableCell_Create**

The method `pfcTable.pfcTable.TableCell_Create` creates the `TableCell` object representing a cell in the drawing table.

Some drawing table methods operate on specific drawing segment. A multisegmented drawing table contains 2 or more areas in the drawing. Inserting or deleting rows in one segment of the table can affect the contents of other segments. Table segments are numbered beginning with 0. If the table has only a single segment, use 0 as the segment id in the relevant methods.

Selecting Drawing Tables and Cells

Methods Introduced:

- **pfcSession.BaseSession.Select**
- **pfcSelect.Selection.GetSelItem**
- **pfcSelect.Selection.GetSelTableCell**
- **pfcSelect.Selection.GetSelTableSegment**

Tables may be selected using the method `pfcSession.BaseSession.Select`. Pass the filter `dwg_table` to select an entire table and the filter `table_cell` to prompt the user to select a particular table cell.

The method pfcSelect.Selection.GetSelItem returns the selected table handle. It is a model item that can be cast to a Table object.

The method pfcSelect.Selection.GetSelTableCell returns the row and column indices of the selected table cell.

The method pfcSelect.Selection.GetSelTableSegment returns the table segment identifier for the selected table cell. If the table consists of a single segment, this method returns the identifier 0.

Creating Drawing Tables

Methods Introduced:

- **pfcTable.pfcTable.TableCreateInstructions_Create**
- **pfcTable.TableOwner.CreateTable**

The method pfcTable.pfcTable.TableCreateInstructions_Create creates the TableCreateInstructions data object that describes how to construct a new table using the method pfcTable.TableOwner.CreateTable.

The parameters of the instructions data object are:

- *Origin*—This parameter stores a three dimensional point specifying the location of the table origin. The origin is the position of the top left corner of the table.
- *RowHeights*—Specifies the height of each row of the table.
- *ColumnData*—Specifies the width of each column of the table and its justification.
- *SizeTypes*—Indicates the scale used to measure the column width and row height of the table.

The method pfcTable.TableOwner.CreateTable creates a table in the drawing specified by the TableCreateInstructions data object.

Retrieving Drawing Tables

Methods Introduced

- **pfcTable.pfcTable.TableRetrieveInstructions_Create**
- **pfcTable.TableRetrieveInstructions.SetFileName**
- **pfcTable.TableRetrieveInstructions.SetPath**
- **pfcTable.TableRetrieveInstructions.SetVersion**
- **pfcTable.TableRetrieveInstructions.SetPosition**
- **pfcTable.TableRetrieveInstructions.SetReferenceSolid**

- **pfcTable.TableRetrieveInstructions.SetReferenceRep**
- **pfcTable.TableOwner.RetrieveTable**
- **pfcTable.TableOwner.RetrieveTableByOrigin**

The method `pfcTable.TableOwner.RetrieveTable` retrieves a table specified by the `TableRetrieveInstructions` data object from a file on the disk. It returns the retrieved table. The data object contains information on the table to retrieve and is returned by the method `pfcTable.pfcTable.TableRetrieveInstructions_Create`.

The method `pfcTable.pfcTable.TableRetrieveInstructions_Create` creates the `TableRetrieveInstructions` data object that describes how to retrieve a drawing table using the methods `pfcTable.TableOwner.RetrieveTable` and `pfcTable.TableOwner.RetrieveTableByOrigin`. The method returns the created instructions data object.

The parameters of the instruction object are:

- *FileName*—Name of the file containing the drawing table.
- *Position*—Coordinates of the point on the drawing sheet, where the retrieved table must be placed. You must specify the value in screen coordinates.

You can also set the parameters for `TableRetrieveInstructions` data object using the following method:

- `pfcTable.TableRetrieveInstructions.SetFileName`—Sets the name of the drawing table. You must not specify the extension.
- `pfcTable.TableRetrieveInstructions.SetPath`—Sets the path to the drawing table file. The path must be specified relative to the working directory.
- `pfcTable.TableRetrieveInstructions.SetVersion`—Sets the version of the drawing table that must be retrieved. If you specify `NULL` rthe latest version of the drawing table is retrieved.
- `pfcTable.TableRetrieveInstructions.SetPosition`—Sets the coordinates of the point on the drawing sheet, where the table must be placed. You must specify the value in screen coordinates.
- `pfcTable.TableRetrieveInstructions.SetReferenceSolid`—Sets the model from which data must be copied into the drawing table. If this argument is passed as `NULL`, an empty table is created.
- `pfcTable.TableRetrieveInstructions.SetReferenceRep`—Sets the handle to the simplified representation in a solid, from which data must be copied into the drawing table. If this argument is passed as `NULL`, and the argument *solid* is not `NULL`, then data from the solid model is copied into the drawing table

The method pfcTable.TableOwner.RetrieveTable retrieves a table specified by the TableRetrieveInstructions data object from a file on the disk. It returns the retrieved table. The upper-left corner of the table is placed on the drawing sheet at the position specified by the pfcTableRetrieveInstructions data object.

The method pfcTable.TableOwner.RetrieveTableByOrigin also retrieves a table specified by the TableRetrieveInstructions data object from a file on the disk. The origin of the table is placed on the drawing sheet at the position specified by the TableRetrieveInstructions data object. Tables can be created with different origins by specifying the option **Direction**, in the **Insert Table** dialog box.

Drawing Tables Information

Methods Introduced:

- **pfcTable.TableOwner.ListTables**
- **pfcTable.TableOwner.GetTable**
- **pfcTable.Table.GetRowCount**
- **pfcTable.Table.GetColumnCount**
- **pfcTable.Table.CheckIfIsFromFormat**
- **pfcTable.Table.GetRowSize**
- **pfcTable.Table.GetColumnSize**
- **pfcTable.Table.GetText**
- **pfcTable.Table.GetCellNote**

The method pfcTable.TableOwner.ListTables returns a sequence of tables found in the model.

The method pfcTable.TableOwner.GetTable returns a table specified by the table identifier in the model. It returns a null value if the table is not found.

The method pfcTable.Table.GetRowCount returns the number of rows in the table.

The method pfcTable.Table.GetColumnCount returns the number of columns in the table.

The method pfcTable.Table.CheckIfIsFromFormat checks if the drawing table was created using the format. The method returns a true value if the table was created by applying the drawing format.

The method pfcTable.Table.GetRowSize returns the height of the drawing table row specified by the segment identifier and the row number.

The method `pfcTable.Table.GetColumnSize` returns the width of the drawing table column specified by the segment identifier and the column number.

The method `pfcTable.Table.GetText` returns the sequence of text in a drawing table cell. Set the value of the parameter *Mode* to `DWGTABLE_NORMAL` to get the text as displayed on the screen. Set it to `DWGTABLE_FULL` to get symbolic text, which includes the names of parameter references in the table text.

The method `pfcTable.Table.GetCellNote` returns the detail note item contained in the table cell.

Drawing Tables Operations

Methods Introduced:

- **pfcTable.Table.Erase**
- **pfcTable.Table.Display**
- **pfcTable.Table.RotateClockwise**
- **pfcTable.Table.InsertRow**
- **pfcTable.Table.InsertColumn**
- **pfcTable.Table.MergeRegion**
- **pfcTable.Table.SubdivideRegion**
- **pfcTable.Table.DeleteRow**
- **pfcTable.Table.DeleteColumn**
- **pfcTable.Table.SetText**
- **pfcTable.TableOwner.DeleteTable**

The method `pfcTable.Table.Erase` erases the specified table temporarily from the display. It still exists in the drawing. The erased table can be displayed again using the method `pfcTable.Table.Display`. The table will also be redisplayed by a window repaint or a regeneration of the drawing. Use these methods to hide a table from the display while you are making multiple changes to the table.

The method `pfcTable.Table.RotateClockwise` rotates a table clockwise by the specified amount of rotation.

The method `pfcTable.Table.InsertRow` inserts a new row in the drawing table. Set the value of the parameter *RowHeight* to specify the height of the row. Set the value of the parameter *InsertAfterRow* to specify the row number after which the new row has to be inserted. Specify 0 to insert a new first row.

The method `pfcTable.Table.InsertColumn` inserts a new column in the drawing table. Set the value of the parameter *ColumnWidth* to specify the width of the column. Set the value of the parameter *InsertAfterColumn* to specify the column number after which the new column has to be inserted. Specify 0 to insert a new first column.

The method `pfcTable.Table.MergeRegion` merges table cells within a specified range of rows and columns to form a single cell. The range is a rectangular region specified by the table cell on the upper left of the region and the table cell on the lower right of the region.

The method `pfcTable.Table.SubdivideRegion` removes merges from a region of table cells that were previously merged. The region to remove merges is specified by the table cell on the upper left of the region and the table cell on the lower right of the region.

The methods `pfcTable.Table.DeleteRow` and `pfcTable.Table.DeleteColumn` delete any specified row or column from the table. The methods also remove the text from the affected cells.

The method `pfcTable.Table.SetText` sets text in the table cell.

Use the method `pfcTable.TableOwner.DeleteTable` to delete a specified drawing table from the model permanently. The deleted table cannot be displayed again.

Note

Many of the above methods provide a parameter *Repaint* If this is set to true the table will be repainted after the change. If set to false or null PTC Creo Parametric will delay the repaint, allowing you to perform several operations before showing changes on the screen.

Example: Creation of a Table Listing Datum Points

The sample code in the file `pfcDrawingExamples.java` located at `<creo_jlink_loadpoint>/jlink_appls/jlinkexamples` creates a drawing table that lists the datum points in a model shown in a drawing view.

Drawing Table Segments

Drawing tables can be constructed with one or more segments. Each segment can be independently placed. The segments are specified by an integer identifier starting with 0.

Methods Introduced:

- **pfcSelect.Selection.GetSelTableSegment**
- **pfcTable.Table.GetSegmentCount**
- **pfcTable.Table.GetSegmentSheet**
- **pfcTable.Table.MoveSegment**
- **pfcTable.Table.GetInfo**

The method `pfcSelect.Selection.GetSelTableSegment` returns the value of the segment identifier of the selected table segment. It returns a null value if the selection does not contain a segment identifier.

The method `pfcTable.Table.GetSegmentCount` returns the number of segments in the table.

The method `pfcTable.Table.GetSegmentSheet` determines the sheet number that contains a specified drawing table segment.

The method `pfcTable.Table.MoveSegment` moves a drawing table segment to a new location. Pass the co-ordinates of the target position in the format x, y, z=0.

> **Note**
>
> Set the value of the parameter *Repaint* to true to repaint the drawing with the changes. Set it to false or null to delay the repaint.

To get information about a drawing table pass the value of the segment identifier as input to the method `pfcTable.Table.GetInfo`. The method returns the table information including the rotation, row and column information, and the 3D outline.

Repeat Regions

Methods Introduced:

- **pfcTable.Table.IsCommentCell**
- **pfcTable.Table.GetCellComponentModel**
- **pfcTable.Table.GetCellReferenceModel**
- **pfcTable.Table.GetCellTopModel**
- **pfcTable.TableOwner.UpdateTables**

The methods `pfcTable.Table.IsCommentCell`, `pfcTable.Table.GetCellComponentModel`, `pfcTable.Table.GetCellReferenceModel`, pfcTable.Table.GetCellTopModel, and pfcTable.TableOwner.UpdateTables apply to repeat regions in drawing tables.

The method pfcTable.Table.IsCommentCell tells you whether a cell in a repeat region contains a comment.

The method pfcTable.Table.GetCellComponentModel returns the path to the assembly component model that is being referenced by a cell in a repeat region of a drawing table. It does not return a valid path if the cell attribute is set to NO DUPLICATE or NO DUPLICATE/LEVEL.

The method pfcTable.Table.GetCellReferenceModel returns the reference component that is being referred to by a cell in a repeat region of a drawing table, even if cell attribute is set to NO DUPLICATE or NO DUPLICATE/LEVEL.

The method pfcTable.Table.GetCellTopModel returns the top model that is being referred to by a cell in a repeat region of a drawing table, even if cell attribute is set to NO DUPLICATE or NO DUPLICATE/LEVEL.

Use the method pfcTable.TableOwner.UpdateTables to update the repeat regions in all the tables to account for changes to the model. It is equivalent to the command **Table**, **Repeat Region**, **Update**.

Detail Items

The methods described in this section operate on detail items.

In J-Link you can create, delete and modify detail items, control their display, and query what detail items are present in the drawing. The types of detail items available are:

- Draft Entities—Contain graphical items created in PTC Creo Parametric. The items are as follows:
    - Arc
    - Ellipse
    - Line
    - Point
    - Polygon
    - Spline
- Notes—Textual annotations
- Symbol Definitions—Contained in the drawing’s symbol gallery.
- Symbol Instances—Instances of a symbol placed in a drawing.

- Draft Groups—Groups of detail items that contain notes, symbol instances, and draft entities.
- OLE objects—Object Linking and Embedding (OLE) objects embedded in the PTC Creo Parametric drawing file.

Listing Detail Items

Methods Introduced:

- **pfcModelItem.ModelItemOwner.ListItems**
- **pfcDetail.DetailItemOwner.ListDetailItems**
- **pfcModelItem.ModelItemOwner.GetItemById**
- **pfcDetail.DetailItemOwner.CreateDetailItem**

The method `pfcModelItem.ModelItemOwner.ListItems` returns a list of detail items specified by the parameter *Type* or returns null if no detail items of the specified type are found.

The values of the parameter *Type* for detail items are:

- `ITEM_DTL_ENTITY`—Detail Entity
- `ITEM_DTL_NOTE`—Detail Note
- `ITEM_DTL_GROUP`—Draft Group
- `ITEM_DTL_SYM_DEFINITION`—Detail Symbol Definition
- `ITEM_DTL_SYM_INSTANCE`—Detail Symbol Instance
- `ITEM_DTL_OLE_OBJECT`—Drawing embedded OLE object

If this parameter is set to null, then all the model items in the drawing are listed.

The method `pfcDetail.DetailItemOwner.ListDetailItems` also lists the detail items in the model. Pass the type of the detail item and the sheet number that contains the specified detail items.

Set the input parameter *Type* to the type of detail item to be listed. Set it to null to return all the detail items. The input parameter *SheetNumber*determines the sheet that contains the specified detail item. Pass null to search all the sheets. This argument is ignored if the parameter *Type* is set to `DETAIL_SYM_DEFINITION`.

The method returns a sequence of detail items and returns a null if no items matching the input values are found.

The method `pfcModelItem.ModelItemOwner.GetItemById` returns a detail item based on the type of the detail item and its integer identifier. The method returns a null if a detail item with the specified attributes is not found.

Creating a Detail Item

Methods Introduced:

- **pfcDetail.DetailItemOwner.CreateDetailItem**
- **pfcDetail.pfcDetail.DetailGroupInstructions_Create**

The method `pfcDetail.DetailItemOwner.CreateDetailItem` creates a new detail item based on the instruction data object that describes the type and content of the new detail item. The instructions data object is returned by the method `pfcDetail.pfcDetail.DetailGroupInstructions_Create`. The method returns the newly created detail item.

Detail Entities

A detail entity in J-Link is represented by the `com.ptc.pfc.pfcDetail.DetailEntityItem`. It is a child of the `DetailItem`.

The `com.ptc.pfc.pfcDetail.DetailEntityInstructions` contains specific information used to describe a detail entity item.

Instructions

Methods Introduced:

- **pfcDetail.pfcDetail.DetailEntityInstructions_Create**
- **pfcDetail.DetailEntityInstructions.GetGeometry**
- **pfcDetail.DetailEntityInstructions.SetGeometry**
- **pfcDetail.DetailEntityInstructions.GetIsConstruction**
- **pfcDetail.DetailEntityInstructions.SetIsConstruction**
- **pfcDetail.DetailEntityInstructions.GetColor**
- **pfcDetail.DetailEntityInstructions.SetColor**
- **pfcDetail.DetailEntityInstructions.GetFontName**
- **pfcDetail.DetailEntityInstructions.SetFontName**
- **pfcDetail.DetailEntityInstructions.GetWidth**
- **pfcDetail.DetailEntityInstructions.SetWidth**
- **pfcDetail.DetailEntityInstructions.GetView**
- **pfcDetail.DetailEntityInstructions.SetView**

The method `pfcDetail.pfcDetail.DetailEntityInstructions_Create` creates an instructions object that describes how to construct a detail entity, for use in the methods `pfcDetail.DetailItemOwner.CreateDetailItem`, `pfcDetail.DetailSymbolDefItem.CreateDetailItem`, and `pfcDetail.DetailEntityItem.Modify`.

The instructions object is created based on the curve geometry and the drawing view associated with the entity. The curve geometry describes the trajectory of the detail entity in world units. The drawing view can be a model view returned by the method `pfcModel2D.Model2D.List2DViews` or a drawing sheet background view returned by the method `pfcSheet.SheetOwner.GetSheetBackgroundView`. The background view indicates that the entity is not associated with a particular model view.

The method returns the created instructions object.

Note

Changes to the values of a `pfcDetail.DetailEntityInstructions` object do not take effect until that instructions object is used to modify the entity using `pfcDetail.DetailEntityItem.Modify`.

The method `pfcDetail.DetailEntityInstructions.GetGeometry` returns the geometry of the detail entity item.

The method `pfcDetail.DetailEntityInstructions.SetGeometry` sets the geometry of the detail entity item. For more information refer to Curve Descriptors on page 245.

The method `pfcDetail.DetailEntityInstructions.GetIsConstruction` returns a value that specifies whether the entity is a construction entity.

The method `pfcDetail.DetailEntityInstructions.SetIsConstruction` specifies if the detail entity is a construction entity.

The method `pfcDetail.DetailEntityInstructions.GetColor` returns the color of the detail entity item.

The method `pfcDetail.DetailEntityInstructions.SetColor` sets the color of the detail entity item. Pass null to use the default drawing color.

The method `pfcDetail.DetailEntityInstructions.GetFontName` returns the line style used to draw the entity. The method returns a null value if the default line style is used.

The method `pfcDetail.DetailEntityInstructions.SetFontName` sets the line style for the detail entity item. Pass null to use the default line style.

The method pfcDetail.DetailEntityInstructions.GetWidth returns the value of the width of the entity line. The method returns a null value if the default line width is used.

The method pfcDetail.DetailEntityInstructions.SetWidth specifies the width of the entity line. Pass null to use the default line width.

The method pfcDetail.DetailEntityInstructions.GetView returns the drawing view associated with the entity. The view can either be a model view or a drawing sheet background view.

The method pfcDetail.DetailEntityInstructions.SetView sets the drawing view associated with the entity. The view can either be a model view or a drawing sheet background view.

Example: Create a Draft Line with Predefined Color

The sample code in the file pfcDrawingExamples.java located at <creo_jlink_loadpoint>/jlink_appls/jlinkexamples shows a utility that creates a draft line in one of the colors predefined in PTC Creo Parametric.

Detail Entities Information

Methods Introduced:

- **pfcDetail.DetailEntityItem.GetInstructions**
- **pfcDetail.DetailEntityItem.GetSymbolDef**

The method pfcDetail.DetailEntityItem.GetInstructions returns the instructions data object that is used to construct the detail entity item.

The method pfcDetail.DetailEntityItem.GetSymbolDef returns the symbol definition that contains the entity. This method returns a null value if the entity is not a part of a symbol definition.

Detail Entities Operations

Methods Introduced:

- **pfcDetail.DetailEntityItem.Draw**
- **pfcDetail.DetailEntityItem.Erase**
- **pfcDetail.DetailEntityItem.Modify**

The method pfcDetail.DetailEntityItem.Draw temporarily draws a detail entity item, so that it is removed during the next draft regeneration.

The method pfcDetail.DetailEntityItem.Erase undraws a detail entity item temporarily, so that it is redrawn during the next draft regeneration.

The method `pfcDetail.DetailEntityItem.Modify` modifies the definition of an entity item using the specified instructions data object.

OLE Objects

An object linking and embedding (OLE) object is an external file, such as a document, graphics file, or video file that is created using an external application and which can be inserted into another application, such as PTC Creo Parametric. You can create and insert supported OLE objects into a two-dimensional PTC Creo Parametric file, such as a drawing, report, format file, notebook, or diagram. The functions described in this section enable you to identify and access OLE objects embedded in drawings.

Methods Introduced:

- **pfcDetail.DetailOLEObject.GetApplicationType**
- **pfcDetail.DetailOLEObject.GetOutline**
- **pfcDetail.DetailOLEObject.GetPath**
- **pfcDetail.DetailOLEObject.GetSheet**

The method `pfcDetail.DetailOLEObject.GetApplicationType` returns the type of the OLE object as a string, for example, `Microsoft Word Document`.

The method `pfcDetail.DetailOLEObject.GetOutline` returns the extent of the OLE object embedded in the drawing.

The method `pfcDetail.DetailOLEObject.GetPath` returns the path to the external file for each OLE object, if it is linked to an external file.

The method `pfcDetail.DetailOLEObject.GetSheet` returns the sheet number for the OLE object.

Detail Notes

A detail note in is represented by the `com.ptc.pfc.pfcDetail.DetailNoteItem`. It is a child of the `DetailItem`.

The `com.ptc.pfc.pfcDetail.DetailNoteInstructions` contains specific information that describes a detail note.

Instructions

Methods Introduced:

- **pfcDetail.pfcDetail.DetailNoteInstructions_Create**
- **pfcDetail.DetailNoteInstructions.GetTextLines**
- **pfcDetail.DetailNoteInstructions.SetTextLines**
- **pfcDetail.DetailNoteInstructions.GetIsDisplayed**
- **pfcDetail.DetailNoteInstructions.SetIsDisplayed**
- **pfcDetail.DetailNoteInstructions.GetIsReadOnly**
- **pfcDetail.DetailNoteInstructions.SetIsReadOnly**
- **pfcDetail.DetailNoteInstructions.GetIsMirrored**
- **pfcDetail.DetailNoteInstructions.SetIsMirrored**
- **pfcDetail.DetailNoteInstructions.GetHorizontal**
- **pfcDetail.DetailNoteInstructions.SetHorizontal**
- **pfcDetail.DetailNoteInstructions.GetVertical**
- **pfcDetail.DetailNoteInstructions.SetVertical**
- **pfcDetail.DetailNoteInstructions.GetColor**
- **pfcDetail.DetailNoteInstructions.SetColor**
- **pfcDetail.DetailNoteInstructions.GetLeader**
- **pfcDetail.DetailNoteInstructions.SetLeader**
- **pfcDetail.DetailNoteInstructions.GetTextAngle**
- **pfcDetail.DetailNoteInstructions.SetTextAngle**

The method `pfcDetail.pfcDetail.DetailNoteInstructions_Create` creates a data object that describes how a detail note item should be constructed when passed to the methods `pfcDetail.DetailItemOwner.CreateDetailItem`, `pfcDetail.DetailSymbolDefItem.CreateDetailItem`, or `pfcDetail.DetailNoteItem.Modify`. The parameter *inTextLines* specifies the sequence of text line data objects that describe the contents of the note.

Note

Changes to the values of a`pfcDetail.DetailNoteInstructions` object do not take effect until that instructions object is used to modify the note using `pfcDetail.DetailNoteItem.Modify`

The method `pfcDetail.DetailNoteInstructions.GetTextLines` returns the description of text line contents in the note.

The method pfcDetail.DetailNoteInstructions.SetTextLines sets the description of the text line contents in the note.

The method pfcDetail.DetailNoteInstructions.GetIsDisplayed returns a boolean indicating if the note is currently displayed.

The method pfcDetail.DetailNoteInstructions.SetIsDisplayed sets the display flag for the note.

The method pfcDetail.DetailNoteInstructions.GetIsReadOnly determines whether the note can be edited by the user, while the method pfcDetail.DetailNoteInstructions.SetIsReadOnly toggles the read only status of the note.

The method pfcDetail.DetailNoteInstructions.GetIsMirrored determines whether the note is mirrored, while the method pfcDetail.DetailNoteInstructions.SetIsMirrored toggles the mirrored status of the note.

The method pfcDetail.DetailNoteInstructions.GetHorizontal returns the value of the horizontal justification of the note, while the method pfcDetail.DetailNoteInstructions.SetHorizontal sets the value of the horizontal justification of the note.

The method pfcDetail.DetailNoteInstructions.GetVertical returns the value of the vertical justification of the note, while the method pfcDetail.DetailNoteInstructions.SetVertical sets the value of the vertical justification of the note.

The method pfcDetail.DetailNoteInstructions.GetColor returns the color of the detail note item. The method returns a null value to represent the default drawing color.

Use the method pfcDetail.DetailNoteInstructions.SetColor to set the color of the detail note item. Pass null to use the default drawing color.

The method pfcDetail.DetailNoteInstructions.GetLeader returns the locations of the detail note item and information about the leaders.

The method pfcDetail.DetailNoteInstructions.SetLeader sets the values of the location of the detail note item and the locations where the leaders are attached to the drawing.

The method pfcDetail.DetailNoteInstructions.GetTextAngle returns the value of the angle of the text used in the note. The method returns a null value if the angle is 0.0.

The method pfcDetail.DetailNoteInstructions.SetTextAngle sets the value of the angle of the text used in the note. Pass null to use the angle 0.0.

Example: Create Drawing Note at Specified Location with Leader to Surface and Surface Name

The sample code in the file `pfcDrawingExamples.java` located at `<creo_jlink_loadpoint>/jlink_appls/jlinkexamples` creates a drawing note at a specified location, with a leader attached to a solid surface, and displays the name of the surface.

Detail Notes Information

Methods Introduced:

- **pfcDetail.DetailNoteItem.GetInstructions**
- **pfcDetail.DetailNoteItem.GetSymbolDef**
- **pfcDetail.DetailNoteItem.GetLineEnvelope**
- **pfcDetail.DetailNoteItem.GetModelReference**

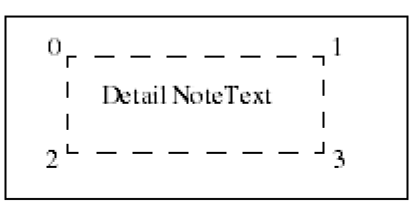
The method `pfcDetail.DetailNoteItem.GetInstructions` returns an instructions data object that describes how to construct the detail note item. This method takes a `ProBoolean` argument, *GiveParametersAsNames*, which determines whether symbolic representations of parameters and drawing properties in the note text should be displayed, or the actual text seen by the user should be displayed.

Note

PTC Creo Parametric does not resolve and replace symbolic callouts for notes which are not displayed. Therefore, if the note is not displayed or is hidden in a layer, the text retrieved may contain symbolic callouts, even when *GiveParametersAsNames* is false.

The method `pfcDetail.DetailNoteItem.GetSymbolDef` returns the symbol definition that contains the note. The method returns a null value if the note is not a part of a symbol definition.

The method `pfcDetail.DetailNoteItem.GetLineEnvelope` determines the screen coordinates of the envelope around the detail note. This envelope is defined by four points. The following figure illustrates how the point order is determined.



The ordering of the points is maintained even if the notes are mirrored or are at an angle.

The method `pfcDetail.DetailNoteItem.GetModelReference` returns the model referenced by the parameterized text in a note. The model is referenced based on the line number and the text index where the parameterized text appears.

Details Notes Operations

Methods Introduced:

- **pfcDetail.DetailNoteItem.Draw**
- **pfcDetail.DetailNoteItem.Show**
- **pfcDetail.DetailNoteItem.Erase**
- **pfcDetail.DetailNoteItem.Remove**
- **pfcDetail.DetailNoteItem.Modify**

The method `pfcDetail.DetailNoteItem.Draw` temporarily draws a detail note item, so that it is removed during the next draft regeneration.

The method `pfcDetail.DetailNoteItem.Show` displays the note item, such that it is repainted during the next draft regeneration.

The method `pfcDetail.DetailNoteItem.Erase` undraws a detail note item temporarily, so that it is redrawn during the next draft regeneration.

The method `pfcDetail.DetailNoteItem.Remove` undraws a detail note item permanently, so that it is not redrawn during the next draft regeneration.

The method `pfcDetail.DetailNoteItem.Modify` modifies the definition of an existing detail note item based on the instructions object that describes the new detail note item.

Detail Groups

A detail group in J-Link is represented by the `com.ptc.pfc.pfcDetail.DetailGroupItem`. It is a child of the `DetailItem`.

The interface `com.ptc.pfc.pfcDetail.DetailGroupInstructions` contains information used to describe a detail group item.

Instructions

Method Introduced:

- **pfcDetail.pfcDetail.DetailGroupInstructions_Create**
- **pfcDetail.DetailGroupInstructions.GetName**
- **pfcDetail.DetailGroupInstructions.SetName**
- **pfcDetail.DetailGroupInstructions.GetElements**
- **pfcDetail.DetailGroupInstructions.SetElements**
- **pfcDetail.DetailGroupInstructions.GetIsDisplayed**
- **pfcDetail.DetailGroupInstructions.SetIsDisplayed**

The method `pfcDetail.pfcDetail.DetailGroupInstructions_Create` creates an instruction data object that describes how to construct a detail group for use in `pfcDetail.DetailItemOwner.CreateDetailItem` and `pfcDetail.DetailGroupItem.Modify`.

Note

Changes to the values of a `pfcDetail.DetailGroupInstructions` object do not take effect until that instructions object is used to modify the group using `pfcDetail.DetailGroupItem.Modify`.

The method `pfcDetail.DetailGroupInstructions.GetName` returns the name of the detail group.

The method `pfcDetail.DetailGroupInstructions.SetName` sets the name of the detail group.

The method `pfcDetail.DetailGroupInstructions.GetElements` returns the sequence of the detail items(notes, groups and entities) contained in the group.

The method `pfcDetail.DetailGroupInstructions.SetElements` sets the sequence of the detail items contained in the group.

The method `pfcDetail.DetailGroupInstructions.GetIsDisplayed` returns whether the detail group is displayed in the drawing.

The method `pfcDetail.DetailGroupInstructions.SetIsDisplayed` toggles the display of the detail group.

Detail Groups Information

Method Introduced:

- **pfcDetail.DetailGroupItem.GetInstructions**

The method `pfcDetail.DetailGroupItem.GetInstructions` gets a data object that describes how to construct a detail group item. The method returns the data object describing the detail group item.

Detail Groups Operations

Methods Introduced:

- **pfcDetail.DetailGroupItem.Draw**
- **pfcDetail.DetailGroupItem.Erase**
- **pfcDetail.DetailGroupItem.Modify**

The method `pfcDetail.DetailGroupItem.Draw` temporarily draws a detail group item, so that it is removed during the next draft generation.

The method `pfcDetail.DetailGroupItem.Erase` temporarily undraws a detail group item, so that it is redrawn during the next draft generation.

The method `pfcDetail.DetailGroupItem.Modify` changes the definition of a detail group item based on the data object that describes how to construct a detail group item.

Example: Create New Group of Items

The sample code in the file `pfcDrawingExamples.java` located at <creo_jlink_loadpoint>/jlink_appls/jlinkexamples creates a group from a set of selected detail items.

Detail Symbols

Detail Symbol Definitions

A detail symbol definition in J-Link is represented by the `pfcDetail.DetailSymbolDefItem`. It is a child of the `DetailItem` .

The interface `pfcDetail.DetailSymbolDefInstructions` contains information that describes a symbol definition. It can be used when creating symbol definition entities or while accessing existing symbol definition entities.

Instructions

Methods Introduced:

- **pfcDetail.pfcDetail.DetailSymbolDefInstructions_Create**
- **pfcDetail.DetailSymbolDefInstructions.GetSymbolHeight**

- **pfcDetail.DetailSymbolDefInstructions.SetSymbolHeight**
- **pfcDetail.DetailSymbolDefInstructions.GetHasElbow**
- **pfcDetail.DetailSymbolDefInstructions.SetHasElbow**
- **pfcDetail.DetailSymbolDefInstructions.GetIsTextAngleFixed**
- **pfcDetail.DetailSymbolDefInstructions.SetIsTextAngleFixed**
- **pfcDetail.DetailSymbolDefInstructions.GetScaledHeight**
- **pfcDetail.DetailSymbolDefInstructions.GetAttachments**
- **pfcDetail.DetailSymbolDefInstructions.SetAttachments**
- **pfcDetail.DetailSymbolDefInstructions.GetFullPath**
- **pfcDetail.DetailSymbolDefInstructions.SetFullPath**
- **pfcDetail.DetailSymbolDefInstructions.GetReference**
- **pfcDetail.DetailSymbolDefInstructions.SetReference**

The method `pfcDetail.pfcDetail.DetailSymbolDefInstructions_Create` creates an instruction data object that describes how to create a symbol definition based on the path and name of the symbol definition. The instructions object is passed to the methods `pfcDetailItemOwner.CreateDetailItem` and `pfcDetailSymbolDefItem.Modify`.

Note

Changes to the values of a `pfcDetail.DetailSymbolDefInstructions` object do not take effect until that instructions object is used to modify the definition using the method `pfcDetail.DetailSymbolDefItem.Modify`.

The method `pfcDetail.DetailSymbolDefInstructions.GetSymbolHeight` returns the value of the height type for the symbol definition. The symbol definition height options are as follows:

- `SYMDEF_FIXED`—Symbol height is fixed.
- `SYMDEF_VARIABLE`—Symbol height is variable.
- `SYMDEF_RELATIVE_TO_TEXT`—Symbol height is determined relative to the text height.

The method `pfcDetail.DetailSymbolDefInstructions.SetSymbolHeight` sets the value of the height type for the symbol definition.

The method `pfcDetail.DetailSymbolDefInstructions.GetHasElbow` determines whether the symbol definition includes an elbow.

The method `pfcDetail.DetailSymbolDefInstructions.SetHasElbow` decides if the symbol definition should include an elbow.

The method `pfcDetail.DetailSymbolDefInstructions.GetIsTextAngleFixed` returns whether the text of the angle is fixed.

The method `pfcDetail.DetailSymbolDefInstructions.SetIsTextAngleFixed` toggles the requirement that the text angle be fixed.

The method `pfcDetail.DetailSymbolDefInstructions.GetScaledHeight` returns the height of the symbol definition in inches.

The method `pfcDetail.DetailSymbolDefInstructions.GetAttachments` returns the value of the sequence of the possible instance attachment points for the symbol definition.

The method `pfcDetail.DetailSymbolDefInstructions.SetAttachments` sets the value of the sequence of the possible instance attachment points for the symbol definition.

The method `pfcDetail.DetailSymbolDefInstructions.GetFullPath` returns the value of the complete path of the symbol definition file.

The method `pfcDetail.DetailSymbolDefInstructions.SetFullPath` sets the value of the complete path of the symbol definition path.

The method `pfcDetail.DetailSymbolDefInstructions.GetReference` returns the text reference information for the symbol definition. It returns a null value if the text reference is not used. The text reference identifies the text item used for a symbol definition which has a height type of `SYMDEF_TEXT_RELATED`.

The method `pfcDetail.DetailSymbolDefInstructions.SetReference` sets the text reference information for the symbol definition.

Detail Symbol Definitions Information

Methods Introduced:

- **pfcDetail.DetailSymbolDefItem.ListDetailItems**
- **pfcDetail.DetailSymbolDefItem.GetInstructions**

The method `pfcDetail.DetailSymbolDefItem.ListDetailItems` lists the detail items in the symbol definition based on the type of the detail item.

The method `pfcDetail.DetailSymbolDefItem.GetInstructions` returns an instruction data object that describes how to construct the symbol definition.

Detail Symbol Definitions Operations

Methods Introduced:

- **pfcDetail.DetailSymbolDefItem.CreateDetailItem**
- **pfcDetail.DetailSymbolDefItem.Modify**

The method `pfcDetail.DetailSymbolDefItem.CreateDetailItem` creates a detail item in the symbol definition based on the instructions data object. The method returns the detail item in the symbol definition.

The method `pfcDetail.DetailSymbolDefItem.Modify` modifies a symbol definition based on the instructions data object that contains information about the modifications to be made to the symbol definition.

Retrieving Symbol Definitions

Methods Introduced:

- **pfcDetail.DetailItemOwner.RetrieveSymbolDefinition**

The method `pfcDetail.DetailItemOwner.RetrieveSymbolDefinition` retrieves a symbol definition from the disk.

The input parameters of this method are:

- *FileName*—Name of the symbol definition file
- *FilePath*—Path to the symbol definition file. It is relative to the path specified by the option "`pro_symbol_dir`" in the configuration file. A null value indicates that the function should search the current directory.
- *Version*—Numerical version of the symbol definition file. A null value retrieves the latest version.
- *UpdateUnconditionally*—True if PTC Creo Parametric should update existing instances of this symbol definition, or false to quit the operation if the definition exists in the model.

The method returns the retrieved symbol definition.

Detail Symbol Instances

A detail symbol instance in J-Link is represented by the `pfcDetail.DetailSymbolInstItem`. It is a child of the `DetailItem`.

The `Detail.DetailSymbolInstInstructions` contains information that describes a symbol instance. It can be used when creating symbol instances and while accessing existing groups.

Instructions

Methods Introduced:

- **pfcDetail.pfcDetail.DetailSymbolInstInstructions_Create**
- **pfcDetail.DetailSymbolInstInstructions.GetIsDisplayed**
- **pfcDetail.DetailSymbolInstInstructions.SetIsDisplayed**
- **pfcDetail.DetailSymbolInstInstructions.GetColor**
- **pfcDetail.DetailSymbolInstInstructions.SetColor**
- **pfcDetail.DetailSymbolInstInstructions.GetSymbolDef**
- **pfcDetail.DetailSymbolInstInstructions.SetSymbolDef**
- **pfcDetail.DetailSymbolInstInstructions.GetAttachOnDefType**
- **pfcDetail.DetailSymbolInstInstructions.SetAttachOnDefType**
- **pfcDetail.DetailSymbolInstInstructions.GetDefAttachment**
- **pfcDetail.DetailSymbolInstInstructions.SetDefAttachment**
- **pfcDetail.DetailSymbolInstInstructions.GetInstAttachment**
- **pfcDetail.DetailSymbolInstInstructions.SetInstAttachment**
- **pfcDetail.DetailSymbolInstInstructions.GetAngle**
- **pfcDetail.DetailSymbolInstInstructions.SetAngle**
- **pfcDetail.DetailSymbolInstInstructions.GetScaledHeight**
- **pfcDetail.DetailSymbolInstInstructions.SetScaledHeight**
- **pfcDetail.DetailSymbolInstInstructions.GetTextValues**
- **pfcDetail.DetailSymbolInstInstructions.SetTextValues**
- **pfcDetail.DetailSymbolInstInstructions.GetCurrentTransform**
- **pfcDetail.DetailSymbolInstInstructions.SetGroups**

The method `pfcDetail.pfcDetail.DetailSymbolInstInstructions_Create` creates a data object that contains information about the placement of a symbol instance.

Note

Changes to the values of a `pfcDetail.DetailSymbolInstInstructions` object do not take effect until that instructions object is used to modify the instance using `pfcDetail.DetailSymbolInstItem.Modify`.

The method `pfcDetail.DetailSymbolInstInstructions.GetIsDisplayed` returns a value that specifies whether the instance of the symbol is displayed.

Use the method `pfcDetail.DetailSymbolInstInstructions.SetIsDisplayed` to switch the display of the symbol instance.

The method `pfcDetail.DetailSymbolInstInstructions.GetColor` returns the color of the detail symbol instance. A null value indicates that the default drawing color is used.

The method `pfcDetail.DetailSymbolInstInstructions.SetColor` sets the color of the detail symbol instance. Pass null to use the default drawing color.

The method `pfcDetail.DetailSymbolInstInstructions.GetSymbolDef` returns the symbol definition used for the instance.

The method `pfcDetail.DetailSymbolInstInstructions.SetSymbolDef` sets the value of the symbol definition used for the instance.

The method `pfcDetail.DetailSymbolInstInstructions.GetAttachOnDefType` returns the attachment type of the instance. The method returns a null value if the attachment represents a free attachment. The attachment options are as follows:

- `SYMDEFATTACH_FREE`—Attachment on a free point.
- `SYMDEFATTACH_LEFT_LEADER`—Attachment via a leader on the left side of the symbol.
- `SYMDEFATTACH_RIGHT_LEADER`— Attachment via a leader on the right side of the symbol.
- `SYMDEFATTACH_RADIAL_LEADER`—Attachment via a leader at a radial location.

- SYMDEFATTACH_ON_ITEM—Attachment on an item in the symbol definition.
- SYMDEFATTACH_NORMAL_TO_ITEM—Attachment normal to an item in the symbol definition.

The method pfcDetail.DetailSymbolInstInstructions.SetAttachOnDefType sets the attachment type of the instance.

The method pfcDetail.DetailSymbolInstInstructions.GetDefAttachment returns the value that represents the way in which the instance is attached to the symbol definition.

The method pfcDetail.DetailSymbolInstInstructions.SetDefAttachment specifies the way in which the instance is attached to the symbol definition.

The method pfcDetail.DetailSymbolInstInstructions.GetInstAttachment returns the value of the attachment of the instance that includes location and leader information.

The method pfcDetail.DetailSymbolInstInstructions.SetInstAttachment sets value of the attachment of the instance.

The method pfcDetail.DetailSymbolInstInstructions.GetAngle returns the value of the angle at which the instance is placed. The method returns a null value if the value of the angle is 0 degrees.

The method pfcDetail.DetailSymbolInstInstructions.SetAngle sets the value of the angle at which the instance is placed.

The method pfcDetail.DetailSymbolInstInstructions.GetScaledHeight returns the height of the symbol instance in the owner drawing or model coordinates. This value is consistent with the height value shown for a symbol instance in the **Properties** dialog box in the PTC Creo Parametric User Interface.

Note

The scaled height obtained using the above method is partially based on the properties of the symbol definition assigned using the method `pfcDetail.DetailSymbolInstInstructions.GetSymbolDef`. Changing the symbol definition may change the calculated value for the scaled height.

The method `pfcDetail.DetailSymbolInstInstructions.SetScaledHeight` sets the value of the height of the symbol instance in the owner drawing or model coordinates.

The method `pfcDetail.DetailSymbolInstInstructions.GetTextValues` returns the sequence of variant text values used while placing the symbol instance.

The method `pfcDetail.DetailSymbolInstInstructions.SetTextValues` sets the sequence of variant text values while placing the symbol instance.

The method `pfcDetail.DetailSymbolInstInstructions.GetCurrentTransform` returns the coordinate transformation matrix to place the symbol instance.

The method `pfcDetail.DetailSymbolInstInstructions.SetGroups` `DetailSymbolGroupOption`

- `DETAIL_SYMBOL_GROUP_INTERACTIVE`—Symbol groups are interactively selected for display. This is the default value in the `GRAPHICS` mode.
- `DETAIL_SYMBOL_GROUP_ALL`—All non-exclusive symbol groups are included for display.
- `DETAIL_SYMBOL_GROUP_NONE`—None of the non-exclusive symbol groups are included for display.
- `DETAIL_SYMBOL_GROUP_CUSTOM`—Symbol groups specified by the application are displayed.

Refer to the section Detail Symbol Groups on page 169 for more information on detail symbol groups.

Detail Symbol Instances Information

Method Introduced:

- **pfcDetail.DetailSymbolInstItem.GetInstructions**

The method `pfcDetail.DetailSymbolInstItem.GetInstructions` returns an instructions data object that describes how to construct a symbol instance. This method takes a `ProBoolean` argument, *GiveParametersAsNames*, which determines whether symbolic representations of parameters and drawing properties in the symbol instance should be displayed, or the actual text seen by the user should be displayed.

Detail Symbol Instances Operations

Methods Introduced:

- **pfcDetail.DetailSymbolInstItem.Draw**
- **pfcDetail.DetailSymbolInstItem.Erase**
- **pfcDetail.DetailSymbolInstItem.Show**
- **pfcDetail.DetailSymbolInstItem.Remove**
- **pfcDetail.DetailSymbolInstItem.Modify**

The method `pfcDetail.DetailSymbolInstItem.Draw` draws a symbol instance temporarily to be removed on the next draft regeneration.

The method `pfcDetail.DetailSymbolInstItem.Erase` undraws a symbol instance temporarily from the display to be redrawn on the next draft generation.

The method `pfcDetail.DetailSymbolInstItem.Show` displays a symbol instance to be repainted on the next draft regeneration.

The method `pfcDetail.DetailSymbolInstItem.Remove` deletes a symbol instance permanently.

The method `pfcDetail.DetailSymbolInstItem.Modify` modifies a symbol instance based on the instructions data object that contains information about the modifications to be made to the symbol instance.

Example: Create a Free Instance of Symbol Definition

The sample code in the file `pfcDrawingExamples.java` located at `<creo_jlink_loadpoint>/jlink_appls/jlinkexamples` creates a free instance of a symbol definition.

Example: Create a Free Instance of a Symbol Definition with drawing unit heights, variable text and groups

The sample code in the file `pfcDrawingExamples.java` located at `<creo_jlink_loadpoint>/jlink_appls/jlinkexamples` creates a free instance of a symbol definition with drawing unit heights, variable text and groups.

Detail Symbol Groups

A detail symbol group in J-Link is represented by the pfcDetail.DetailSymbolGroup. It is a child of the pfcObject.Object . A detail symbol group is accessible only as a part of the contents of a detail symbol definition or instance.

The interface pfcDetail.DetailSymbolGroupInstructions contains information that describes a symbol group. It can be used when creating new symbol groups, or while accessing or modifying existing groups.

Instructions

Methods Introduced:

- **pfcDetail.pfcDetail.DetailSymbolGroupInstructions_Create**
- **pfcDetail.DetailSymbolGroupInstructions.GetItems**
- **pfcDetail.DetailSymbolGroupInstructions.SetItems**
- **pfcDetail.DetailSymbolGroupInstructions.GetName**
- **pfcDetail.DetailSymbolGroupInstructions.SetName**

The method pfcDetail.pfcDetail.DetailSymbolGroupInstructions_Create creates the pfcDetail.DetailSymbolGroupInstructions data object that stores the name of the symbol group and the list of detail items to be included in the symbol group.

Note

Changes to the values of the pfcDetail.DetailSymbolGroupInstructions data object do not take effect until this object is used to modify the instance using the method pfcDetail.DetailSymbolGroup.Modify.

The method pfcDetail.DetailSymbolGroupInstructions.GetItems returns the list of detail items included in the symbol group.

The method pfcDetail.DetailSymbolGroupInstructions.SetItems sets the list of detail items to be included in the symbol group.

The method pfcDetail.DetailSymbolGroupInstructions.GetName returns the name of the symbol group.

The method `pfcDetail.DetailSymbolGroupInstructions.SetName` assigns the name of the symbol group.

Detail Symbol Group Information

Methods Introduced:

- **pfcDetail.DetailSymbolGroup.GetInstructions**
- **pfcDetail.DetailSymbolGroup.GetParentGroup**
- **pfcDetail.DetailSymbolGroup.GetParentDefinition**
- **pfcDetail.DetailSymbolGroup.ListChildren**
- **pfcDetail.DetailSymbolDefItem.ListSubgroups**
- **pfcDetail.DetailSymbolDefItem.IsSubgroupLevelExclusive**
- **pfcDetail.DetailSymbolInstItem.ListGroups**

The method `pfcDetail.DetailSymbolGroup.GetInstructions` returns the `pfcDetail.DetailSymbolGroupInstructions` data object that describes how to construct a symbol group.

The method `pfcDetail.DetailSymbolGroup.GetParentGroup` returns the parent symbol group to which a given symbol group belongs.

The method `pfcDetail.DetailSymbolGroup.GetParentDefinition` returns the symbol definition of a given symbol group.

The method `pfcDetail.DetailSymbolGroup.ListChildren` lists the subgroups of a given symbol group.

The method `pfcDetail.DetailSymbolDefItem.ListSubgroups` lists the subgroups of a given symbol group stored in the symbol definition at the indicated level.

The method `pfcDetail.DetailSymbolDefItem.IsSubgroupLevelExclusive` identifies if the subgroups of a given symbol group stored in the symbol definition at the indicated level are exclusive or independent. If groups are exclusive, only one of the groups at this level can be active in the model at any time. If groups are independent, any number of groups can be active.

The method `pfcDetail.DetailSymbolInstItem.ListGroups` lists the symbol groups included in a symbol instance. The `SymbolGroupFilter` argument determines the types of symbol groups that can be listed. It takes the following values:

- `DTLSYMINST_ALL_GROUPS`—Retrieves all groups in the definition of the symbol instance.
- `DTLSYMINST_ACTIVE_GROUPS`—Retrieves only those groups that are actively shown in the symbol instance.
- `DTLSYMINST_INACTIVE_GROUPS`—Retrieves only those groups that are not shown in the symbol instance.

Detail Symbol Group Operations

Methods Introduced:

- **pfcDetail.DetailSymbolGroup.Delete**
- **pfcDetail.DetailSymbolGroup.Modify**
- **pfcDetail.DetailSymbolDefItem.CreateSubgroup**
- **pfcDetail.DetailSymbolDefItem.SetSubgroupLevelExclusive**
- **pfcDetail.DetailSymbolDefItem.SetSubgroupLevelIndependent**

The method `pfcDetail.DetailSymbolGroup.Delete` deletes the specified symbol group from the symbol definition. This method does not delete the entities contained in the group.

The method `pfcDetail.DetailSymbolGroup.Modify` modifies the specified symbol group based on the `pfcDetail.DetailSymbolGroupInstructions` data object that contains information about the modifications that can be made to the symbol group.

The method `pfcDetail.DetailSymbolDefItem.CreateSubgroup`creates a new subgroup in the symbol definition at the indicated level below the parent group.

The method `pfcDetail.DetailSymbolDefItem.SetSubgroupLevelExclusive` makes the subgroups of a symbol group exclusive at the indicated level in the symbol definition.

Note

After you set the subgroups of a symbol group as exclusive, only one of the groups at the indicated level can be active in the model at any time.

The method `pfcDetail.DetailSymbolDefItem.SetSubgroupLevelIndependent` makes the subgroups of a symbol group independent at the indicated level in the symbol definition.

Note

After you set the subgroups of a symbol group as independent, any number of groups at the indicated level can be active in the model at any time.

Detail Attachments

A detail attachment in J-Link is represented by the `pfcDetail.Attachment`. It is used for the following tasks:

- The way in which a drawing note or a symbol instance is placed in a drawing.
- The way in which a leader on a drawing note or symbol instance is attached.

Method Introduced:

- **pfcDetail.Attachment.GetType**

The method `pfcDetail.Attachment.GetType` returns the `pfcDetail.AttachmentType`object containing the types of detail attachments. The detail attachment types are as follows:

- `ATTACH_FREE`—The attachment is at a free point possibly with respect to a given drawing view.
- `ATTACH_PARAMETRIC`—The attachment is to a point on a surface or an edge of a solid.
- `ATTACH_OFFSET`—The attachment is offset to another drawing view, to a model item, or to a 3D model annotation.
- `ATTACH_TYPE_UNSUPPORTED`—The attachment is to an item that cannot be represented in PFC at the current time. However, you can still retrieve the location of the attachment.

Free Attachment

The `ATTACH_FREE` detail attachment type is represented by the `pfcDetail.FreeAttachment`. It is a child of the `pfcDetail.Attachment` .

MethodsIntroduced:

- **pfcDetail.FreeAttachment.GetAttachmentPoint**
- **pfcDetail.FreeAttachment.SetAttachmentPoint**
- **pfcDetail.FreeAttachment.GetView**
- **pfcDetail.FreeAttachment.SetView**

The method pfcDetail.FreeAttachment.GetAttachmentPoint returns the attachment point. This location is in screen coordinates for drawing items, symbol instances and surface finishes on flat-to-screen annotation planes, and in model coordinates for symbols and surface finishes on 3D model annotation planes.

The method pfcDetail.FreeAttachment.SetAttachmentPoint sets the attachment point.

The method pfcDetail.FreeAttachment.GetView returns the drawing view to which the attachment is related. The attachment point is relative to the drawing view, that is the attachment point moves when the drawing view is moved. This method returns a NULL value, if the detail attachment is not related to a drawing view, but is placed at the specified location in the drawing sheet, or if the attachment is offset to a model item or to a 3D model annotation.

The method pfcDetail.FreeAttachment.SetView sets the drawing view.

Parametric Attachment

The ATTACH_PARAMETRIC detail attachment type is represented by the pfcDetail.ParametricAttachment. It is a child of the pfcDetail.Attachment.

MethodsIntroduced:

- **pfcDetail.ParametricAttachment.GetAttachedGeometry**
- **pfcDetail.ParametricAttachment.SetAttachedGeometry**

The method pfcDetail.ParametricAttachment.GetAttachedGeometry returns the pfcSelect.Selection object representing the item to which the detail attachment is attached. This includes the drawing view in which the attachment is made.

The method pfcDetail.ParametricAttachment.SetAttachedGeometry assigns the pfcSelect.Selection object representing the item to which the detail attachment is attached. This object must include the target drawing view. The attachment will occur at the selected parameters.

Offset Attachment

The ATTACH_OFFSET detail attachment type is represented by the pfcDetail.OffsetAttachment. It is a child of the pfcDetail.Attachment.

MethodsIntroduced:

- **pfcDetail.OffsetAttachment.GetAttachedGeometry**
- **pfcDetail.OffsetAttachment.SetAttachedGeometry**
- **pfcDetail.OffsetAttachment.GetAttachmentPoint**
- **pfcDetail.OffsetAttachment.SetAttachmentPoint**

The method `pfcDetail.OffsetAttachment.GetAttachedGeometry` returns the `pfcSelect.Selection` object representing the item to which the detail attachment is attached. This includes the drawing view where the attachment is made, if the offset reference is in a model.

The method `pfcDetail.OffsetAttachment.SetAttachedGeometry` assigns the `pfcSelect.Selection` object representing the item to which the detail attachment is attached. This can include the drawing view. The attachment will occur at the selected parameters.

The method `pfcDetail.OffsetAttachment.GetAttachmentPoint` returns the attachment point. This location is in screen coordinates for drawing items, symbol instances and surface finishes on flat-to-screen annotation planes, and in model coordinates for symbols and surface finishes on 3D model annotation planes. The distance from the attachment point to the location of the item to which the detail attachment is attached is saved as the offset distance.

The method `pfcDetail.OffsetAttachment.SetAttachmentPoint` sets the attachment point in screen coordinates.

Unsupported Attachment

The `ATTACH_TYPE_UNSUPPORTED` detail attachment type is represented by the `pfcDetail.UnsupportedAttachment`. It is a child of the `pfcDetail.Attachment`.

Method Introduced:

- **pfcDetail.UnsupportedAttachment.GetAttachmentPoint**
- **pfcDetail.UnsupportedAttachment.SetAttachmentPoint**

The method `pfcDetail.UnsupportedAttachment.GetAttachmentPoint` returns the attachment point. This location is in screen coordinates for drawing items, symbol instances and surface finishes on flat-to-screen annotation planes, and in model coordinates for symbols and surface finishes on 3D model annotation planes.

The method `pfcDetail.UnsupportedAttachment.SetAttachmentPoint` assigns the attachment point in screen coordinates.

12

Solid

Most of the objects and methods in J-Link are used with solid models (parts and assemblies). Because solid objects inherit from the interface `Model`, you can use any of the `Model` methods on any `Solid`, `Part`, or `Assembly` object.

Getting a Solid Object

Methods Introduced:

- **pfcSession.BaseSession.CreatePart**
- **pfcSession.BaseSession.CreateAssembly**
- **pfcAssembly.ComponentPath.GetRoot**
- **pfcAssembly.ComponentPath.GetLeaf**
- **pfcMFG.MFG.GetSolid**

The methods `pfcSession.BaseSession.CreatePart` and `pfcSession.BaseSession.CreateAssembly` create new solid models with the names you specify.

The methods `pfcAssembly.ComponentPath.GetRoot` and `pfcAssembly.ComponentPath.GetLeaf` specify the solid objects that make up the component path of an assembly component model. You can get a component path object from any component that has been interactively selected.

The method `pfcMFG.MFG.GetSolid` retrieves the storage solid in which the manufacturing model's features are placed. In order to create a UDF group in the manufacturing model, call the method `pfcSolid.Solid.CreateUDFGroup` on the storage solid.

Solid Information

Methods Introduced:

- **pfcSolid.Solid.GetRelativeAccuracy**
- **pfcSolid.Solid.SetRelativeAccuracy**
- **pfcSolid.Solid.GetAbsoluteAccuracy**
- **pfcSolid.Solid.SetAbsoluteAccuracy**

You can set the relative and absolute accuracy of any solid model using these methods. Relative accuracy is relative to the size of the solid. For example, a relative accuracy of .01 specifies that the solid must be accurate to within 1/100 of its size. Absolute accuracy is measured in absolute units (inches, centimeters, and so on).

Note

For a change in accuracy to take effect, you must regenerate the model.

Solid Operations

Methods Introduced:

- **pfcSolid.Solid.Regenerate**
- **pfcSolid.pfcSolid.RegenInstructions_Create**
- **pfcSolid.RegenInstructions.SetAllowFixUI**
- **pfcSolid.RegenInstructions.SetForceRegen**
- **pfcSolid.RegenInstructions.SetFromFeat**
- **pfcSolid.RegenInstructions.SetRefreshModelTree**
- **pfcSolid.RegenInstructions.SetResumeExcludedComponents**
- **pfcSolid.RegenInstructions.SetUpdateAssemblyOnly**
- **pfcSolid.RegenInstructions.SetUpdateInstances**
- **pfcSolid.Solid.GetGeomOutline**
- **pfcSolid.Solid.EvalOutline**
- **pfcSolid.Solid.GetIsSkeleton**
- **pfcSolid.Solid.ListGroups**

The method `pfcSolid.Solid.Regenerate` causes the solid model to regenerate according to the instructions provided in the form of the `pfcSolid.RegenInstructions` object. Passing a null value for the instructions argument causes an automatic regeneration.

Pro/ENGINEER Wildfire 5.0 introduces the No-Resolve mode, wherein if a model and feature regeneration fails, failed features and children of failed features are created and regeneration of other features continues. However, J-Link does not support regeneration in this mode. The method `pfcSolid.Solid.Regenerate` throws an exception `pfcExceptions.XToolkitBadContext`, if PTC Creo Parametric is running in the No-Resolve mode. To continue with the Pro/ENGINEER Wildfire 4.0 behavior in the Resolve mode, set the configuration option `regen_failure_handling` to `resolve_mode` in the PTC Creo Parametric session.

Note

Setting the configuration option to switch to Resolve mode ensures the old behavior as long as you do not retrieve the models saved under the No-Resolve mode. To consistently preserve the old behavior, use Resolve mode from the beginning and throughout your PTC Creo Parametric session.

The `pfcSolid.RegenInstructions` object contains the following input parameters:

- *AllowFixUI*—Determines whether or not to activate the **Fix Model** user interface, if there is an error.

  Use the method `pfcSolid.RegenInstructions.SetAllowFixUI` to modify this parameter.
- *ForceRegen*—PTC Creo Parametric

  Use the method `pfcSolid.RegenInstructions.SetForceRegen` to modify this parameter.
- *FromFeat*—Not currently used. This parameter is reserved for future use.

  Use the method `pfcSolid.RegenInstructions.SetFromFeat` to modify this parameter.
- *RefreshModelTree*—PTC Creo Parametric **Model Tree**

  Use the method `pfcSolid.RegenInstructions.SetRefreshModelTree` to modify this parameter.
- *ResumeExcludedComponents*—PTC Creo Parametric

  Use the method `pfcSolid.RegenInstructions.SetResumeExcludedComponents` to modify this parameter.
- *UpdateAssemblyOnly*—Updates the placements of an assembly and all its sub-assemblies, and regenerates the assembly features and intersected parts. If the affected assembly is retrieved as a simplified representation, then the locations of the components are updated. If this attribute is false, the component locations are not updated, even if the simplified representation is retrieved. By default, it is false.

  Use the method `pfcSolid.RegenInstructions.SetUpdateAssemblyOnly` to modify this parameter.
- *UpdateInstances*—Updates the instances of the solid model in memory. This may slow down the regeneration process. By default, this attribute is false.

  Use the method `pfcSolid.RegenInstructions.SetUpdateInstances` to modify this parameter.

The method `pfcSolid.Solid.GetGeomOutline` returns the three-dimensional bounding box for the specified solid. The method `pfcSolid.Solid.EvalOutline` also returns a three-dimensional bounding box, but you can specify the coordinate system used to compute the extents of the solid object.

The method `pfcSolid.Solid.GetIsSkeleton` determines whether the part model is a skeleton or a concept model. It returns a true value if the model is a skeleton, else it returns a false.

The method `pfcSolid.Solid.ListGroups` returns the list of groups (including UDFs) in the solid.

Solid Units

Each model has a basic system of units to ensure all material properties of that model are consistently measured and defined. All models are defined on the basis of the system of units. A part can have only one system of unit.

The following types of quantities govern the definition of units of measurement:

- Basic Quantities—The basic units and dimensions of the system of units. For example, consider the `Centimeter Gram Second (CGS)` system of unit. The basic quantities for this system of units are:
    - Length—`cm`
    - Mass—`g`
    - Force—`dyne`
    - Time—`sec`
    - Temperature—`K`
- Derived Quantities—The derived units are those that are derived from the basic quantities. For example, consider the `Centimeter Gram Second (CGS)` system of unit. The derived quantities for this system of unit are as follows:
    - Area—`cm^2`
    - Volume—`cm^3`
    - Velocity—`cm/sec`

In J-Link, individual units in the model are represented by the interface `pfcUnits.Unit`.

Types of Unit Systems

The types of systems of units are as follows:

- Pre-defined system of units—This system of unit is provided by default.
- Custom-defined system of units—This system of unit is defined by the user only if the model does not contain standard metric or nonmetric units, or if the material file contains units that cannot be derived from the predefined system of units or both.

In PTC Creo Parametric, the system of units are categorized as follows:

- Mass Length Time (MLT)—The following systems of units belong to this category:
  - `CGS`—`Centimeter Gram Second`
  - `MKS`—`Meter Kilogram Second`
  - `mmKS`—`millimeter Kilogram Second`
- Force Length Time (FLT)—The following systems of units belong to this category:
  - PTC Creo Parametric Default—`Inch lbm Second`. This is the default system followed by PTC Creo Parametric.
  - `FPS`—`Foot Pound Second`
  - `IPS`—`Inch Pound Second`
  - `mmNS`—`Millimeter Newton Second`

In J-Link, the system of units followed by the model is represented by the interface `pfcUnits.UnitSystem`.

Accessing Individual Units

Methods Introduced:

- **pfcSolid.Solid.ListUnits**
- **pfcSolid.Solid.GetUnit**
- **pfcUnits.Unit.GetName**
- **pfcUnits.Unit.GetExpression**
- **pfcUnits.Unit.GetType**
- **pfcUnits.Unit.GetIsStandard**
- **pfcUnits.Unit.GetReferenceUnit**
- **pfcUnits.Unit.GetConversionFactor**
- **pfcUnits.UnitConversionFactor.GetOffset**
- **pfcUnits.UnitConversionFactor.GetScale**

The method `pfcSolid.Solid.ListUnits` returns the list of units available to the specified model.

The method `pfcSolid.Solid.GetUnit` retrieves the unit, based on its name or expression for the specified model in the form of the `pfcUnits.Unit` object.

The method `pfcUnits.Unit.GetName` returns the name of the unit.

The method `pfcUnits.Unit.GetExpression` returns a user-friendly unit description in the form of the name (for example,) for ordinary units and the expression (for example, `N/m^3`) for system-generated units.

The method `pfcUnits.Unit.GetType` returns the type of quantity represented by the unit in terms of the `pfcBase.UnitType` object. The types of units are as follows:

- `UNIT_LENGTH`—Specifies length measurement units.
- `UNIT_MASS`—Specifies mass measurement units.
- `UNIT_FORCE`—Specifies force measurement units.
- `UNIT_TIME`—Specifies time measurement units.
- `UNIT_TEMPERATURE`—Specifies temperature measurement units.
- `UNIT_ANGLE`—Specifies angle measurement units.

The method `pfcUnits.Unit.GetIsStandard` identifies whether the unit is system-defined (if the property *IsStandard* is set to true) or user-defined (if the property *IsStandard* is set to false).

The method `pfcUnits.Unit.GetReferenceUnit` returns a reference unit (one of the available system units) in terms of the `pfcUnits.Unit` object.

The method `pfcUnits.Unit.GetConversionFactor` identifies the relation of the unit to its reference unit in terms of the `pfcUnits.UnitConversionFactor` object. The unit conversion factors are as follows:

- `Offset`—Specifies the offset value applied to the values in the reference unit.
- `Scale`—Specifies the scale applied to the values in the reference unit to get the value in the actual unit.

```
Example - Consider the formula to convert temperature from Centigrade
 to Fahrenheit
F = a + (C * b)
where
F is the temperature in Fahrenheit
C is the temperature in Centigrade
a = 32 (constant signifying the offset value)
b = 9/5 (ratio signifying the scale of the unit)
```

Note

PTC Creo Parametric scales the length dimensions of the model using the factors listed above. If the scale is modified, the model is regenerated. When you scale the model, the model units are not changed. Imported geometry cannot be scaled.

Use the methods `pfcUnits.UnitConversionFactor.GetOffset` and `pfcUnits.UnitConversionFactor.GetScale` to retrieve the unit conversion factors listed above.

Modifying Individual Units

Methods Introduced:

- **pfcUnits.Unit.Modify**
- **pfcUnits.Unit.Delete**
- **pfcUnits.Unit.SetName**
- **pfcUnits.UnitConversionFactor.SetOffset**
- **pfcUnits.UnitConversionFactor.SetScale**

The method `pfcUnits.Unit.Modify` modifies the definition of a unit by applying a new conversion factor specified by the `pfcUnits.UnitConversionFactor` object and a reference unit.

The method `pfcUnits.Unit.Delete` deletes the unit.

Note

You can delete only custom units and not standard units.

The method `pfcUnits.Unit.SetName` modifies the name of the unit.

Use the methods `pfcUnits.UnitConversionFactor.SetOffset` and `pfcUnits.UnitConversionFactor.SetScale` to modify the unit conversion factors.

Creating a New Unit

Methods Introduced:

- **pfcSolid.Solid.CreateCustomUnit**
- **pfcUnits.pfcUnits.UnitConversionFactor_Create**

The method pfcSolid.Solid.CreateCustomUnit creates a custom unit based on the specified name, the conversion factor given by the pfcUnits.UnitConversionFactor object, and a reference unit.

The method pfcUnits.pfcUnits.UnitConversionFactor_Create creates the pfcUnits.UnitConversionFactor object containing the unit conversion factors.

Accessing Systems of Units

Methods Introduced:

- **pfcSolid.Solid.ListUnitSystems**
- **pfcSolid.Solid.GetPrincipalUnits**
- **pfcUnits.UnitSystem.GetUnit**
- **pfcUnits.UnitSystem.GetName**
- **pfcUnits.UnitSystem.GetType**
- **pfcUnits.UnitSystem.GetIsStandard**

The method pfcSolid.Solid.ListUnitSystems returns the list of unit systems available to the specified model.

The method pfcSolid.Solid.GetPrincipalUnits returns the system of units assigned to the specified model in the form of the pfcUnits.UnitSystem object.

The method pfcUnits.UnitSystem.GetUnit retrieves the unit of a particular type used by the unit system.

The method pfcUnits.UnitSystem.GetName returns the name of the unit system.

The method pfcUnits.UnitSystem.GetType returns the type of the unit system in the form of the pfcUnits.UnitSystemType object. The types of unit systems are as follows:

- UNIT_SYSTEM_MASS_LENGTH_TIME—Specifies the Mass Length Time (MLT) unit system.
- UNIT_SYSTEM_FORCE_LENGTH_TIME—Specifies the Force Length Time (FLT) unit system.

For more information on these unit systems listed above, refer to the section Types of Unit Systems on page 179.

The method pfcUnits.UnitSystem.GetIsStandard identifies whether the unit system is system-defined (if the property *IsStandard* is set to true) or user-defined (if the property *IsStandard* is set to false).

Modifying Systems of Units

Methods Introduced:

- **pfcUnits.UnitSystem.Delete**
- **pfcUnits.UnitSystem.SetName**

The method `pfcUnits.UnitSystem.Delete` deletes a custom-defined system of units.

> **Note**
>
> You can delete only a custom-defined system of units and not a standard system of units.

Use the method `pfcUnits.UnitSystem.SetName` to rename a custom-defined system of units. Specify the new name for the system of units as an input parameter for this function.

Creating a New System of Units

Method Introduced:

- **pfcSolid.Solid.CreateUnitSystem**

The method `pfcSolid.Solid.CreateUnitSystem` creates a new system of units in the model based on the specified name, the type of unit system given by the `pfcUnits.UnitSystemType` object, and the types of units specified by the `pfcUnits.Units` sequence to use for each of the base measurement types (length, force or mass, and temperature).

Conversion to a New Unit System

Methods Introduced:

- **pfcSolid.Solid.SetPrincipalUnits**
- **pfcUnits.pfcUnits.UnitConversionOptions_Create**
- **pfcUnits.UnitConversionOptions.SetDimensionOption**
- **pfcUnits.UnitConversionOptions.SetIgnoreParamUnits**

The method `pfcSolid.Solid.SetPrincipalUnits` changes the principal system of units assigned to the solid model based on the the unit conversion options specified by the `pfcUnits.UnitConversionOptions` object. The method `pfcUnits.pfcUnits.UnitConversionOptions_Create` creates the `pfcUnits.UnitConversionOptions` object containing the unit conversion options listed below.

The types of unit conversion options are as follows:

- `DimensionOption`—Use the option while converting the dimensions of the model.

  Use the method `pfcUnits.UnitConversionOptions.SetDimensionOption` to modify this option.

  This option can be of the following types:

  - `UNITCONVERT_SAME_DIMS`—Specifies that unit conversion occurs by interpreting the unit value in the new unit system. For example, 1 inch will equal to 1 millimeter.
  - `UNITCONVERT_SAME_SIZE`—Specifies that unit conversion will occur by converting the unit value in the new unit system. For example, 1 inch will equal to 25.4 millimeters.
- `IgnoreParamUnits`—This boolean attribute determines whether or not ignore the parameter units. If it is null or true, parameter values and units do not change when the unit system is changed. If it is false, parameter units are converted according to the rule.

  Use the method `pfcUnits.UnitConversionOptions.SetIgnoreParamUnits` to modify this attribute.

Mass Properties

Method Introduced:

- **pfcSolid.Solid.GetMassProperty**

The function `pfcSolid.Solid.GetMassProperty` provides information about the distribution of mass in the part or assembly. It can provide the information relative to a coordinate system datum, which you name, or the default one if you provide `null` as the name. It returns a class called `MassProperty`.

The class contains the following fields:

- The volume.
- The surface area.
- The density. The density value is 1.0, unless a material has been assigned.
- The mass.
- The center of gravity (COG).
- The inertia matrix.
- The inertia tensor.

- The inertia about the COG.
- The principal moments of inertia (the eigen values of the COG inertia).
- The principal axes (the eigenvectors of the COG inertia).

Example Code: Retrieving a Mass Property Object

The sample code in the file `pfcSolidMassPropExample.java` located at <creo_jlink_loadpoint>/jlink_appls/jlinkexamples retrieves a `MassProperty` object from a specified solid model. The solid's mass, volume, and center of gravity point are then printed.

Annotations

Methods Introduced:

- **pfcNote.Note.GetLines**
- **pfcNote.Note.SetLines**
- **pfcNote.Note.GetText**
- **pfcNote.Note.GetURL**
- **pfcNote.Note.SetURL**
- **pfcNote.Note.Display**
- **pfcNote.Note.Delete**
- **pfcNote.Note.GetOwner**

3D model notes are instance of ModelItem objects. They can be located and accessed using methods that locate model items in solid models, and downcast to the Note interface to use the methods in this section.

The method `pfcNote.Note.GetLines` returns the text contained in the 3D model note. The method `pfcNote.Note.SetLines` modifies the note text.

The method `pfcNote.Note.GetText` returns the the text of the solid model note. If you set the parameter *GiveParametersAsNames* to `TRUE`, then the text displays the parameter callouts with ampersands (&). If you set the parameter to `FALSE`, then the text displays the parameter values with no callout information.

The method `pfcNote.Note.GetURL` returns the URL stored in the 3D model note. The method `pfcNote.Note.SetURL` modifies the note URL.

The method `pfcNote.Note.Display` forces the display of the model note.

The method `pfcNote.Note.Delete` deletes a model note.

The method `pfcNote.Note.GetOwner` returns the solid model owner of the note.

Cross Sections

Methods Introduced:

- **pfcSolid.Solid.ListCrossSections**
- **pfcSolid.Solid.GetCrossSection**
- **pfcXSection.XSection.GetName**
- **pfcXSection.XSection.SetName**
- **pfcXSection.XSection.GetXSecType**
- **pfcXSection.XSection.Delete**
- **pfcXSection.XSection.Display**
- **pfcXSection.XSection.Regenerate**

The method `pfcSolid.Solid.ListCrossSections` returns a sequence of cross section objects represented by the Xsection interface. The method `pfcSolid.Solid.GetCrossSection` searches for a cross section given its name.

The method `pfcXSection.XSection.GetName` returns the name of the cross section in PTC Creo Parametric. The method `pfcXSection.XSection.SetName` modifies the cross section name.

The method `pfcXSection.XSection.GetXSecType` returns the type of cross section, that is planar or offset, and the type of item intersected by the cross section.

The method `pfcXSection.XSection.Delete` deletes a cross section.

The method `pfcXSection.XSection.Display` forces a display of the cross section in the window.

The method `pfcXSection.XSection.Regenerate` regenerates a cross section.

Materials

J-Link enables you to programmatically access the material types and properties of parts. Using the methods and properties described in the following sections, you can perform the following actions:

- Create or delete materials
- Set the current material
- Access and modify the material types and properties

Methods Introduced:

- **pfcPart.Material.Save**
- **pfcPart.Material.Delete**
- **pfcPart.Part.GetCurrentMaterial**
- **pfcPart.Part.SetCurrentMaterial**
- **pfcPart.Part.ListMaterials**
- **pfcPart.Part.CreateMaterial**
- **pfcPart.Part.RetrieveMaterial**

The method `pfcPart.Material.Save` writes to a material file that can be imported into any PTC Creo Parametric part.

The method `pfcPart.Material.Delete` removes material from the part.

The method `pfcPart.Part.GetCurrentMaterial` returns the currently assigned material for the part.

The method `pfcPart.Part.SetCurrentMaterial` sets the material assigned to the part.

Note

By default, while assigning a material to a sheetmetal part, the method `pfcPart.Part.SetCurrentMaterial` modifies the values of the sheetmetal properties such as Y factor and bend table according to the material file definition. This modification triggers a regeneration and a modification of the developed length calculations of the sheetmetal part. However, you can avoid this behavior by setting the value of the configuration option `material_update_smt_bend_table` to `never_replace`

The method `pfcPart.Part.SetCurrentMaterial` may change the model display, if the new material has a default appearance assigned to it.

The method may also change the family table, if the parameter `PTC_MATERIAL_NAME` is a part of the family table.

The method`pfcPart.Part.ListMaterials` returns a list of the materials available in the part.

The method `pfcPart.Part.CreateMaterial` creates a new empty material in the specified part.

The method `pfcPart.Part.RetrieveMaterial` imports a material file into the part. The name of the file read can be as either:

- <name>.mtl—Specifies the new material file format.
- <name>.mat—Specifies the material file format prior to Pro/ENGINEER Wildfire 3.0.

If the material is not already in the part database, pfcPart.Part.RetrieveMaterial adds the material to the database after reading the material file. If the material is already in the database, the function replaces the material properties in the database with those contained in the material file.

Accessing Material Types

Methods Introduced:

- **pfcPart.Material.GetStructuralMaterialType**
- **pfcPart.Material.SetStructuralMaterialType**
- **pfcPart.Material.GetThermalMaterialType**
- **pfcPart.Material.SetThermalMaterialType**
- **pfcPart.Material.GetSubType**
- **pfcPart.Material.SetSubType**
- **pfcPart.Material.GetPermittedSubTypes**

The method pfcPart.Material.GetStructuralMaterialType returns the material type for the structural properties of the material. The material types are as follows:

- MTL_ISOTROPIC—Specifies a a material with an infinite number of planes of material symmetry, making the properties equal in all directions.
- MTL_ORTHOTROPIC—Specifies a material with symmetry relative to three mutually perpendicular planes.
- MTL_TRANSVERSELY_ISOTROPIC—Specifies a material with rotational symmetry about an axis. The properties are equal for all directions in the plane of isotropy.

Use the method pfcPart.Material.SetStructuralMaterialType to set the material type for the structural properties of the material.

The method pfcPart.Material.GetThermalMaterialType returns the material type for the thermal properties of the material. The material types are as follows:

- MTL_ISOTROPIC—Specifies a material with an infinite number of planes of material symmetry, making the properties equal in all directions.
- MTL_ORTHOTROPIC—Specifies a material with symmetry relative to three mutually perpendicular planes.
- MTL_TRANSVERSELY_ISOTROPIC—Specifies a material with rotational symmetry about an axis. The properties are equal for all directions in the plane of isotropy.

Use the method pfcPart.Material.SetThermalMaterialType to set the material type for the thermal properties of the material.

The method pfcPart.Material.GetSubType returns the subtype for the MTL_ISOTROPIC material type.

Use the method pfcPart.Material.SetSubType to set the subtype for the MTL_ISOTROPIC material type.

Use the method pfcPart.Material.GetPermittedSubTypes to retrieve a list of the permitted string values for the material subtype.

Accessing Material Properties

The methods listed in this section enable you to access material properties.

Methods Introduced:

- **pfcPart.pfcPart.MaterialProperty_Create**
- **pfcPart.Material.GetPropertyValue**
- **pfcPart.Material.SetPropertyValue**
- **pfcPart.Material.SetPropertyUnits**
- **pfcPart.Material.RemoveProperty**
- **pfcPart.Material.GetDescription**
- **pfcPart.Material.SetDescription**
- **pfcPart.Material.GetFatigueType**
- **pfcPart.Material.SetFatigueType**
- **pfcPart.Material.GetPermittedFatigueTypes**
- **pfcPart.Material.GetFatigueMaterialType**
- **pfcPart.Material.SetFatigueMaterialType**
- **pfcPart.Material.GetPermittedFatigueMaterialTypes**
- **pfcPart.Material.GetFatigueMaterialFinish**
- **pfcPart.Material.SetFatigueMaterialFinish**
- **pfcPart.Material.GetPermittedFatigueMaterialFinishes**

- **pfcPart.Material.GetFailureCriterion**
- **pfcPart.Material.SetFailureCriterion**
- **pfcPart.Material.GetPermittedFailureCriteria**
- **pfcPart.Material.GetHardness**
- **pfcPart.Material.SetHardness**
- **pfcPart.Material.GetHardnessType**
- **pfcPart.Material.SetHardnessType**
- **pfcPart.Material.GetCondition**
- **pfcPart.Material.SetCondition**
- **pfcPart.Material.GetBendTable**
- **pfcPart.Material.SetBendTable**
- **pfcPart.Material.GetCrossHatchFile**
- **pfcPart.Material.SetCrossHatchFile**
- **pfcPart.Material.GetMaterialModel**
- **pfcPart.Material.SetMaterialModel**
- **pfcPart.Material.GetPermittedMaterialModels**
- **pfcPart.Material.GetModelDefByTests**
- **pfcPart.Material.SetModelDefByTests**

The method `pfcPart.pfcPart.MaterialProperty_Create` creates a new instance of a material property object.

All numerical material properties are accessed using the same set of APIs. You must provide a property type to indicate the property you want to read or modify.

The method `pfcPart.Material.GetPropertyValue` returns the value and the units of the material property.

Use the method `pfcPart.Material.SetPropertyValue` to set the value and units of the material property. If the property type does not exist for the material, then this method creates it.

Use the method `pfcPart.Material.SetPropertyUnits` to set the units of the material property.

Use the method `pfcPart.Material.RemoveProperty` to remove the material property.

Material properties that are non-numeric can be accessed via property-specific get and set methods.

The methods `pfcPart.Material.GetDescription` and `pfcPart.Material.SetDescription` return and setthe description string for the material respectively.

The methods pfcPart.Material.GetFatigueType and pfcPart.Material.SetFatigueType return and set the valid fatigue type for the material respectively.

Use the method pfcPart.Material.GetPermittedFatigueTypes to get a list of the permitted string values for the fatigue type.

The methods pfcPart.Material.GetFatigueMaterialType and pfcPart.Material.SetFatigueMaterialType return and set the class of material when determining the effect of the fatigue respectively.

Use the method pfcPart.Material.GetPermittedFatigueMaterialTypes to retrieve a list of the permitted string values for the fatigue material type.

The methods pfcPart.Material.GetFatigueMaterialFinish and pfcPart.Material.SetFatigueMaterialFinish return and set the type of surface finish for the fatigue material respectively.

Use the method pfcPart.Material.GetPermittedFatigueMaterialFinishes to retrieve a list of permitted string values for the fatigue material finish.

The method pfcPart.Material.GetFailureCriterion returnsthe reduction factor for the failure strength of the material. This factor is used to reduce the endurance limit of the material to account for unmodeled stress concentrations, such as those found in welds. Use the method pfcPart.Material.SetFailureCriterion to set the reduction factor for the failure strength of the material.

Use the method pfcPart.Material.GetPermittedFailureCriteria to retrieve a list of permitted string values for the material failure criterion.

The methods pfcPart.Material.GetHardness and pfcPart.Material.SetHardness return and set the hardness for the specified material respectively.

The methods pfcPart.Material.GetHardnessType and pfcPart.Material.SetHardnessType return and set the hardness type for the specified material respectively.

The methods pfcPart.Material.GetCondition and pfcPart.Material.GetCondition return and set the condition for the specified material respectively.

The methods pfcPart.Material.GetBendTable and pfcPart.Material.SetBendTable return and set the bend table for the specified material respectively.

The methods pfcPart.Material.GetCrossHatchFile and pfcPart.Material.SetCrossHatchFile return and set the file containing the crosshatch pattern for the specified material respectively.

The methods `pfcPart.Material.GetMaterialModel` and `pfcPart.Material.SetMaterialModel` return and set the type of hyperelastic isotropic material model respectively.

Use the method `pfcPart.Material.GetPermittedMaterialModels` to retrieve a list of the permitted string values for the material model.

The methods `pfcPart.Material.GetModelDefByTests` determines whether the hyperelastic isotropic material model has been defined using experimental data for stress and strain.

Use the method `pfcPart.Material.SetModelDefByTests` to define the hyperelastic isotropic material model using experimental data for stress and strain.

Accessing User-defined Material Properties

Materials permit assignment of user-defined parameters. These parameters allow you to place non-standard properties on a given material. Therefore `pfcPart.Material` is a child of `pfcModelItem.ParameterOwner`, which provides access to user-defined parameters and properties of materials through the methods in that interface.

13

Windows and Views

J-Link provides access to PTC Creo Parametric windows and saved views. This chapter describes the methods that provide this access.

Windows

This section describes the J-Link methods that access `Window` objects. The topics are as follows:

Getting a Window Object

Methods Introduced:

- **pfcSession.BaseSession.GetCurrentWindow**
- **pfcSession.BaseSession.CreateModelWindow**
- **pfcModel.Model.Display**
- **pfcSession.BaseSession.ListWindows**
- **pfcSession.BaseSession.GetWindow**
- **pfcSession.BaseSession.OpenFile**
- **pfcSession.BaseSession.GetModelWindow**

The method `pfcSession.BaseSession.GetCurrentWindow` provides access to the current active window in PTC Creo Parametric.

The method `pfcSession.BaseSession.CreateModelWindow` creates a new window that contains the model that was passed as an argument.

> **Note**
>
> You must call the method `pfcModel.Model.Display` for the model geometry to be displayed in the window.

Use the method `pfcSession.BaseSession.ListWindows` to get a list of all the current windows in session.

The method `pfcSession.BaseSession.GetWindow` gets the handle to a window given its integer identifier.

The method `pfcSession.BaseSession.OpenFile` returns the handle to a newly created window that contains the opened model.

> **Note**
>
> If a model is already open in a window the method returns a handle to the window.

The method `pfcSession.BaseSession.GetModelWindow` returns the handle to the window that contains the opened model, if it is displayed.

Window Operations

Methods Introduced:

- **pfcWindow.Window.GetHeight**
- **pfcWindow.Window.GetWidth**
- **pfcWindow.Window.GetXPos**
- **pfcWindow.Window.GetYPos**
- **pfcWindow.Window.GetGraphicsAreaHeight**
- **pfcWindow.Window.GetGraphicsAreaWidth**
- **pfcWindow.Window.Clear**
- **pfcWindow.Window.Repaint**
- **pfcWindow.Window.Refresh**
- **pfcWindow.Window.Close**
- **pfcWindow.Window.Activate**
- **pfcWindow.Window.GetId**
- **pfcSession.BaseSession.FlushCurrentWindow**

The methods `pfcWindow.Window.GetHeight`, `pfcWindow.Window.GetWidth`, `pfcWindow.Window.GetXPos`, and `pfcWindow.Window.GetYPos` retrieve the height, width, x-position, and y-position of the window respectively. The values of these parameters are normalized from 0 to 1.

The methods `pfcWindow.Window.GetGraphicsAreaHeight` and `pfcWindow.Window.GetGraphicsAreaWidth` retrieve the height and width of the PTC Creo Parametric graphics area window without the border respectively. The values of these parameters are normalized from 0 to 1. For both the window and graphics area sizes, if the object occupies the whole screen, the window size returned is 1. For example, if the screen is `1024 pixels` wide and the graphics area is `512 pixels`, then the width of the graphics area window is returned as 0.5.

The method `pfcWindow.Window.Clear` removes geometry from the window.

Both `pfcWindow.Window.Repaint` and `pfcWindow.Window.Refresh` repaint solid geometry. However, the `Refresh` method does not remove highlights from the screen and is used primarily to remove temporary geometry entities from the screen.

Use the method `pfcWindow.Window.Close` to close the window. If the current window is the original window created when PTC Creo Parametric started, this method clears the window. Otherwise, it removes the window from the screen.

The method `pfcWindow.Window.Activate` activates a window. This function is available only in the asynchronous mode.

The method `pfcWindow.Window.GetId` retrieves the ID of the PTC Creo Parametric window.

The method `pfcSession.BaseSession.FlushCurrentWindow` flushes the pending display commands on the current window.

Note

It is recommended to call this method only after completing all the display operations. Excessive use of this method will cause major slow down of systems running on Windows Vista and Windows 7.

Embedded Browser

Methods Introduced:

- **pfcWindow.Window.GetURL**
- **pfcWindow.Window.SetURL**
- **pfcWindow.Window.GetBrowserSize**
- **pfcWindow.Window.SetBrowserSize**

The methods `pfcWindow.Window.GetURL` and `pfcWindow.Window.SetURL` enables you to find and change the URL displayed in the embedded browser in the PTC Creo Parametric window.

The methods pfcWindow.Window.GetBrowserSize and pfcWindow.Window.SetBrowserSize enables you to find and change the size of the embedded browser in the PTC Creo Parametric window.

Note

The methods pfcWindow.Window.GetBrowserSize and pfcWindow.Window.SetBrowserSize are not supported if the browser is open in a separate window.

Views

This section describes the J-Link methods that access View objects. The topics are as follows:

- Getting a View Object on page 199
- View Operations on page 200

Getting a View Object

Methods Introduced:

- **pfcView.ViewOwner.RetrieveView**
- **pfcView.ViewOwner.GetView**
- **pfcView.ViewOwner.ListViews**
- **pfcView.ViewOwner.GetCurrentView**

Any solid model inherits from the interface ViewOwner. This will enable you to use these methods on any solid object.

The method pfcView.ViewOwner.RetrieveView sets the current view to the orientation previously saved with a specified name.

Use the method pfcView.ViewOwner.GetView to get a handle to a named view without making any modifications.

The method pfcView.ViewOwner.ListViews returns a list of all the views previously saved in the model.

From Creo Parametric 2.0 M120 onward, the method, pfcView.ViewOwner.GetCurrentView has been deprecated. The method returns a view handle that represents the current orientation. Although this view does not have a name, you can use this view to find or modify the current orientation.

View Operations

Methods Introduced:

- **pfcView.View.GetName**
- **pfcView.View.GetIsCurrent**
- **pfcView.View.Reset**
- **pfcView.ViewOwner.SaveView**

To get the name of a view given its identifier, use the method `pfcView.View.GetName`.

The method `pfcView.View.GetIsCurrent` determines if the View object represents the current view.

The `pfcView.View.Reset` method restores the current view to the default view.

To store the current view under the specified name, call the method `pfcView.ViewOwner.SaveView`.

Coordinate Systems and Transformations

his section describes the various coordinate systems used by PTC Creo Parametric and accessible from J-Link and how to transform from one coordinate system to another.

Coordinate Systems

PTC Creo Parametric and J-Link use the following coordinate systems:

- Solid Coordinate System on page 201
- Screen Coordinate System on page 201
- Window Coordinate System on page 201
- Drawing Coordinate System on page 202
- Drawing View Coordinate System on page 202
- Assembly Coordinate System on page 202
- Datum Coordinate System on page 202
- Section Coordinate System on page 202

The following sections describe each of these coordinate systems.

Solid Coordinate System

The solid coordinate system is the three-dimensional, Cartesian coordinate system used to describe the geometry of a PTC Creo Parametric solid model. In a part, the solid coordinate system describes the geometry of the surfaces and edges. In an assembly, the solid coordinate system also describes the locations and orientations of the assembly members.

You can visualize the solid coordinate system in PTC Creo Parametric by creating a coordinate system datum with the option **Default**. Distances measured in solid coordinates correspond to the values of dimensions as seen by the PTC Creo Parametric user.

Solid coordinates are used by J-Link for all the methods that look at geometry and most of the methods that draw three-dimensional graphics.

Screen Coordinate System

The screen coordinate system is two-dimensional coordinate system that describes locations in a PTC Creo Parametric window. This is an intermediate coordinate system after which the screen points are transformed to screen pixels. All the models are first mapped to the screen coordinate system. When the user zooms or pans the view, the screen coordinate system follows the display of the solid, so a particular point on the solid always maps to the same screen coordinate. The mapping changes only when the view orientation is changed.

Screen coordinates are used by some of the graphics methods, the mouse input methods, and all methods that draw graphics or manipulate items on a drawing.

Window Coordinate System

The window coordinate system is similar to the screen coordinate system. After mapping the models to the screen coordinate system, they are mapped to the window coordinate before being drawn to screen pixels based on screen resolution. When pan or zoom values are applied to the coordinates in the screen coordinate system, they result in window coordinates. When an object is first displayed in a window, or the option **View ▸ Refit** is used, the screen and window coordinates are the same.

Window coordinates are needed only if you need to take account of zoom and pan —for example, to find out whether a point on the solid is visible in the window, or to draw two-dimensional text in a particular window location, regardless of pan and zoom.

Drawing Coordinate System

The drawing coordinate system is a two-dimensional system that describes the location on a drawing relative to the bottom, left corner, and measured in drawing units. For example, on a U.S. letter-sized, landscape-format drawing sheet that uses inches, the top, right-corner is (11, 8.5) in drawing coordinates.

The J-Link methods and properties that manipulate drawings generally use screen coordinates.

Drawing View Coordinate System

The drawing view coordinate system is used to describe the locations of entities in a drawing view.

Assembly Coordinate System

An assembly has its own coordinate system that describes the positions and orientations of the member parts, subassemblies, and the geometry of datum features created in the assembly.

When an assembly is retrieved into memory each member is also loaded and continues to use its own solid coordinate system to describe its geometry.

This is important when you are analyzing the geometry of a subassembly and want to extract or display the results relative to the coordinate system of the parent assembly.

Datum Coordinate System

A coordinate system datum can be created anywhere in any part or assembly, and represents a user-defined coordinate system. It is often a requirement in a J-Link application to describe geometry relative to such a datum.

Section Coordinate System

Every sketch has a coordinate system used to locate entities in that sketch. Sketches used in features will use a coordinate system different from that of the solid model.

Transformations

Methods Introduced:

- **pfcBase.Transform3D.Invert**
- **pfcBase.Transform3D.TransformPoint**
- **pfcBase.Transform3D.TransformVector**

- **pfcBase.Transform3D.GetMatrix**
- **pfcBase.Transform3D.SetMatrix**
- **pfcBase.Transform3D.GetOrigin**
- **pfcBase.Transform3D.GetXAxis**
- **pfcBase.Transform3D.GetYAxis**
- **pfcBase.Transform3D.GetZAxis**

All coordinate systems are treated in J-Link as if they were three-dimensional. Therefore, a point in any of the coordinate systems is always represented by the `pfcBase.Point3D` class:

Vectors store the same data but are represented for clarity by the `pfcBase.Vector3D` class.

Screen coordinates contain a z-value whose positive direction is outwards from the screen. The value of z is not generally important when specifying a screen location as an input to a method, but it is useful in other situations. For example, if you select a datum plane, you can find the direction of the plane by calculating the normal to the plane, transforming to screen coordinates, then looking at the sign of the z-coordinate.

A transformation between two coordinate systems is represented by the `IpfcBase.Transform3D` class. This class contains a 4x4 matrix that combines the conventional 3x3 matrix that describes the relative orientation of the two systems, and the vector that describes the shift between them.

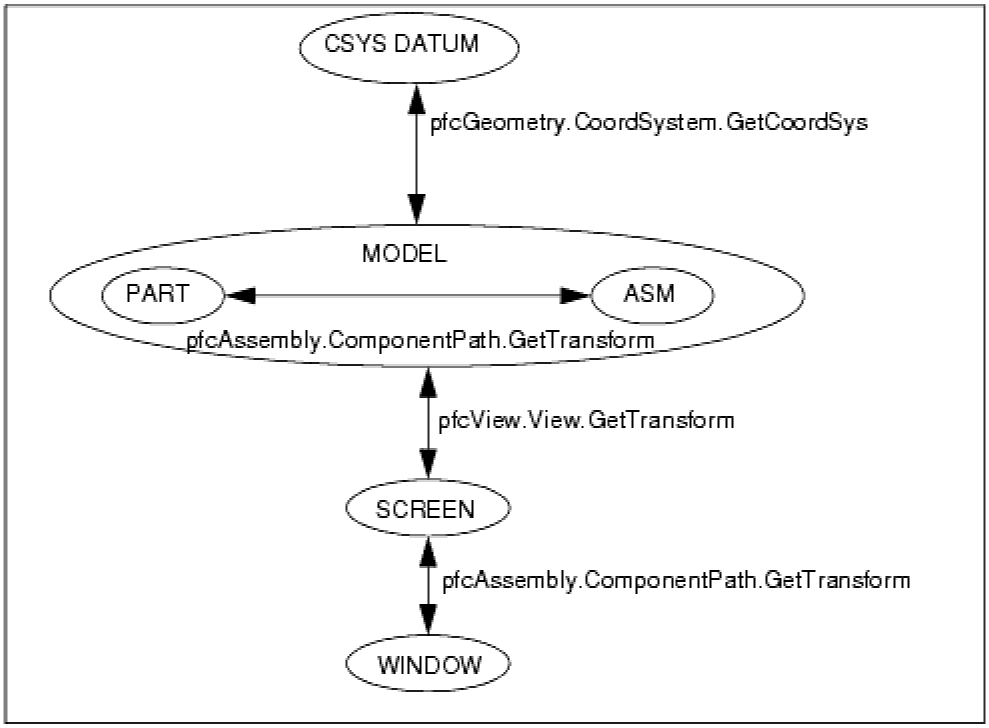
The 4x4 matrix used for transformations is as follows:

$$\begin{bmatrix} X' & Y' & Z' & 1 \end{bmatrix} = \begin{bmatrix} X & Y & Z & 1 \end{bmatrix} \begin{bmatrix} \ldots & \ldots & \ldots & 0 \\ \ldots & \ldots & \ldots & 0 \\ \ldots & \ldots & \ldots & 0 \\ X_s & Y_s & Z_s & 1 \end{bmatrix}$$

The utility method `ptcBase.Transform3D.Invert` inverts a transformation matrix so that it can be used to transform points in the opposite direction.

J-Link provides two utilities for performing coordinate transformations. The method `ptcBase.Transform3D.TransformPoint` transforms a three-dimensional point and `ptcBase.Transform3D.TransformVector` transforms a three-dimensional vector.

The following diagram summarizes the coordinate transformations needed when using J-Link and specifies the J-Link methods that provide the transformation matrix.



Transforming to Screen Coordinates

Methods Introduced:

- **pfcView.View.GetTransform**
- **pfcView.View.SetTransform**
- **pfcView.View.Rotate**

The view matrix describes the transformation from solid to screen coordinates. The method `pfcView.View.GetTransform` provides the view matrix for the specified view. The method `pfcView.View.SetTransform` allows you to specify a matrix for the view.

The method `pfcView.View.Rotate` rotates a view, relative to the X, Y, or Z axis, in the amount that you specifiy.

To transform from screen to solid coordinates, invert the transformation matrix using the method `pfcBase.Transform3D.Invert`.

Transforming to Coordinate System Datum Coordinates

Method Introduced:

- **pfcGeometry.CoordSystem.GetCoordSys**

The method `pfcGeometry.CoordSystem.GetCoordSys` provides the location and orientation of the coordinate system datum in the coordinate system of the solid that contains it. The location is in terms of the directions of the three axes and the position of the origin.

Transforming Window Coordinates

MethodsIntroduced

- **pfcWindow.Window.GetScreenTransform**
- **pfcWindow.Window.SetScreenTransform**
- **pfcBase.ScreenTransform.SetPanX**
- **pfcBase.ScreenTransform.SetPanY**
- **pfcBase.ScreenTransform.SetZoom**

You can alter the pan and zoom of a window by using a Screen Transform object. This object contains three attributes. PanX and PanY represent the horizontal and vertical movement. Every increment of 1.0 moves the view point one screen width or height. Zoom represents a scaling factor for the view. This number must be greater than zero.

Transforming Coordinates of an Assembly Member

Method Introduced:

- **pfcAssembly.ComponentPath.GetTransform**

The method `pfcAssembly.ComponentPath.GetTransform` provides the matrix for transforming from the solid coordinate system of the assembly member to the solid coordinates of the parent assembly, or the reverse.

The method `viewTransfer` accepts two views and transfers the matrix from the first to the second. This matrix is normalized using the second method, `matrixNormalize`.

Views can be changed to a normalized matrix only. The example method `UtilMatrixNormalize` takes a `Matrix3D` object and normalizes it.

Note

Both of these methods are declared to throw the exception `jxthrowable`. You need to put your error-handling code in the methods that call the utility methods.

14

ModelItem

This chapter describes the J-Link methods that enable you to access and manipulate `ModelItems`.

Solid Geometry Traversal

Solid models are made up of 11 distinct types of `ModelItem`, as follows:

- `pfcFeature.Feature`
- `pfcGeometry.Surface`
- `pfcGeometry.Edge`
- `pfcGeometry.Curve` (datum curve)
- `pfcGeometry.Axis` (datum axis)
- `pfcGeometry.Point` (datum point)
- `pfcGeometry.Quilt` (datum quilt)
- `pfcLayer.Layer`
- `pfcNote.Note`
- `pfcDimension.Dimension`
- `pfcDimension.RefDimension`

Each model item is assigned a unique identification number that will never change. In addition, each model item can be assigned a string name. Layers, points, axes, dimensions, and reference dimensions are automatically assigned a name that can be changed.

Getting ModelItem Objects

Methods Introduced:

- **pfcModelItem.ModelItemOwner.ListItems**
- **pfcFeature.Feature.ListSubItems**
- **pfcLayer.Layer.ListItems**
- **pfcModelItem.ModelItemOwner.GetItemById**
- **pfcModelItem.ModelItemOwner.GetItemByName**
- **pfcFamily.FamColModelItem.GetRefItem**
- **pfcSelect.Selection.GetSelItem**

All models inherit from the `ModelItemOwner`. The method `pfcModelItem.ModelItemOwner.ListItems` returns a sequence of `ModelItems` contained in the model. You can specify which type of `ModelItem` to collect by passing in one of the enumerated `ModelItemType` objects, or you can collect all `ModelItems` by passing `null` as the model item type.

Note

The part modeling features introduced in Creo Parametric 1.0 will be excluded from the list of features returned by the method `pfcModelItem.ModelItemOwner.ListItems` if the model item type is specified as`ITEM_FEATURE`. For example edit round features, flexible modeling features, and so on will be excluded from the list.

The methods `pfcFeature.Feature.ListSubItems` and `pfcLayer.Layer.ListItems` produce similar results for specific features and layers. These methods return a list of subitems in the feature or items in the layer.

To access specific model items, call the method `pfcModelItem.ModelItemOwner.GetItemById`. This methods enables you to access the model item by identifier.

To access specific model items, call the method `pfcModelItem.ModelItemOwner.GetItemByName`. This methods enables you to access the model item by name.

The method `pfcFamily.FamColModelItem.GetRefItem` returns the dimension or feature used as a header for a family table.

The method `pfcSelect.Selection.GetSelItem` returns the item selected interactively by the user.

ModelItem Information

Methods Introduced:

- **pfcModelItem.ModelItem.GetName**
- **pfcModelItem.ModelItem.SetName**
- **pfcModelItem.ModelItem.GetId**
- **pfcModelItem.ModelItem.GetType**

Certain `ModelItems` also have a string name that can be changed at any time. The methods `GetName` and `SetName` access this name.

The method `Id` returns the unique integer identifier for the `ModelItem`.

The `Type` method returns an enumeration object that indicates the model item type of the specified `ModelItem`. See the section Solid Geometry Traversal on page 208 for the list of possible model item types.

Duplicating ModelItems

Methods Introduced:

- **pfcSession.BaseSession.AllowDuplicateModelItems**

You can control the creation of ModelItems more than twice for the same PTC Creo Parametric item. The method `pfcSession.BaseSession.AllowDuplicateModelItems` allows you to turn ON or OFF the option to duplicate model items. By default, this option is OFF. To turn the option ON, set the boolean value to `FALSE`.

> **Note**
>
> If this option is not handled properly on the application side, it can cause memory corruption. Thus, althought you can turn ON and OFF this option as many times as you want, PTC recommends turning ON and OFF this option only once, right after the session is obtained.

Layer Objects

In J-Link, layers are instances of `ModelItem`. The following sections describe how to get layer objects and the operations you can perform on them.

Getting Layer Objects

Method Introduced:

- **pfcModel.Model.CreateLayer**

The method `pfcModel.Model.CreateLayer` returns a new layer with the name you specify.

See the section Getting ModelItem Objects on page 208 for other methods that can return layer objects.

Layer Operations

Methods Introduced:

- **pfcLayer.Layer.GetStatus**
- **pfcLayer.Layer.SetStatus**
- **pfcLayer.Layer.ListItems**
- **pfcLayer.Layer.AddItem**

- **pfcLayer.Layer.RemoveItem**
- **pfcLayer.Layer.Delete**
- **pfcLayer.Layer.CountUnsupportedItems**

Superseded Method:

- **pfcLayer.Layer.HasUnsupportedItems**

The methods `pfcLayer.Layer.GetStatus` and `pfcLayer.Layer.SetStatus` enables you to access the display status of a layer. The corresponding enumeration class is `DisplayStatus` and the possible values are `Normal`, `Displayed`, `Blank`, or `Hidden`.

Use the methods `pfcLayer.Layer.ListItems`, `pfcLayer.Layer.AddItem`, and `pfcLayer.Layer.RemoveItem` to control the contents of a layer.

Note

You cannot add the following items to a layer:

- `ITEM_SURFACE`,
- `ITEM_EDGE`,
- `ITEM_COORD_SYS`,
- `ITEM_AXIS`,
- `ITEM_SIMPREP`,
- `ITEM_DTL_SYM_DEFINITION`,
- `ITEM_DTL_OLE_OBJECT`,
- `ITEM_EXPLODED_STATE`.

For these items the method will throw the exception `pfcExceptions.XToolkitInvalidType`.

The method `pfcLayer.Layer.Delete` removes the layer (but not the items it contains) from the model.

The method `pfcLayer.Layer.CountUnsupportedItems` returns the number of item types not supported as a `pfcModelItem` object in the specified layer. This method deprecates the method `pfcLayer::HasUnsupportedItems`.

15

Features

All PTC Creo Parametric solid models are made up of features. This chapter describes how to program on the feature level using J-Link.

Access to Features

Methods Introduced:

- **pfcFeature.Feature.ListChildren**
- **pfcFeature.Feature.ListParents**
- **pfcFeature.FeatureGroup.GetGroupLeader**
- **pfcFeature.FeaturePattern.GetPatternLeader**
- **pfcFeature.FeaturePattern.ListMembers**
- **pfcSolid.Solid.ListFailedFeatures**
- **pfcSolid.Solid.ListFeaturesByType**
- **pfcSolid.Solid.GetFeatureById**

The methods `pfcFeature.Feature.ListChildren` and `pfcFeature.Feature.ListParents` return a sequence of features that contain all the children or parents of the specified feature.

To get the first feature in the specified group access the method `pfcFeature.FeatureGroup.GetGroupLeader`.

The methods `pfcFeature.FeaturePattern.GetPatternLeader` and the method `pfcFeature.FeaturePattern.ListMembers` return features that make up the specified feature pattern. See the section Feature Groups and Patterns on page 218 for more information on feature patterns.

The method `pfcSolid.Solid.ListFailedFeatures` returns a sequence that contains all the features that failed regeneration.

The method `pfcSolid.Solid.ListFeaturesByType` returns a sequence of features contained in the model. You can specify which type of feature to collect by passing in one of the `FeatureType` enumeration objects, or you can collect all features by passing `void null` as the type. If you list all features, the resulting sequence will include invisible features that PTC Creo Parametric creates internally. Internal features are invisible features used internally for construction purposes. Use the method's *VisibleOnly* argument to exclude them. If the argument *VisibleOnly* is `True`, the function lists the public features only. If the argument is `False`, the function lists both public and internal features.

The method `pfcSolid.Solid.GetFeatureById` returns the feature object with the corresponding integer identifier.

Feature Information

Methods Introduced:

- **pfcFeature.Feature.GetFeatType**
- **pfcFeature.Feature.GetStatus**

- **pfcFeature.Feature.GetIsVisible**
- **pfcFeature.Feature.GetIsReadonly**
- **pfcFeature.Feature.GetIsEmbedded**
- **pfcFeature.Feature.GetNumber**
- **pfcFeature.Feature.GetFeatTypeName**
- **pfcFeature.Feature.GetFeatSubType**
- **pfcRoundFeat.RoundFeat.GetIsAutoRoundMember**

The enumeration classes `FeatureType` and `FeatureStatus` provide information for a specified feature. The following methods specify this information:

- `pfcFeature.Feature.GetFeatType`—Returns the type of a feature.
- `pfcFeature.Feature.GetStatus`—Returns whether the feature is suppressed, active, or failed regeneration.

The other methods that gather feature information include the following:

- `pfcFeature.Feature.GetIsVisible`—Identifies whether the specified feature will be visible on the screen. The method distinguishes visible features from internal features. Internal features are invisible features used for construction purposes.
- `pfcFeature.Feature.GetIsReadonly`—Identifies whether the specified feature can be modified.
- `pfcFeature.Feature.GetIsEmbedded`—Specifies whether the specified feature is an embedded datum.
- `pfcFeature.Feature.GetNumber`—Returns the feature regeneration number. This method returns `void null` if the feature is suppressed.

The method `pfcFeature.Feature.GetFeatTypeName` returns a string representation of the feature type.

The method `pfcFeature.Feature.GetFeatSubType` returns a string representation of the feature subtype, for example, "Extrude" for a protrusion feature.

The method `pfcRoundFeat.RoundFeat.GetIsAutoRoundMember` determines whether the specified round feature is a member of an Auto Round feature.

Feature Operations

Methods Introduced:

- **pfcSolid.Solid.ExecuteFeatureOps**
- **pfcFeature.Feature.CreateSuppressOp**
- **pfcFeature.SuppressOperation.SetClip**
- **pfcFeature.SuppressOperation.SetAllowGroupMembers**
- **pfcFeature.SuppressOperation.SetAllowChildGroupMembers**
- **pfcFeature.Feature.CreateDeleteOp**
- **pfcFeature.DeleteOperation.SetClip**
- **pfcFeature.DeleteOperation.SetAllowGroupMembers**
- **pfcFeature.DeleteOperation.SetAllowChildGroupMembers**
- **pfcFeature.DeleteOperation.SetKeepEmbeddedDatums**
- **pfcFeature.Feature.CreateResumeOp**
- **pfcFeature.ResumeOperation.SetWithParents**
- **pfcFeature.Feature.CreateReorderBeforeOp**
- **pfcFeature.ReorderBeforeOperation.SetBeforeFeat**
- **pfcFeature.Feature.CreateReorderAfterOp**
- **pfcFeature.ReorderAfterOperation.SetAfterFeat**
- **pfcFeature.FeatureOperations.create**

The method `pfcSolid.Solid.ExecuteFeatureOps` causes a sequence of feature operations to run in order. Feature operations include suppressing, resuming, reordering, and deleting features. The optional `RegenInstructions` argument specifies whether the user will be allowed to fix the model if a regeneration failure occurs.

Note

The method `pfcSolid.Solid.ExecuteFeatureOps` is not supported in the No-Resolve mode, introduced in Pro/ENGINEER Wildfire 5.0. It throws an exception `pfcExceptions.XToolkitBadContext`. To continue with the Pro/ENGINEER Wildfire 4.0 behavior in the Resolve mode, set the configuration option `regen_failure_handling` to `resolve_mode` in the PTC Creo Parametric session. Refer to the Solid Operations on page 177 section in the Solid on page 175 chapter for more information on the No-Resolve mode.

You can create an operation that will delete, suppress, reorder, or resume certain features using the methods in the interface `pfcFeature.Feature`. Each created operation must be passed as a member of the `FeatureOperations` object to the method `pfcSolid.Solid.ExecuteFeatureOps`. You can create a sequence of the `FeatureOperations` object using the method `pfcFeature.FeatureOperations.create`.

Some of the operations have specific options that you can modify to control the behavior of the operation:

- `Clip`—Specifies whether to delete or suppress all features after the selected feature. By default, this option is false.

  Use the methods `pfcFeature.DeleteOperation.SetClip` and `pfcFeature.SuppressOperation.SetClip` to modify this option.
- `AllowGroupMembers`—If this option is set to true and if the feature to be deleted or suppressed is a member of a group, then the feature will be deleted or suppressed out of the group. If this option is set to false, then the entire group containing the feature is deleted or suppressed. By default, this option is false. It can be set to true only if the option `Clip` is set to true.

  Use the methods `pfcFeature.SuppressOperation.SetAllowGroupMembers` and `pfcFeature.DeleteOperation.SetAllowGroupMembers` to modify this option.
- `AllowChildGroupMembers`—If this option is set to true and if the children of the feature to be deleted or suppressed are members of a group, then the children of the feature will be individually deleted or suppressed out of the group. If this option is set to false, then the entire group containing the feature and its children is deleted or suppressed. By default, this option is false. It can be set to true only if the options `Clip` and `AllowGroupMembers` are set to true.

  Use the methods `pfcFeature.SuppressOperation.SetAllowChildGroupMembers` and `pfcFeature.DeleteOperation.SetAllowChildGroupMembers` to modify this option.
- `KeepEmbeddedDatums`—Specifies whether to retain the embedded datums stored in a feature while deleting the feature. By default, this option is false.

  Use the method `pfcFeature.DeleteOperation.SetKeepEmbeddedDatums` to modify this option.
- `WithParents`—Specifies whether to resume the parents of the selected feature.

  Use the method `pfcFeature.ResumeOperation.SetWithParents` to modify this option.

- BeforeFeat—Specifies the feature before which you want to reorder the features.

  Use the method pfcFeature.ReorderBeforeOperation.SetBeforeFeat to modify this option.
- AfterFeat—Specifies the feature after which you want to reorder the features.

  Use the method pfcFeature.ReorderAfterOperation.SetAfterFeat to modify this option.

Feature Groups and Patterns

Patterns are treated as features in PTC Creo Parametric. A feature type, FEATTYPE_PATTERN_HEAD, is used for the pattern header feature.

The result of the pattern header feature for users of previous versions of J-Link is as follows:

- Models that contain patterns get one extra feature of type FEATTYPE_PATTERN_HEAD in the regeneration list. This changes the feature numbers of all subsequent features, including those in the pattern.

Note

The pattern header feature is not treated as a leader or a member of the pattern by the methods described in the following section.

Methods Introduced:

- **pfcFeature.Feature.GetGroup**
- **pfcFeature.Feature.GetPattern**
- **pfcSolid.Solid.CreateLocalGroup**
- **pfcFeature.FeatureGroup.GetPattern**
- **pfcFeature.FeatureGroup.GetGroupLeader**
- **pfcFeature.FeaturePattern.GetPatternLeader**
- **pfcFeature.FeaturePattern.ListMembers**
- **pfcFeature.FeaturePattern.Delete**

The method pfcFeature.Feature.GetGroup returns a handle to the local group that contains the specified feature.

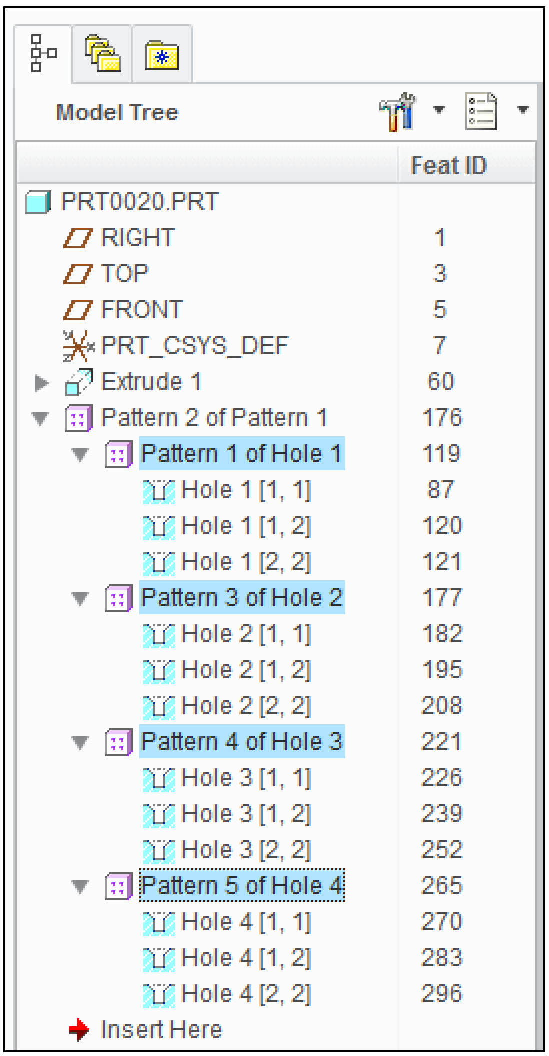
To get the first feature in the specified group call the method `pfcFeature.FeatureGroup.GetGroupLeader`.

The methods `pfcFeature.FeaturePattern.GetPatternLeader` and `pfcFeature.FeaturePattern.ListMembers` return features that make up the specified feature pattern.

A pattern is composed of a pattern header feature and a number of member features. You can pattern only a single feature. To pattern several features, create a local group and pattern this group.

You can also create a pattern of pattern. This creates a multiple level pattern. From Creo Parametric 2.0 M170 onward, for a pattern of pattern, the method `pfcFeature.FeaturePattern.ListMembers` returns all the pattern header features created at the first level.

For example, consider a model where a pattern of pattern has been created. The model tree is as shown below:



The method pfcFeature.FeaturePattern.ListMembers returns the pattern header features with following IDs for a pattern of pattern:

- 119
- 177
- 221
- 265

The methods pfcFeature.Feature.GetPattern and pfcFeature.FeatureGroup.GetPattern return the FeaturePattern object that contains the corresponding Feature or FeatureGroup. Use the method pfcSolid.Solid.CreateLocalGroup to take a sequence of features and create a local group with the specified name. To delete a FeaturePattern object, call the method pfcFeature.FeaturePattern.Delete.

Changes To Feature Groups

Beginning in Revision 2000i², the structure of feature groups is different than in previous releases. Feature groups now have a group header feature, which shows up in the model information and feature list for the model. This feature will be inserted in the regeneration list to a position just before the first feature in the group. Existing models, when retrieved into Revision 2000i², will have their groups automatically updated to this structure upon retrieval.

The results of these changes are as follows:

- Models that contain groups will get one extra feature in the regeneration list, of type FeatureType.FEATTYPE_GROUP_HEAD. This will change the feature numbers of all subsequent features, including those in the group.
- Each group automatically contains one new feature in the list of features returned from pfcFeature.FeatureGroup.ListMembers.
- Each group automatically gets a different leader feature (the group head feature is the leader). This is returned from pfcFeature.FeatureGroup.GetGroupLeader.
- Each group pattern contains a series of groups, and each group in the pattern will be similarly altered.

User Defined Features

Groups in PTC Creo Parametric represent sets of contiguous features that act as a single feature for specific operations. Individual features are affected by most operations while some operations apply to an entire group:

- Suppress
- Delete
- Layers
- Patterning

User defined Features (UDFs) are groups of features that are stored in a file. When a UDF is placed in a new model the created features are automatically assigned to a group. A local group is a set of features that have been specifically assigned to a group to make modifications and patterning easier.

Note

All methods in this section can be used for UDFs and local groups.

Read Access to Groups and User Defined Features

Methods Introduced:

- **pfcFeature.FeatureGroup.GetUDFName**
- **pfcFeature.FeatureGroup.GetUDFInstanceName**
- **pfcFeature.FeatureGroup.ListUDFDimensions**
- **pfcUDFGroup.UDFDimension.GetUDFDimensionName**

User defined features (UDF's) are groups of features that can be stored in a file and added to a new model. A local group is similar to a UDF except it is available only in the model in which is was created.

The method `pfcFeature.FeatureGroup.GetUDFName` provides the name of the group for the specified group instance. A particular group definition can be used more than once in a particular model.

If the group is a family table instance, the method `pfcFeature.FeatureGroup.GetUDFInstanceName` supplies the instance name.

The method `pfcFeature.FeatureGroup.ListUDFDimensions` traverses the dimensions that belong to the UDF. These dimensions correspond to the dimensions specified as variables when the UDF was created. Dimensions of the original features that were not variables in the UDF are not included unless the UDF was placed using the `Independent` option.

The method pfcUDFGroup.UDFDimension.GetUDFDimensionName provides access to the dimension name specified when the UDF was created, and not the name of the dimension in the current model. This name is required to place the UDF programmatically using the method pfcSolid.Solid.CreateUDFGroup.

Creating Features from UDFs

Method Introduced:

- **pfcSolid.Solid.CreateUDFGroup**

The method pfcSolid.Solid.CreateUDFGroup is used to create new features by retrieving and applying the contents of an existing UDF file. It is equivalent to the PTC Creo Parametric command Feature, Create, User Defined.

To understand the following explanation of this method, you must have a good knowledge and understanding of the use of UDF's in PTC Creo Parametric. PTC recommends that you read about UDF's in the PTC Creo Parametric help, and practice defining and using UDF's in PTC Creo Parametric before you attempt to use this method.

When you create a UDF interactively, PTC Creo Parametric prompts you for the information it needs to fix the properties of the resulting features. When you create a UDF from J-Link, you can provide some or all of this information programmatically by filling several compact data classes that are inputs to the method pfcSolid.Solid.CreateUDFGroup.

During the call to pfcSolid.Solid.CreateUDFGroup, PTC Creo Parametric prompts you for the following:

- Information required by the UDF that was not provided in the input data structures.
- Correct information to replace erroneous information

Such prompts are a useful way of diagnosing errors when you develop your application. This also means that, in addition to creating UDF's programmatically to provide automatic synthesis of model geometry, you can also use pfcSolid.Solid.CreateUDFGroup to create UDF's semi-interactively. This can simplify the interactions needed to place a complex UDF making it easier for the user and less prone to error.

Creating UDFs

Creating a UDF requires the following information:

- Name—The name of the UDF you are creating and the instance name if applicable.
- Dependency—Specify if the UDF is independent of the UDF definition or is modified by the changers made to it.
- Scale—How to scale the UDF relative to the placement model.
- Variable Dimension—The new values of the variables dimensions and pattern parameters, those whose values can be modified each time the UDF is created.
- Dimension Display—Whether to show or blank non-variable dimensions created within the UDF group.
- References—The geometrical elements that the UDF needs in order to relate the features it contains to the existing models features. The elements correspond to the picks that PTC Creo Parametric prompts you for when you create a UDF interactively using the prompts defined when the UDF was created. You cannot select an embedded datum as the UDF reference.
- Parts Intersection—When a UDF that is being created in an assembly contains features that modify the existing geometry you must define which parts are affected or intersected. You also need to know at what level in an assembly each intersection is going to be visible.
- Orientations—When a UDF contains a feature with a direction that is defined in respect to a datum plane PTC Creo Parametric must know what direction the new feature will point to. When you create such a UDF interactively PTC Creo Parametric prompt you for this information with a flip arrow.
- Quadrants—When a UDF contains a linearly placed feature that references two datum planes to define it's location in the new model PTC Creo Parametric prompts you to pick the location of the new feature. This is determined by which side of each datum plane the feature must lie. This selection is referred to as the quadrant because the are four possible combinations for each linearly place feature.

To pass all the above values to PTC Creo Parametric, J-Link uses a special class that prepares and sets all the options and passes them to PTC Creo Parametric.

Creating Interactively Defined UDFs

Method Introduced:

- **pfcUDFGroup.pfcUDFGroup.UDFPromptCreateInstructions_Create**

This static method is used to create an instructions object that can be used to prompt a user for the required values that will create a UDF interactively.

Creating a Custom UDF

Method Introduced:

- **pfcUDFCreate.pfcUDFCreate.UDFCustomCreateInstructions_Create**

This method creates a `UDFCustomCreateInstructions` object with a specified name. To set the UDF creation parameters programmatically you must modify this object as described below. The members of this class relate closely to the prompts PTC Creo Parametric gives you when you create a UDF interactively. PTC recommends that you experiment with creating the UDF interactively using PTC Creo Parametric before you write the J-Link code to fill the structure.

Setting the Family Table Instance Name

Methods Introduced:

- **pfcUDFCreate.UDFCustomCreateInstructions.SetInstanceName**
- **pfcUDFCreate.UDFCustomCreateInstructions.GetInstanceName**

If the UDF contains a family table, this field can be used to select the instance in the table. If the UDF does not contain a family table, or if the generic instance is to be selected, the do not set the string.

Setting Dependency Type

Methods Introduced:

- **pfcUDFCreate.UDFCustomCreateInstructions.SetDependencyType**
- **pfcUDFCreate.UDFCustomCreateInstructions.GetDependencyType**

The `UDFDependencyType` object represents the dependency type of the UDF. The choices correspond to the choices available when you create a UDF interactively. This enumerated type takes the following values:

- `UDFDEP_INDEPENDENT`
- `UDFDEP_DRIVEN`

Note

`UDFDEP_INDEPENDENT` is the default value, if this option is not set.

Setting Scale and Scale Type

Methods Introduced:

- **pfcUDFCreate.UDFCustomCreateInstructions.SetScaleType**
- **pfcUDFCreate.UDFCustomCreateInstructions.GetScaleType**
- **pfcUDFCreate.UDFCustomCreateInstructions.SetScale**
- **pfcUDFCreate.UDFCustomCreateInstructions.GetScale**

ScaleType specifies the length units of the UDF in the form of the `UDFScaleType` object. This enumerated type takes the following values:

- `UDFSCALE_SAME_SIZE`
- `UDFSCALE_SAME_DIMS`
- `UDFSCALE_CUSTOM`
- `UDFSCALE_nil`

Note

The default value is `UDFSCALE_SAME_SIZE` if this option is not set.

Scale specifies the scale factor. If the *ScaleType* is set to `UDFSCALE_CUSTOM`, `SetScale` assigns the user defined scale factor. Otherwise, this attribute is ignored.

Setting the Appearance of the Non UDF Dimensions

Methods Introduced:

- **pfcUDFCreate.UDFCustomCreateInstructions.SetDimDisplayType**
- **pfcUDFCreate.UDFCustomCreateInstructions.GetDimDisplayType**

The `pfcUDFCreate.UDFDimensionDisplayType` object sets the options in PTC Creo Parametric for determining the appearance in the model of UDF dimensions and pattern parameters that were not variable in the UDF, and therefore cannot be modified in the model. This enumerated type takes the following values:

- `UDFDISPLAY_NORMAL`
- `UDFDISPLAY_READ_ONLY`
- `UDFDISPLAY_BLANK`

Note

The default value is `UDFDISPLAY_NORMAL` if this option is not set.

Setting the Variable Dimensions and Parameters

Methods Introduced:

- **pfcUDFCreate.UDFCustomCreateInstructions.SetVariantValues**
- **pfcUDFCreate.UDFVariantValues.create**
- **pfcUDFCreate.UDFVariantValues.insert**
- **pfcUDFCreate.pfcUDFCreate.UDFVariantDimension_Create**
- **pfcUDFCreate.pfcUDFCreate.UDFVariantPatternParam_Create**

`pfcUDFVariantValues` class represents an array of variable dimensions and pattern parameters.

Use `pfcUDFCreate.UDFVariantValues.create` to create an empty object and then use `pfcUDFCreate.UDFVariantValues.insert` to add `pfcUDFCreate.UDFVariantPatternParam` or `pfcUDFCreate.UDFVariantDimension` objects one by one.

`pfcUDFCreate.pfcUDFCreate.UDFVariantDimension_Create` is a static method creating a `pfcUDFCreate.UDFVariantDimension`. It accepts the following parameters:

- *Name*—The symbol that the dimension had when the UDF was originally defined not the prompt that the UDF uses when it is created interactively. To make this name easy to remember, before you define the UDF that you plan to create with the J-Link, you should modify the symbols of all the dimensions that you want to select to be variable. If you get the name wrong, `pfcSolid.Solid.CreateUDFGroup` will not recognize the dimension and prompts the user for the value in the usual way does not modify the value.
- *DimensionValue*—The new value.

If you do not remember the name, you can find it by creating the UDF interactively in a test model, then using the `pfcFeature.FeatureGroup.ListUDFDimensions` and `pfcUDFGroup.UDFDimension.GetUDFDimensionName` to find out the name.

`pfcUDFCreate.pfcUDFCreate.UDFVariantPatternParam_Create` is a static method which creates a `pfcUDFCreate.UDFVariantPatternParam`. It accepts the following parameters:

- *name*—The string name that the pattern parameter had when the UDF was originally defined
- —The new value.

After the `pfcUDFCreate.UDFVariantValues` object has been compiled, use `pfcUDFCreate.UDFCustomCreateInstructions.SetVariantValues` to add the variable dimensions and parameters to the instructions.

Setting the User Defined References

Methods Introduced:

- **pfcUDFCreate.UDFReferences.create**
- **pfcUDFCreate.UDFReferences.insert**
- **pfcUDFCreate.pfcUDFCreate.UDFReference_Create**
- **pfcUDFCreate.UDFReference.SetIsExternal**
- **pfcUDFCreate.UDFReference.SetReferenceItem**
- **pfcUDFCreate.UDFCustomCreateInstructions.SetReferences**

UDFReferences class represents an array of element references. Use `pfcUDFCreate.UDFReferences.create` to create an empty object and then use `pfcUDFCreate.UDFReferences.insert` to add `UDFReference` objects one by one.

The method `pfcUDFCreate.pfcUDFCreate.UDFReference_Create` is a static method creating a `UDFReference` object. It accepts the following parameters:

- *PromptForReference*—The prompt defined for this reference when the UDF was originally set up. It indicates which reference this structure is providing. If you get the prompt wrong, `pfcSolid.Solid.CreateUDFGroup` will not recognize it and prompts the user for the reference in the usual way.
- *ReferenceItem*—Specifies the `pfcSelect.Selection` object representing the referenced element. You can set `Selection` programmatically or prompt the user for a selection separately. You cannot set an embedded datum as the UDF refereence.

  There are two types of reference:

  - Internal—The referenced element belongs directly to the model that will contain the UDF. For an assembly, this means that the element belongs to the top level.
  - External—The referenced element belongs to an assembly member other than the placement member.

To set the reference type, use the method `pfcUDFCreate.UDFReference.SetIsExternal`.

To set the item to be used for reference, use the method `pfcUDFCreate.UDFReference.SetReferenceItem`.

After the UDFReferences object has been set, use pfcUDFCreate.UDFCustomCreateInstructions.SetReferences to add the program-defined references.

Setting the Assembly Intersections

Methods Introduced:

- **pfcUDFCreate.UDFAssemblyIntersections.create()**
- **pfcUDFCreate.UDFAssemblyIntersections.insert()**
- **pfcUDFCreate.pfcUDFCreate.UDFAssemblyIntersection_Create**
- **pfcUDFCreate.UDFAssemblyIntersection.SetInstanceNames**
- **pfcUDFCreate.UDFCustomCreateInstructions.SetIntersections**

The pfcUDFCreate.UDFAssemblyIntersections class represents an array of element references.

Use pfcUDFCreate.pfcUDFCreate.UDFAssemblyIntersections.create to create an empty object and then use pfcUDFCreate.UDFAssemblyIntersections.insert to add pfcUDFCreate.UDFAssemblyIntersection objects one by one.

pfcUDFCreate.pfcUDFCreate.UDFAssemblyIntersection_Create is a static method creating a pfcUDFCreate.UDFReference object. It accepts the following parameters:

- *ComponentPath*—Is an com.ptc.cipjava.intseq type object representing the component path of the part to be intersected.
- *Visibility level*—The number that corresponds to the visibility level of the intersected part in the assembly. If the number is equal to the length of the component path the feature is visible in the part that it intersects. If *Visibility level* is 0, the feature is visible at the level of the assembly containing the UDF.

pfcUDFCreate.UDFAssemblyIntersection.SetInstanceNames sets an array of names for the new instances of parts created to represent the intersection geometry. This method accepts the following parameters:

- *instance names*—is a com.ptc.cipjava.stringseq type object representing the array of new instance names.

After the pfcUDFCreate.UDFAssemblyIntersections object has been set, use pfcUDFCreate.UDFCustomCreateInstructions.SetIntersections to add the assembly intersections.

Setting Orientations

Methods Introduced:

- **pfcUDFCreate.UDFCustomCreateInstructions.SetOrientations**
- **pfcUDFCreate.UDFOrientations.create**
- **pfcUDFCreate.UDFOrientations.insert**

`pfcUDFCreate.UDFOrientations` class represents an array of orientations that provide the answers to PTC Creo Parametric prompts that use a flip arrow. Each term is a `pfcUDFCreate.UDFOrientation` object that takes the following values:

- `UDFORIENT_INTERACTIVE`—Prompt for the orientation using a flip arrow.
- `UDFORIENT_NO_FLIP`—Accept the default flip orientation.
- `UDFORIENT_FLIP`—Invert the orientation from the default orientation.

Use `pfcUDFCreate.UDFOrientations.create` to create an empty object and then use `pfcUDFCreate.UDFOrientations.insert` to add `pfcUDFCreate.UDFOrientation` objects one by one.

The order of orientations should correspond to the order in which PTC Creo Parametric prompts for them when the UDF is created interactively. If you do not provide an orientation that PTC Creo Parametric needs, it uses the default value `NO_FLIP`.

After the `pfcUDFCreate.UDFOrientations` object has been set use `pfcUDFCreate.UDFCustomCreateInstructions.SetOrientations` to add the orientations.

Setting Quadrants

Methods Introduced:

- **pfcUDFCreate.UDFCustomCreateInstructions.SetQuadrants**

The method `pfcUDFCreate.UDFCustomCreateInstructions.SetQuadrants` sets an array of points, which provide the X, Y, and Z coordinates that correspond to the picks answering the PTC Creo Parametric prompts for the feature positions. The order of quadrants should correspond to the order in which PTC Creo Parametric prompts for them when the UDF is created interactively.

Setting the External References

Methods Introduced:

- **pfcUDFCreate.UDFCustomCreateInstructions.SetExtReferences**

The method pfcUDFCreate.UDFCustomCreateInstructions.SetExtReferences sets an external reference assembly to be used when placing the UDF. This will be required when placing the UDF in the component using references outside of that component. References could be to the top level assembly of another component.

Example Code 1

The sample code in the file pfcUDFCreateExamples.java located at <creo_jlink_loadpoint>/jlink_appls/jlinkexamples copies of a node UDF at a particular coordinate system location in a part. The node UDF is a spherical cut centered at the coordinate system whose diameter is driven by the 'diam' argument to the method. The method returns the FeatureGroup object created, or null if an error occurred..

16

Datum Features

This chapter describes the J-Link methods that provide read access to the properties of datum features.

Datum Plane Features

The properties of the Datum Plane feature are defined in the `pfcDatumPlaneFeat.DatumPlaneFeat`data object.

Methods Introduced:

- **pfcDatumPlaneFeat.DatumPlaneFeat.GetFlip**
- **pfcDatumPlaneFeat.DatumPlaneFeat.GetConstraints**
- **pfcDatumPlaneFeat.DatumPlaneConstraint.GetConstraintType**
- **pfcDatumPlaneFeat.DatumPlaneThroughConstraint.GetThroughRef**
- **pfcDatumPlaneFeat.DatumPlaneNormalConstraint.GetNormalRef**
- **pfcDatumPlaneFeat.DatumPlaneParallelConstraint.GetParallelRef**
- **pfcDatumPlaneFeat.DatumPlaneTangentConstraint.GetTangentRef**
- **pfcDatumPlaneFeat.DatumPlaneOffsetConstraint.GetOffsetRef**
- **pfcDatumPlaneFeat.DatumPlaneOffsetConstraint.GetOffsetValue**
- **pfcDatumPlaneFeat.DatumPlaneOffsetCoordSysConstraint.GetCsysAxis**
- **pfcDatumPlaneFeat.DatumPlaneAngleConstraint.GetAngleRef**
- **pfcDatumPlaneFeat.DatumPlaneAngleConstraint.GetAngleValue**
- **pfcDatumPlaneFeat.DatumPlaneSectionConstraint.GetSectionRef**
- **pfcDatumPlaneFeat.DatumPlaneSectionConstraint.GetSectionIndex**

The properties of the `pfcDatumPlaneFeat.DatumPlaneFeat` object are described as follows:

- `Flip`—Specifies whether the datum plane was flipped during creation. Use the method `pfcDatumPlaneFeat.DatumPlaneFeat.GetFlip` to determine if the datum plane was flipped during creation.
- `Constraints`—Specifies a collection of constraints given by the `pfcDatumPlaneFeat.DatumPlaneConstraint` object. The method `pfcDatumPlaneFeat.DatumPlaneFeat.GetConstraints` obtains the collection of constraints defined for the datum plane.

Use the method `pfcDatumPlaneFeat.DatumPlaneConstraint.GetConstraintType` to obtain the type of constraint. The type of constraint is given by the `pfcDatumPlaneFeat.DatumPlaneConstraintType` enumerated type. The available types are as follows:

- `DTMPLN_THRU`—Specifies the Through constraint. The `pfcDatumPlaneFeat DatumPlaneThroughConstraint` object specifies this constraint. Use the method `pfcDatumPlaneFeat.DatumPlaneThroughConstraint.`

GetThroughRef to get the reference selection handle for the Through constraint.

- DTMPLN_NORM—Specifies the Normal constraint. The pfcDatumPlaneFeat DatumPlaneNormalConstraint object specifies this constraint. Use the method pfcDatumPlaneFeat.DatumPlaneNormalConstraint .GetNormalRef to get the reference selection handle for the Normal constraint.
- DTMPLN_PRL—Specifies the Parallel constraint. The DatumPlaneFeatDatumPlaneParallelConstraint object specifies this constraint. Use the method pfcDatumPlaneFeat.DatumPlaneParallelConstraint .GetParallelRef to get the reference selection handle for the Parallel constraint.
- DTMPLN_TANG—Specifies the Tangent constraint. The pfcDatumPlaneFeat .DatumPlaneTangentConstraint object specifies this constraint. Use the method pfcDatumPlaneFeat.DatumPlaneTangentConstraint .GetTangentRef to get the reference selection handle for the Tangent constraint.
- DTMPLN_OFFS—Specifies the Offset constraint. The pfcDatumPlaneFeat .DatumPlaneOffsetConstraint object specifies this constraint. Use the method pfcDatumPlaneFeat.DatumPlaneOffsetConstraint .GetOffsetRef to get the reference selection handle for the Offset constraint. Use the method pfcDatumPlaneFeat.DatumPlaneOffsetConstraint.GetOffsetValue to get the offset value.

  An Offset constraint where the offset reference is a coordinate system is given by the pfcDatumPlaneFeat.DatumPlaneOffsetCoordSysConstraint object. Use the method pfcDatumPlaneFeat.DatumPlaneOffsetCoordSysConstraint .GetCsysAxis to get the reference coordinate axis.
- DTMPLN_ANG—Specifies the Angle constraint. The pfcDatumPlaneFeat .DatumPlaneAngleConstraint object specifies this constraint. Use the method pfcDatumPlaneFeat.DatumPlaneAngleConstraint

`.GetAngleRef` to get the reference selection handle for the Angle constraint. Use the method `pfcDatumPlaneFeat.DatumPlaneAngleConstraint .GetAngleValue` to get the angle value.

- `DTMPLN_SEC`—Specifies the Section constraint. The `pfcDatumPlaneFeat DatumPlaneSectionConstraint` object specifies this constraint. Use the method `pfcDatumPlaneFeat.DatumPlaneSectionConstraint. GetSectionRef` to get the reference selection for the Section constraint. Use the method `pfcDatumPlaneFeat.DatumPlaneSectionConstraint .GetSectionIndex` to get the section index.

Datum Axis Features

The properties of the Datum Axis feature are defined in the `pfcDatumAxisFeat.DatumAxisFeat` data object.

Methods Introduced:

- **pfcDatumAxisFeat.DatumAxisFeat.GetConstraints**
- **pfcDatumAxisFeat.DatumAxisConstraint.GetConstraintType**
- **pfcDatumAxisFeat.DatumAxisConstraint.GetConstraintRef**
- **pfcDatumAxisFeat.DatumAxisFeat.GetDimConstraints**
- **pfcDatumAxisFeat.DatumAxisDimensionConstraint.GetDimOffset**
- **pfcDatumAxisFeat.DatumAxisDimensionConstraint.GetDimRef**

The properties of the `pfcDatumAxisFeat.DatumAxisFeat` object are described as follows:

- `Constraints`—Specifies a collection of constraints given by the `pfcDatumAxisFeat.DatumAxisConstraint` object. The method `pfcDatumAxisFeat.DatumAxisFeat.GetConstraints` obtains the collection of constraints applied to the Datum Axis feature.

  This object contains the following attributes:

  - `ConstraintType`—Specifies the type of constraint in terms of the `pfcDatumAxisFeat.DatumAxisConstraintType` enumerated type. The constraint type determines the type of datum axis. The constraint types are:
    - `DTMAXIS_NORMAL`—Specifies the Normal datum constraint.

- DTMAXIS_THRU—Specifies the Through datum constraint.
- DTMAXIS_TANGENT—Specifies the Tangent datum constraint.
- DTMAXIS_CENTER—Specifies the Center datum constraint.

Use the method pfcDatumAxisFeat.DatumAxisConstraint.GetConstraintType to get the constraint type.

- ConstraintRef—Specifies the reference selection for the constraint. Use the method pfcDatumAxisFeat.DatumAxisConstraint.GetConstraintRef to get the reference selection handle.

- DimConstraints—Specifies a collection of dimension constraints given by the pfcDatumAxisFeat.DatumAxisDimensionConstraint object. The method pfcDatumAxisFeat.DatumAxisFeat.GetDimConstraints obtains the collection of dimension constraints applied to the Datum Axis feature.

  This pfcDatumAxisFeat.DatumAxisDimensionConstraint object contains the following attributes:

  - DimOffset—Specifies the offset value for the dimension constraint. Use the method pfcDatumAxisFeat.DatumAxisDimensionConstraint.GetDimOffset to get the offset value.
  - DimRef—Specifies the reference selection for the dimension constraint. Use the method pfcDatumAxisFeat.DatumAxisDimensionConstraint.GetDimRef to get the reference selection handle.

General Datum Point Features

The properties of the General Datum Point feature are defined in the pfcDatumPointFeat.DatumPointFeat. data object.

Methods Introduced:

- **pfcDatumPointFeat.DatumPointFeat.GetFeatName**
- **pfcDatumPointFeat.DatumPointFeat.GetPoints**
- **pfcDatumPointFeat.GeneralDatumPoint.GetName**
- **pfcDatumPointFeat.GeneralDatumPoint.GetPlaceConstraints**
- **pfcDatumPointFeat.GeneralDatumPoint.GetDimConstraints**

- **pfcDatumPointFeat.DatumPointConstraint.GetConstraintRef**
- **pfcDatumPointFeat.DatumPointConstraint.GetConstraintType**
- **pfcDatumPointFeat.DatumPointConstraint.GetValue**

The properties of the `pfcDatumPointFeat.DatumPointFeat` object are described as follows:

- `FeatName`—Specifies the name of the General Datum Point feature. Use the method `pfcDatumPointFeat.DatumPointFeat.GetFeatName` to get the name.
- `GeneralDatumPoints`—Specifies a collection of general datum points given by the `pfcDatumPointFeat.GeneralDatumPoint` object. Use the method `pfcDatumPointFeat.DatumPointFeat.GetPoints` to obtain the collection of general datum points. The `pfcDatumPointFeat.GeneralDatumPoint` object consists of the following attributes:
  - `Name`—Specifies the name of the general datum point. Use the method `pfcDatumPointFeat.GeneralDatumPoint.GetName` to get the name.
  - `PlaceConstraints`—Specifies a collection of placement constraints given by the `pfcDatumPointFeat.DatumPointPlacementConstraint` object. Use the method `pfcDatumPointFeat.GeneralDatumPoint.GetPlaceConstraints` to obtain the collection of placement constraints.
  - `DimConstraints`—Specifies a collection of dimension constraints given by the `pfcDatumPointFeat.DatumPointDimensionConstraint` object. Use the method `pfcDatumPointFeat.GeneralDatumPoint.GetDimConstraints` to obtain the collection of dimension constraints.

The constraints for a datum point are given by the `pfcDatumPointFeat.DatumPointConstraint` object. This object contains the following attributes:

- `ConstraintRef`—Specifies the reference selection for the datum point constraint. Use the method `pfcDatumPointFeat.DatumPointConstraint.GetConstraintRef` to get the reference selection handle.
- `ConstraintType`—Specifies the type of datum point constraint. in terms of the `pfcDatumPointFeat.DatumPointConstraintType`

enumerated type. Use the method `pfcDatumPointFeat.DatumPointConstraint.GetConstraintType` to get the constraint type.

- `Value`—Specifies the constraint reference value with respect to the datum point. Use the method `pfcDatumPointFeat.DatumPointConstraint.GetValue` to get the value of the constraint reference with respect to the datum point.

The `pfcDatumPointFeat.DatumPointPlacementConstraint` and `pfcDatumPointFeat.DatumPointDimensionConstraint` objects inherit from the `pfcDatumPointFeat.DatumPointConstraint` object. Use the methods of the `pfcDatumPointFeat.DatumPointConstraint` object for the inherited objects.

Datum Coordinate System Features

The properties of the Datum Coordinate System feature are defined in the `pfcCoordSysFeat.CoordSysFeat` object.

Methods Introduced:

- **pfcCoordSysFeat.CoordSysFeat.GetOriginConstraints**
- **pfcCoordSysFeat.DatumCsysOriginConstraint.GetOriginRef**
- **pfcCoordSysFeat.CoordSysFeat.GetDimensionConstraints**
- **pfcCoordSysFeat.DatumCsysDimensionConstraint.GetDimRef**
- **pfcCoordSysFeat.DatumCsysDimensionConstraint.GetDimValue**
- **pfcCoordSysFeat.DatumCsysDimensionConstraint.GetDimConstraintType**
- **pfcCoordSysFeat.CoordSysFeat.GetOrientationConstraints**
- **pfcCoordSysFeat.DatumCsysOrientMoveConstraint.GetOrientMoveConstraintType**
- **pfcCoordSysFeat.DatumCsysOrientMoveConstraint.GetOrientMoveValue**
- **pfcCoordSysFeat.CoordSysFeat.GetIsNormalToScreen**
- **pfcCoordSysFeat.CoordSysFeat.GetOffsetType**
- **pfcCoordSysFeat.CoordSysFeat.GetOnSurfaceType**
- **pfcCoordSysFeat.CoordSysFeat.GetOrientByMethod**

The properties of the `pfcCoordSysFeat.CoordSysFeat` object are described as follows:

- OriginConstraints—Specifies a collection of origin constraints given by the pfcCoordSysFeat.DatumCsysOriginConstraint object. Use the method pfcCoordSysFeat.CoordSysFeat.GetOriginConstraints to obtain the collection of origin constraints for the coordinate system. This object contains the following attribute:
  - OriginRef—Specifies the selection reference for the origin. Use the method pfcCoordSysFeat.DatumCsysOriginConstraint.GetOriginRef to get the selection reference handle.
- DimensionConstraints—Specifies a collection of dimension constraints given by the pfcCoordSysFeat.DatumCsysDimensionConstraint object. Use the method pfcCoordSysFeat.CoordSysFeat.GetDimensionConstraints to obtain the collection of dimension constraints for the coordinate system. This object contains the following attributes:
  - DimRef—Specifies the reference selection for the dimension constraint. Use the method pfcCoordSysFeat.DatumCsysDimensionConstraint.GetDimRef to get the reference selection handle.
  - DimValue—Specifies the value of the reference. Use the method pfcCoordSysFeat.DatumCsysDimensionConstraint.GetDimValue to get the value.
  - DimConstraintType—Specifies the type of dimension constraint in terms of the pfcCoordSysFeat.DatumCsysDimConstraintType enumerated type. Use the method pfcCoordSysFeat.DatumCsysDimensionConstraint.GetDimConstraintType to get the constraint type. The constraint types are:
    - DTMCSYS_DIM_OFFSET—Specifies the offset type constraint.
    - DTMCSYS_DIM_ALIGN—Specifies the align type constraint.
- OrientationConstraints—Specifies a collection of orientation constraints given by the CoordSysFeat.DatumCsysOrientMoveConstraint object. Use the method pfcCoordSysFeat.CoordSysFeat.GetOrientationCon

straints to obtain the collection of orientation constraints for the coordinate system. This object contains the following attributes:

  - OrientMoveConstraintType—Specifies the type of orientation for the constraint. The orientation type is given by the pfcCoordSysFeat.DatumCsysOrientMoveConstraintType enumerated type. Use the method pfcCoordSysFeat.DatumCsysOrientMoveConstraint.GetOrientMoveConstraintType to get the orientation type.
  - OrientMoveValue—Specifies the reference value for the constraint. Use the method pfcCoordSysFeat.DatumCsysOrientMoveConstraint.GetOrientMoveValue to get the reference value.
- IsNormalToScreen—Specifies if the coordinate system is normal to screen. Use the method pfcCoordSysFeat.CoordSysFeat.GetIsNormalToScreen to determine if the coordinate system is normal to screen.
- OffsetType—Specifies the offset type of the coordinate system in terms of the pfcCoordSysFeat.DatumCsysOffsetType enumerated type. Use the method pfcCoordSysFeat.CoordSysFeat.GetOffsetType to get the offset type. The offset types are:
  - DTMCSYS_OFFSET_CARTESIAN—Specifies a cartesian coordinate system that has been defined by setting the values for the DTMCSYS_MOVE_TRAN_X, DTMCSYS_MOVE_TRAN_Y, and DTMCSYS_MOVE_TRAN_Z or DTMCSYS_MOVE_ROT_X, DTMCSYS_MOVE_ROT_Y, and DTMCSYS_MOVE_ROT_Z orientation constants.
  - DTMCSYS_OFFSET_CYLINDRICAL—Specifies a cylindrical coordinate system that has been defined by setting the values for the DTMCSYS_MOVE_RAD, DTMCSYS_MOVE_THETA, and DTMCSYS_MOVE_TRAN_ZI orientation constants.
  - DTMCSYS_OFFSET_SPHERICAL—Specifies a spherical coordinate system that has been defined by setting the values for the DTMCSYS_MOVE_RAD, DTMCSYS_MOVE_THETA, and DTMCSYS_MOVE_TRAN_PHI orientation constants.
- OnSurfaceType—Specifies the on surface type for the coordinate system in terms of the pfcCoordSysFeat.DatumCsysOffsetType enumerated type. Use the method

pfcCoordSysFeat.CoordSysFeat.GetOnSurfaceType to get the on surface type property of the coordinate system. The on surface types are:

- DTMCSYS_ONSURF_LINEAR—Specifies a coordinate system placed on the selected surface by using two linear dimensions.
- DTMCSYS_ONSURF_RADIAL—Specifies a coordinate system placed on the selected surface by using a linear dimension and an angular dimension. The radius value is used to specify the linear dimension.
- DTMCSYS_ONSURF_DIAMETER—This type is similar to the DTMCSYS_ONSURF_RADIAL type, except that the diameter value is used to specify the linear dimension. It is available only when planar surfaces are used as the reference.

- OrientByMethod—Specifies the orientation method in terms of the pfcCoordSysFeat.DatumCsysOrientByMethod enumerated type. Use the method pfcCoordSysFeat.CoordSysFeat.GetOrientByMethod to get the orientation method. The available orientation types are:
  - DTMCSYS_ORIENT_BY_SEL_REFS—Specifies the orientation by selected references.
  - DTMCSYS_ORIENT_BY_SEL_CSYS_AXES—Specifies the orientation by corordinate system axes.

Example: Reading Properties of Datum Features

The sample code in the file pfcReadBasicFeatPropertiesExamples.java located at <creo_jlink_loadpoint>/jlink_appls/jlinkexamples demonstrates how to read the basic properties of datum features.

17

Geometry Evaluation

This chapter describes geometry representation and discusses how to evaluate geometry using J-Link.

Geometry Traversal

- A simple rectangular face has one contour and four edges.
- A contour will traverse a boundary so that the part face is always on the right-hand side (RHS). For an external contour the direction of traversal is clockwise. For an internal contour the direction of traversal is counterclockwise.
- If a part is extruded from a sketch that has a U-shaped cross section there will be separate surfaces at each leg of the U-channel.
- If a part is extruded from a sketch that has a square-shaped cross section, and a slot feature is then cut into the part to make it look like a U-channel, there will be one surface across the legs of the U-channel. The original surface of the part is represented as one surface with a cut through it.

Geometry Terms

Following are definitions for some geometric terms:

- Surface—An ideal geometric representation, that is, an infinite plane.
- Face—A trimmed surface. A face has one or more contours.
- Contour—A closed loop on a face. A contour consists of multiple edges. A contour can belong to one face only.
- Edge—The boundary of a trimmed surface.

An edge of a solid is the intersection of two surfaces. The edge belongs to those two surfaces and to two contours. An edge of a datum surface can be either the intersection of two datum surfaces or the external boundary of the surface.

If the edge is the intersection of two datum surfaces it will belong to those two surfaces and to two contours. If the edge is the external boundary of the datum surface it will belong to that surface alone and to a single contour.

Traversing the Geometry of a Solid Block

Methods Introduced:

- **pfcModelItem.ModelItemOwner.ListItems**
- **pfcGeometry.Surface.ListContours**
- **pfcGeometry.Contour.ListElements**

To traverse the geometry, follow these steps:

1. Starting at the top-level model, use `pfcModelItem.ModelItemOwner.ListItems` with an argument of `ModelItemType.ITEM_SURFACE`.
2. Use `pfcGeometry.Surface.ListContours` to list the contours contained in a specified surface.
3. Use `pfcGeometry.Contour.ListElements` to list the edges contained in the contour.

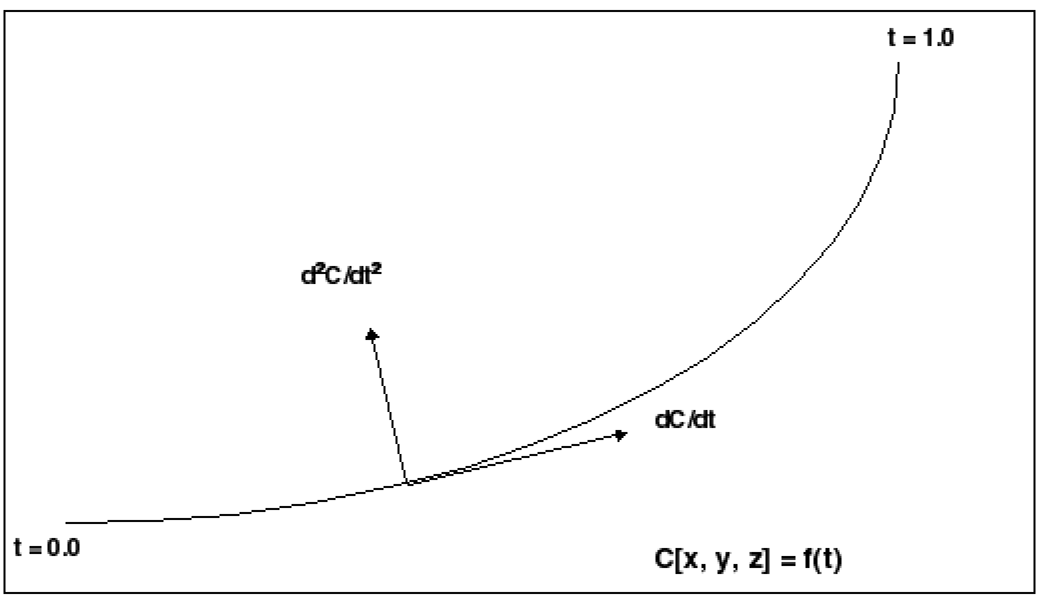
Curves and Edges

Datum curves, surface edges, and solid edges are represented in the same way in J-Link. You can get edges through geometry traversal or get a list of edges using the methods presented in the chapter ModelItem on page 207.

The t Parameter

The geometry of each edge or curve is represented as a set of three parametric equations that represent the values of x, y, and z as functions of an independent parameter, `t`. The `t` parameter varies from 0.0 at the start of the curve to 1.0 at the end of it.

The following figure illustrates curve and edge parameterization.



Curve and Edge Types

Solid edges and datum curves can be any of the following types:

- LINE—A straight line represented by the class `pfcGeometry.Line`.
- ARC—A circular curve represented by the class `pfcGeometry.Arc`.
- SPLINE—A nonuniform cubic spline, represented by the class `pfcGeometry.Spline`.
- B-SPLINE—A nonuniform rational B-spline curve or edge, represented by the class `pfcGeometry.BSpline`.
- COMPOSITE CURVE—A combination of two or more curves, represented by the class `pfcGeometry.CompositeCurve`. This is used for datum curves only.

See the appendix Geometry Representations on page 429 for the parameterization of each curve type. To determine what type of curve a `pfcGeometry.Edge` or `pfcGeometry.Curve` object represents, use the Java `instanceof` operator.

Because each curve class inherits from `pfcGeometry.GeomCurve`, you can use all the evaluation methods in `GeomCurve` on any edge or curve.

The following curve types are not used in solid geometry and are reserved for future expansion:

- CIRCLE (`pfcGeometry.Circle`)
- ELLIPSE (`pfcGeometry.Ellipse`)
- POLYGON (`pfcGeometry.Polygon`)
- ARROW (`pfcGeometry.Arrow`)
- TEXT (`pfcGeometry.Text`)

Evaluation of Curves and Edges

Methods Introduced:

- **pfcGeometry.GeomCurve.Eval3DData**
- **pfcGeometry.GeomCurve.EvalFromLength**
- **pfcGeometry.GeomCurve.EvalParameter**
- **pfcGeometry.GeomCurve.EvalLength**
- **pfcGeometry.GeomCurve.EvalLengthBetween**

The methods in `GeomCurve` provide information about any curve or edge.

The method `pfcGeometry.GeomCurve.Eval3DData` returns a `CurveXYZData` object with information on the point represented by the input parameter `t`. The method `pfcGeometry.GeomCurve.EvalFromLength` returns a similar object with information on the point that is a specified distance from the starting point.

The method pfcGeometry.GeomCurve.EvalParameter returns the t parameter that represents the input Point3D object.

Both pfcGeometry.GeomCurve.EvalLength and pfcGeometry.GeomCurve.EvalLengthBetween return numerical values for the length of the curve or edge.

Solid Edge Geometry

Methods Introduced:

- **pfcGeometry.Edge.GetSurface1**
- **pfcGeometry.Edge.GetSurface2**
- **pfcGeometry.Edge.GetEdge1**
- **pfcGeometry.Edge.GetEdge2**
- **pfcGeometry.Edge.EvalUV**
- **pfcGeometry.Edge.GetDirection**

> **Note**
>
> The methods in the interface Edge provide information only for solid or surface edges.

The methods pfcGeometry.Edge.GetSurface1 and pfcGeometry.Edge.GetSurface2 return the surfaces bounded by this edge. The methods pfcGeometry.Edge.GetEdge1 and pfcGeometry.Edge.GetEdge2 return the next edges in the two contours that contain this edge.

The method pfcGeometry.Edge.EvalUV evaluates geometry information based on the UV parameters of one of the bounding surfaces.

The method pfcGeometry.Edge.GetDirection returns a positive 1 if the edge is parameterized in the same direction as the containing contour, and –1 if the edge is parameterized opposite to the containing contour.

Curve Descriptors

A curve descriptor is a data object that describes the geometry of a curve or edge. A curve descriptor describes the geometry of a curve without being a part of a specific model.

Methods Introduced:

- **pfcGeometry.GeomCurve.GetCurveDescriptor**
- **pfcGeometry.GeomCurve.GetNURBSRepresentation**

> **Note**
>
> To get geometric information for an edge, access the `CurveDescriptor` object for one edge using `pfcGeometry.GeomCurve.GetCurveDescriptor`.

The method `pfcGeometry.GeomCurve.GetCurveDescriptor` returns a curve's geometry as a data object.

The method `pfcGeometry.GeomCurve.GetNURBSRepresentation` returns a Non-Uniform Rational B-Spline Representation of a curve.

Contours

Methods Introduced:

- **pfcGeometry.Surface.ListContours**
- **pfcGeometry.Contour.GetInternalTraversal**
- **pfcGeometry.Contour.FindContainingContour**
- **pfcGeometry.Contour.EvalArea**
- **pfcGeometry.Contour.EvalOutline**
- **pfcGeometry.Contour.VerifyUV**

Contours are a series of edges that completely bound a surface. A contour is not a `ModelItem`. You cannot get contours using the methods that get different types of `ModelItem`. Use the method `pfcGeometry.Surface.ListContours` to get contours from their containing surfaces.

The method `pfcGeometry.Contour.GetInternalTraversal` returns a `ContourTraversal` enumerated type that identifies whether a given contour is on the outside or inside of a containing surface.

Use the method `pfcGeometry.Contour.FindContainingContour` to find the contour that entirely encloses the specified contour.

The method `pfcGeometry.Contour.EvalArea` provides the area enclosed by the contour.

The method `pfcGeometry.Contour.EvalOutline` returns the points that make up the bounding rectangle of the contour.

Use the method pfcGeometry.Contour.VerifyUV to determine whether the given UVParams argument lies inside the contour, on the boundary, or outside the contour.

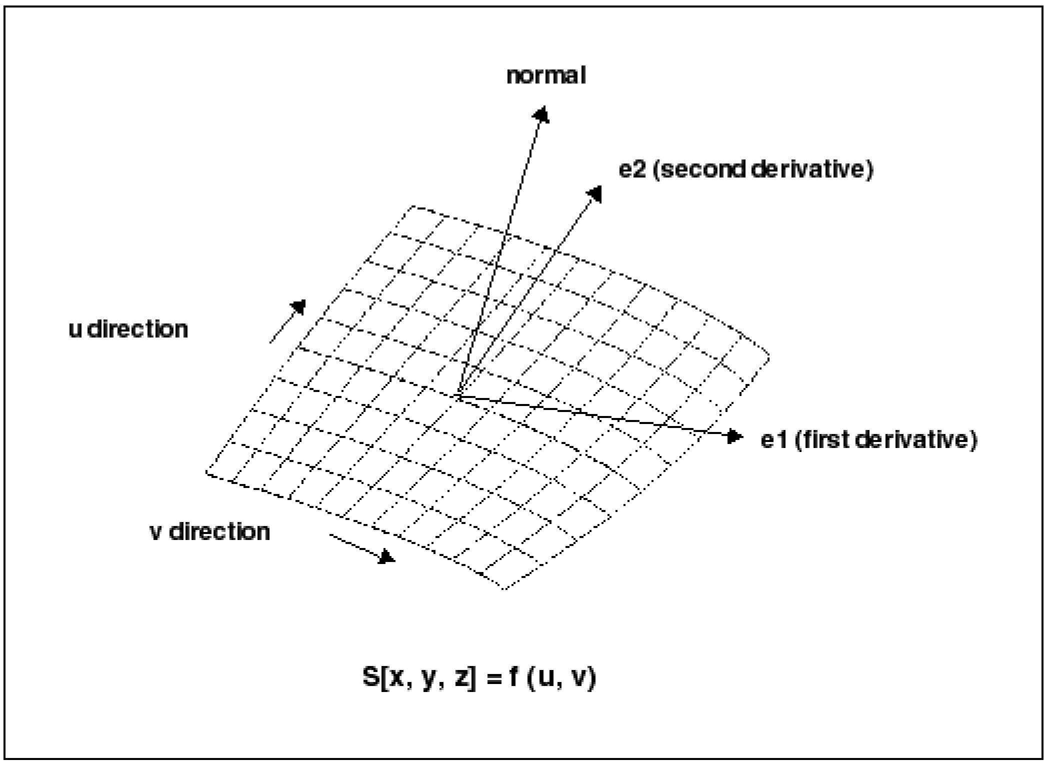
Surfaces

Using J-Link you access datum and solid surfaces in the same way.

UV Parameterization

A surface in PTC Creo Parametric is described as a series of parametric equations where two parameters, u and v, determine the x, y, and z coordinates. Unlike the edge parameter, t, these parameters need not start at 0.0, nor are they limited to 1.0.

The figure on the following page illustrates surface parameterization.



Surface Types

Surfaces within PTC Creo Parametric can be any of the following types:

- PLANE—A planar surface represented by the class `pfcGeometry.Plane`.
- CYLINDER—A cylindrical surface represented by the class `IGeometry.Cylinder`.
- CONE—A conic surface region represented by the class `pfcGeometry.Cone`.
- TORUS—A toroidal surface region represented by the class `pfcGeometry.Torus`.
- REVOLVED SURFACE—Generated by revolving a curve about an axis. This is represented by the class `pfcGeometry.RevSurface`.
- RULED SURFACE—Generated by interpolating linearly between two curve entities. This is represented by the class `pfcGeometry.RuledSurface`.
- TABULATED CYLINDER—Generated by extruding a curve linearly. This is represented by the class `pfcGeometry.TabulatedCylinder`.
- QUILT—A combination of two or more surfaces. This is represented by the class `pfcGeometry.Quilt`.

> **Note**
>
> This is used only for datum surfaces.

- COONS PATCH—A coons patch is used to blend surfaces together. It is represented by the class `pfcGeometry.CoonsPatch`.
- FILLET SURFACE—A filleted surface is found where a round or fillet is placed on a curved edge or an edge with a non-consistant arc radii. On a straight edge a cylinder is used to represent a fillet. This is represented by the class `pfcGeometry.FilletedSurface`.
- SPLINE SURFACE— A nonuniform bicubic spline surface that passes through a grid with tangent vectors given at each point. This is represented by the class `pfcGeometry.SplineSurface`.
- NURBS SURFACE—A NURBS surface is defined by basic functions (in *u* and *v*), expandable arrays of knots, weights, and control points. This is represented by the class `pfcGeometry.NURBSSurface`.
- CYLINDRICAL SPLINE SURFACE— A cylindrical spline surface is a nonuniform bicubic spline surface that passes through a grid with tangent vectors given at each point. This is represented by the class `pfcGeometry.CylindricalSplineSurface`.

To determine which type of surface a pfcGeometry.Surface object represents, access the surface type using pfcGeometry.Geometry.GetSurfaceType.

Surface Information

Methods Introduced:

- **pfcGeometry.Surface.GetSurfaceType**
- **pfcGeometry.Surface.GetXYZExtents**
- **pfcGeometry.Surface.GetUVExtents**
- **pfcGeometry.Surface.GetOrientation**

Evaluation of Surfaces

Surface methods allow you to use multiple surface information to calculate, evaluate, determine, and examine surface functions and problems.

Methods Introduced:

- **pfcGeometry.Surface.GetOwnerQuilt**
- **pfcGeometry.Surface.EvalClosestPoint**
- **pfcGeometry.Surface.EvalClosestPointOnSurface**
- **pfcGeometry.Surface.Eval3DData**
- **pfcGeometry.Surface.EvalParameters**
- **pfcGeometry.Surface.EvalArea**
- **pfcGeometry.Surface.EvalDiameter**
- **pfcGeometry.Surface.EvalPrincipalCurv**
- **pfcGeometry.Surface.VerifyUV**
- **pfcGeometry.Surface.EvalMaximum**
- **pfcGeometry.Surface.EvalMinimum**
- **pfcGeometry.Surface.ListSameSurfaces**

The method pfcGeometry.Surface.GetOwnerQuilt returns the Quilt object that contains the datum surface.

The method pfcGeometry.Surface.EvalClosestPoint projects a three-dimensional point onto the surface. Use the method pfcGeometry.Surface.EvalClosestPointOnSurface to determine whether the specified three-dimensional point is on the surface, within the accuracy of the part. If it is, the method returns the point that is exactly on the surface. Otherwise the method returns null.

The method pfcGeometry.Surface.Eval3DData returns a SurfXYZData object that contains information about the surface at the specified *u* and *v* parameters. The method pfcGeometry.Surface.EvalParameters returns the *u* and *v* parameters that correspond to the specified three-dimensional point.

The method pfcGeometry.Surface.EvalArea returns the area of the surface, whereas pfcGeometry.Surface.EvalDiameter returns the diameter of the surface. If the diameter varies the optional UVParams argument identifies where the diameter should be evaluated.

The method pfcGeometry.Surface.EvalPrincipalCurv returns a CurvatureData object with information regarding the curvature of the surface at the specified *u* and *v* parameters.

Use the method pfcGeometry.Surface.VerifyUV to determine whether the UVParams are actually within the boundary of the surface.

The methods pfcGeometry.Surface.EvalMaximum and pfcGeometry.Surface.EvalMinimum return the three-dimensional point on the surface that is the furthest in the direction of (or away from) the specified vector.

The method pfcGeometry.Surface.ListSameSurfaces identifies other surfaces that are tangent and connect to the given surface.

Surface Descriptors

A surface descriptor is a data object that describes the shape and geometry of a specified surface. A surface descriptor allows you to describe a surface in 3D without an owner ID.

Methods Introduced:

- **pfcGeometry.Surface.GetSurfaceDescriptor**
- **pfcGeometry.Surface.GetNURBSRepresentation**

The method pfcGeometry.Surface.GetSurfaceDescriptor returns a surfaces geometry as a data object.

The method pfcGeometry.Surface.GetNURBSRepresentation returns a Non-Uniform Rational B-Spline Representation of a surface.

Axes, Coordinate Systems, and Points

Coordinate axes, datum points, and coordinate systems are all model items. Use the methods that return ModelItems to get one of these geometry objects. Refer to the chapter ModelItem on page 207 for additional information.

Evaluation of ModelItems

MethodsIntroduced:

- **pfcGeometry.Axis.GetSurf**
- **pfcGeometry.CoordSystem.GetCoordSys**
- **pfcGeometry.Point.GetPoint**

The method `pfcGeometry.Axis.GetSurf` returns the revolved surface that uses the axis.

The method `pfcGeometry.CoordSystem.GetCoordSys` returns the `Transform3D` object (which includes the origin and x-, y-, and z- axes) that defines the coordinate system.

The method `pfcGeometry.Point.GetPoint` returns the xyz coordinates of the datum point.

Interference

PTC Creo Parametric assemblies can contain interferences between components when constraint by certain rules defined by the user. The `com.ptc.pfc.pfcInterference` packageallows the user to detect and analyze any interferences within the assembly. The analysis of this functionality should be looked at from two standpoints: global and selection based analysis.

Methods Introduced:

- **pfcInterference.pfcInterference.CreateGlobalEvaluator**
- **pfcInterference.GlobalEvaluator.ComputeGlobalInterference**
- **pfcInterference.GlobalEvaluator.GetAssem**
- **pfcInterference.GlobalEvaluator.SetAssem**
- **pfcInterference.GlobalInterference.GetVolume**
- **pfcInterference.GlobalInterference.GetSelParts**

To compute all the interferences within an Assembly one has to call `pfcInterference.pfcInterference.CreateGlobalEvaluator` with a `Assembly.Assembly` object as an argument. This call returns a`pfcGlobalEvaluator` object. The `GlobalEvaluator` can be used to extract an assembly object or to set an assembly object for the interference computation.

The methods `pfcInterference.GlobalEvaluator.GetAssem` and `pfcInterference.GlobalEvaluator.SetAssem` with `pfcAssembly.Assembly` as an argument allow you to do exactly that.

The method pfcInterference.GlobalEvaluator.ComputeGlobalInterference determines the set of all the interferences within the assembly.

This method will return a sequence of pfcInterference.GlobalInterference objects or null if there are no interfering parts. Each object contains a pair of intersecting parts and an object representing the interference volume, which can be extracted by using pfcInterference.GlobalInterference.GetSelParts and pfcInterference.GlobalInterference.GetVolume respectively.

Analyzing Interference Information

Methods Introduced:

- **pfcSelect.pfcSelect.SelectionPair_Create**
- **pfcInterference.pfcInterference.CreateSelectionEvaluator**
- **pfcInterference.SelectionEvaluator.GetSelections**
- **pfcInterference.SelectionEvaluator.SetSelections**
- **pfcInterference.SelectionEvaluator.ComputeInterference**
- **pfcInterference.SelectionEvaluator.ComputeClearance**
- **pfcInterference.SelectionEvaluator.ComputeNearestCriticalDistance**

The method pfcSelect.pfcSelect.SelectionPair_Create creates a pfcSelect.SelectionPair object using two pfcSelect.Selection objects as arguments.

A return from this method will serve as an argument to pfcInterference.pfcInterference.CreateSelectionEvaluator, which will provide a way to determine the interference data between the two selections.

pfcInterference.SelectionEvaluator.GetSelections and pfcInterference.SelectionEvaluator.SetSelections will extract and set the object to be evaluated respectively.

pfcInterference.SelectionEvaluator.ComputeInterference determines the interfering information about the provided selections. This method will return the pfcInterference.InterferenceVolume object or null if the selections do no interfere.

pfcInterference.SelectionEvaluator.ComputeClearance computes the clearance data for the two selection. This method returns a pfcInterference.ClearanceData object, which can be used to obtain and set clearance distance, nearest points between selections, and a boolean IsInterferening variable.

pfcInterference.SelectionEvaluator.

ComputeNearestCriticalDistance finds a critical point of the distance function between two selections.

This method returns a pfcInterference.CriticalDistanceData object, which is used to determine and set critical points, surface parameters, and critical distance between points.

Analyzing Interference Volume

Methods Introduced:

- **pfcInterference.InterferenceVolume.ComputeVolume**
- **pfcInterference.InterferenceVolume.Highlight**
- **pfcInterference.InterferenceVolume.GetBoundaries**

The method pfcInterference.InterferenceVolume.ComputeVolume will calculate a value for interfering volume.

The method pfcInterference.InterferenceVolume.Highlight will highlight the interfering volume with the color provided in the argument to the function.

The method pfcInterference.InterferenceVolume.GetBoundaries will return a set of boundary surface descriptors for the interference volume.

Example Code

The sample code in the file UsrInterference.java located at <creo_jlink_loadpoint>/jlink_appls/jlinkexamples finds the interference in an assembly, highlights the interfering surfaces, and highlights calculates the interference volume.

This application finds the interference in an assembly, highlights the interfering surfaces, and highlights calculates the interference volume.

18

Dimensions and Parameters

This chapter describes the J-Link methods and classes that affect dimensions and parameters.

Overview

Dimensions and parameters in PTC Creo Parametric have similar characteristics but also have significant differences. In J-Link, the similarities between dimensions and parameters are contained in the `ModelItem.BaseParameter` interface. This interface allows access to the parameter or dimension value and to information regarding a parameter's designation and modification. The differences between parameters and dimensions are recognizable because `Dimension` inherits from the interface `ModelItem`, and can be assigned tolerances, whereas parameters are not `ModelItems` and cannot have tolerances.

The ParamValue Object

Both parameters and dimension objects contain an object of type `ModelItem.ParamValue`. This object contains the integer, real, string, or Boolean value of the parameter or dimension. Because of the different possible value types that can be associated with a `ParamValue` object there are different methods used to access each value type and some methods will not be applicable for some `ParamValue` objects. If you try to use an incorrect method an exception will be thrown.

Accessing a ParamValue Object

Methods Introduced:

- **pfcModelItem.pfcModelItem.CreateIntParamValue**
- **pfcModelItem.pfcModelItem.CreateDoubleParamValue**
- **pfcModelItem.pfcModelItem.CreateStringParamValue**
- **pfcModelItem.pfcModelItem.CreateBoolParamValue**
- **pfcModelItem.pfcModelItem.CreateNoteParamValue**
- **pfcModelItem.BaseParameter.GetValue**

The `pfcModelItem` utility class contains methods for creating each type of `ParamValue` object. Once you have established the value type in the object, you can change it. The method `pfcModelItem.BaseParameter.GetValue` returns the `ParamValue` associated with a particular parameter or dimension.

A `NoteParamValue` is an integer value that refers to the ID of a specified note. To create a parameter of this type the identified note must already exist in the model.

Accessing the ParamValue Value

Methods Introduced:

- **pfcModelItem.ParamValue.Getdiscr**
- **pfcModelItem.ParamValue.GetIntValue**
- **pfcModelItem.ParamValue.SetIntValue**
- **pfcModelItem.ParamValue.GetDoubleValue**
- **pfcModelItem.ParamValue.SetDoubleValue**
- **pfcModelItem.ParamValue.GetStringValue**
- **pfcModelItem.ParamValue.SetStringValue**
- **pfcModelItem.ParamValue.GetBoolValue**
- **pfcModelItem.ParamValue.SetBoolValue**
- **pfcModelItem.ParamValue.GetNoteId**

The method `pfcModelItem.ParamValue.Getdiscr` returns a enumeration object that identifies the type of value contained in the ParamValue object. Use this information with the `Get` and`Set` methods to access the value. If you use an incorrect `Get` or `Set` method an exception of type `Exceptions.XBadGetParamValue` will be thrown.

Parameter Objects

The following sections describe the J-Link methods that access parameters. The topics are as follows:

- Creating and Accessing Parameters on page 257
- Parameter Selection Options on page 258
- Parameter Information on page 260
- Parameter Restrictions on page 262

Creating and Accessing Parameters

Methods Introduced:

- **pfcModelItem.ParameterOwner.CreateParam**
- **pfcModelItem.ParameterOwner.CreateParamWithUnits**
- **pfcModelItem.ParameterOwner.GetParam**
- **pfcModelItem.ParameterOwner.ListParams**
- **pfcModelItem.ParameterOwner.SelectParam**
- **pfcModelItem.ParameterOwner.SelectParameters**
- **pfcFamily.FamColParam.GetRefParam**

In J-Link, models, features, surfaces, and edges inherit from the `ModelItem.ParameterOwner` , because each of the objects can be assigned parameters in PTC Creo Parametric.

The method `pfcModelItem.ParameterOwner.GetParam` gets a parameter given its name.

The method `pfcModelItem.ParameterOwner.ListParams` returns a sequence of all parameters assigned to the object.

To create a new parameter with a name and a specific value, call the method `pfcModelItem.ParameterOwner.CreateParam`.

To create a new parameter with a name, a specific value, and units, call the method `pfcModelItem.ParameterOwner.CreateParamWithUnits`.

The method `pfcModelItem.ParameterOwner.SelectParam` allows you to select a parameter from the PTC Creo Parametric user interface. The top model from which the parameters are selected must be displayed in the current window.

The method `pfcModelItem.ParameterOwner.SelectParameters` allows you to interactively select parameters from the PTC Creo Parametric **Parameter** dialog box based on the parameter selection options specified by the `ModelItem.ParameterSelectionOptions` object. The top model from which the parameters are selected must be displayed in the current window. Refer to the section Parameter Selection Options on page 258 for more information.

The method `pfcFamily.FamColParam.GetRefParam` returns the reference parameter from the parameter column in a family table.

Parameter Selection Options

Parameter selection options in J-Link are represented by the `ModelItem.ParameterSelectionOptions` .

Methods Introduced:

- **pfcModelItem.pfcModelItem.ParameterSelectionOptions_Create**
- **pfcModelItem.ParameterSelectionOptions.SetAllowContextSelection**
- **pfcModelItem.ParameterSelectionOptions.SetContexts**
- **pfcModelItem.ParameterSelectionOptions.SetAllowMultipleSelections**
- **pfcModelItem.ParameterSelectionOptions.SetSelectButtonLabel**

The method `pfcModelItem.pfcModelItem.ParameterSelectionOptions_Create` creates a new instance of the `ParameterSelectionOptions` object that is used by the method `pfcModelItem.ParameterOwner.SelectParameters()`.

The parameter selection options are as follows:

- `AllowContextSelection`—This boolean attribute indicates whether to allow parameter selection from multiple contexts, or from the invoking parameter owner. By default, it is false and allows selection only from the invoking parameter owner. If it is true and if specific selection contexts are not yet assigned, then you can select the parameters from any context.

  Use the method `pfcModelItem.ParameteSelectionOptions.SetAllowContextSelection` to modify the value of this attribute.
- `Contexts`—The permitted parameter selection contexts in the form of the `ModelItem.ParameterSelectionContexts` object. Use the method `pfcModelItem.ParameterSelectionOptions.SetContexts` to assign the parameter selection context. By default, you can select parameters from any context.
- The types of parameter selection contexts are as follows:
  - `PARAMSELECT_MODEL`—Specifies that the top level model parameters can be selected.
  - `PARAMSELECT_PART`—Specifies that any part's parameters (at any level of the top model) can be selected.
  - `PARAMSELECT_ASM`—Specifies that any assembly's parameters (at any level of the top model) can be selected.
  - `PARAMSELECT_FEATURE`—Specifies that any feature's parameters can be selected.
  - `PARAMSELECT_EDGE`—Specifies that any edge's parameters can be selected.
  - `PARAMSELECT_SURFACE`—Specifies that any surface's parameters can be selected.
  - `PARAMSELECT_QUILT`—Specifies that any quilt's parameters can be selected.
  - `PARAMSELECT_CURVE`—Specifies that any curve's parameters can be selected.
  - `PARAMSELECT_COMPOSITE_CURVE`—Specifies that any composite curve's parameters can be selected.
  - `PARAMSELECT_INHERITED`—Specifies that any inheritance feature's parameters can be selected.
  - `PARAMSELECT_SKELETON`—Specifies that any skeleton's parameters can be selected.
  - `PARAMSELECT_COMPONENT`—Specifies that any component's parameters can be selected.

- `AllowMultipleSelections`—This boolean attribute indicates whether or not to allow multiple parameters to be selected from the dialog box, or only a single parameter. By default, it is true and allows selection of multiple parameters.

  Use the method `pfcModelItem.ParameterSelectionOptions.SetAllowMultipleSelections` to modify this attribute.
- `SelectButtonLabel`—The visible label for the select button in the dialog box.

  Use the method `pfcModelItem.ParameterSelectionOptions.SetSelectButtonLabel` to set the label. If not set, the default label in the language of the active PTC Creo Parametric session is displayed.

Parameter Information

Methods Introduced:

- **pfcModelItem.BaseParameter.GetValue**
- **pfcModelItem.BaseParameter.SetValue**
- **pfcModelItem.Parameter.GetScaledValue**
- **pfcModelItem.Parameter.SetScaledValue**
- **pfcModelItem.Parameter.GetUnits**
- **pfcModelItem.BaseParameter.GetIsDesignated**
- **pfcModelItem.BaseParameter.SetIsDesignated**
- **pfcModelItem.BaseParameter.GetIsModified**
- **pfcModelItem.BaseParameter.ResetFromBackup**
- **pfcModelItem.Parameter.GetDescription**
- **pfcModelItem.Parameter.SetDescription**
- **pfcModelItem.Parameter.GetRestriction**
- **pfcModelItem.Parameter.GetDriverType**
- **pfcModelItem.Parameter.Reorder**
- **pfcModelItem.Parameter.Delete**
- **pfcModelItem.NamedModelItem.GetName**

Parameters inherit methods from the `BaseParameter`, `Parameter` and `NamedModelItem`.

The method pfcModelItem.BaseParameter.GetValue returns the value of the parameter or dimension.

The method pfcModelItem.BaseParameter.SetValue assigns a particular value to a parameter or dimension.

The method pfcModelItem.Parameter.GetScaledValue returns the parameter value in the units of the parameter, instead of the units of the owner model as returned by pfcModelItem.BaseParameter.GetValue.

The method pfcModelItem.Parameter.SetScaledValue assigns the parameter value in the units provided, instead of using the units of the owner model as assumed by pfcModelItem.BaseParameter.GetValue.

The method pfcModelItem.Parameter.GetUnits returns the units assigned to the parameter.

You can access the designation status of the parameter using the methods pfcModelItem.BaseParameter.GetIsDesignated and pfcModelItem.BaseParameter.SetIsDesignated.

The methods pfcModelItem.BaseParameter.GetIsModified and pfcModelItem.BaseParameter.ResetFromBackup enable you to identify a modified parameter or dimension, and reset it to the last stored value. A parameter is said to be "modified" when the value has been changed but the parameter's owner has not yet been regenerated.

The method pfcModelItem.Parameter.GetDescription returns the parameter description, or null, if no description is assigned.

The method pfcModelItem.Parameter.SetDescription assigns the parameter description.

The method pfcModelItem.Parameter.GetRestriction identifies if the parameter’s value is restricted to a certain range or enumeration. It returns the ModelItem.ParameterRestriction object. Refer to the section Parameter Restrictions on page 262 for more information.

The methodpfcModelItem.Parameter.GetDriverType returns the driver type for a material parameter. The driver types are as follows:

- PARAMDRIVER_PARAM—Specifies that the parameter value is driven by another parameter.
- PARAMDRIVER_FUNCTION—Specifies that the parameter value is driven by a function.
- PARAMDRIVER_RELATION—Specifies that the parameter value is driven by a relation. This is equivalent to the value obtained using pfcModelItem.BaseParameter.GetIsRelationDriven for a parameter object type.

The method pfcModelItem.Parameter.Reorder reorders the given parameter to come immediately after the indicated parameter in the **Parameter** dialog box and information files generated by PTC Creo Parametric.

The method pfcModelItem.Parameter.Delete permanently removes a specified parameter.

The method pfcModelItem.NamedModelItem.GetName accesses the name of the specified parameter.

Parameter Restrictions

PTC Creo Parametric allows users to assign specified limitations to the value allowed for a given parameter (wherever the parameter appears in the model). You can only read the details of the permitted restrictions from J-Link, but not modify the permitted values or range of values. Parameter restrictions in J-Link are represented by the interface ModelItem.ParameterRestriction.

Method Introduced:

- **pfcModelItem.ParameterRestriction.GetType**

The method pfcModelItem.ParameterRestriction.GetType returns the ModelItem.RestrictionType object containing the types of parameter restrictions. The parameter restrictions are of the following types:

- PARAMSELECT_ENUMERATION—Specifies that the parameter is restricted to a list of permitted values.
- PARAMSELECT_RANGE—Specifies that the parameter is limited to a specified range of numeric values.

Enumeration Restriction

The PARAMSELECT_ENUMERATION type of parameter restriction is represented by the ModelItem.ParameterEnumeration. It is a child of the ModelItem.ParameterRestriction .

Method Introduced:

- **pfcModelItem.ParameterEnumeration.GetPermittedValues**

The method pfcModelItem.ParameterEnumeration.GetPermittedValues returns a list of permitted parameter values allowed by this restriction in the form of a sequence of the ModelItem.ParamValue objects.

Range Restriction

The PARAMSELECT_RANGE type of parameter restriction is represented by the interface ModelItem.ParameterRange. It is a child of the ModelItem.ParameterRestriction interface.

Methods Introduced:

- **pfcModelItem.ParameterRange.GetMaximum**
- **pfcModelItem.ParameterRange.GetMinimum**
- **pfcModelItem.ParameterLimit.GetType**
- **pfcModelItem.ParameterLimit.GetValue**

The method pfcModelItem.ParameterRange.GetMaximum returns the maximum value limit for the parameter in the form of the ModelItem.ParameterLimit object.

The method pfcModelItem.ParameterRange.GetMinimum returns the minimum value limit for the parameter in the form of the ModelItem.ParameterLimit object.

The method pfcModelItem.ParameterLimit.GetType returns the ModelItem.ParameterLimitType containing the types of parameter limits. The parameter limits are of the following types:

- PARAMLIMIT_LESS_THAN—Specifies that the parameter must be less than the indicated value.
- PARAMLIMIT_LESS_THAN_OR_EQUAL—Specifies that the parameter must be less than or equal to the indicated value.
- PARAMLIMIT_GREATER_THAN—Specifies that the parameter must be greater than the indicated value.
- PARAMLIMIT_GREATER_THAN_OR_EQUAL—Specifies that the parameter must be greater than or equal to the indicated value.

The method pfcModelItem.ParameterLimit.GetValue returns the boundary value of the parameter limit in the form of the ModelItem.ParamValue object.

Example Code: Updating Model Parameters

The sample code in the file pfcParameterExamples.java located at <creo_jlink_loadpoint>/jlink_appls/jlinkexamples contains a single static utility method. This method reads a Java "properties" file and creates or updates model parameters for each property which exists in the file. Since each property value is returned as a String, a utility method parses the String into int, double, or boolean values if possible.

Dimension Objects

Dimension objects include standard PTC Creo Parametric dimensions as well as reference dimensions. Dimension objects enable you to access dimension tolerances and enable you to set the value for the dimension. Reference dimensions allow neither of these actions.

Getting Dimensions

Dimensions and reference dimensions are PTC Creo Parametric model items. See the section Getting ModelItem Objects on page 208 for methods that can return `Dimension` and `RefDimension` objects.

Dimension Information

Methods Introduced:

- **pfcModelItem.BaseParameter.GetValue**
- **pfcModelItem.BaseParameter.SetValue**
- **pfcModelItem.BaseDimension.GetDimValue**
- **pfcModelItem.BaseDimension.SetDimValue**
- **pfcModelItem.BaseParameter.GetIsDesignated**
- **pfcModelItem.BaseParameter.SetIsDesignated**
- **pfcModelItem.BaseParameter.GetIsModified**
- **pfcModelItem.BaseParameter.ResetFromBackup**
- **pfcModelItem.BaseParameter.GetIsRelationDriven**
- **pfcDimension.BaseDimension.GetDimType**
- **pfcDimension.BaseDimension.GetSymbol**
- **pfcDimension.BaseDimension.GetTexts**
- **pfcDimension.BaseDimension.SetTexts**

All the `BaseParameter` methods are accessible to Dimensions as well as Parameters. See the section Parameter Objects on page 257 for brief descriptions.

Note

You cannot set the value or designation status of reference dimension objects.

The methods pfcModelItem.BaseDimension.GetDimValue and pfcModelItem.BaseDimension.SetDimValue access the dimension value as a double. These methods provide a shortcut for accessing the dimensions' values without using a ParamValue object.

The pfcModelItem.BaseParameter.GetIsRelationDriven method identifies whether the part or assembly relations control a dimension.

The method pfcDimension.BaseDimension.GetDimType returns an enumeration object that identifies whether a dimension is linear, radial, angular, or diametrical.

The method pfcDimension.BaseDimension.GetSymbol returns the dimension or reference dimension symbol (that is, “d#” or “rd#”).

The pfcDimension.BaseDimension.GetTexts and pfcDimension.BaseDimension.SetTexts methods allows access to the text strings that precede or follow the dimension value.

Dimension Tolerances

Methods Introduced:

- **pfcDimension.Dimension.GetTolerance**
- **pfcDimension.Dimension.SetTolerance**
- **pfcDimension.pfcDimension.DimTolPlusMinus_Create**
- **pfcDimension.pfcDimension.DimTolSymmetric_Create**
- **pfcDimension.pfcDimension.DimTolLimits_Create**
- **pfcDimension.pfcDimension.DimTolSymSuperscript_Create**
- **pfcDimension.pfcDimension.DimTolISODIN_Create**

Only true dimension objects can have geometric tolerances.

The methods pfcDimension.Dimension.GetTolerance and pfcDimension.Dimension.SetTolerance enable you to access the dimension tolerance. The object types for the dimension tolerance are:

- DimTolLimits—Displays dimension tolerances as upper and lower limits.

> **Note**
>
> This format is not available when only the tolerance value for a dimension is displayed.

- DimTolPlusMinus—Displays dimensions as nominal with plus-minus tolerances. The positive and negative values are independent.

- DimTolSymmetric—Displays dimensions as nominal with a single value for both the positive and the negative tolerance.
- DimTolSymSuperscript—Displays dimensions as nominal with a single value for positive and negative tolerance. The text of the tolerance is displayed in a superscript format with respect to the dimension text.
- DimTolISODIN—Displays the tolerance table type, table column, and table name, if the dimension tolerance is set to a hole or shaft table (DIN/ISO standard).

A null value is similar to the nominal option in PTC Creo Parametric.

To determine whether a given tolerance is plus/minus, symmetric, limits, or superscript use instanceof.

Example Code: Setting Tolerances to a Specified Range

The sample code in the file pfcDimensionExamples.java located at <creo_jlink_loadpoint>/jlink_appls/jlinkexamples shows a utility method that sets angular tolerances to a specified range.

The example code shows a utility method that sets angular tolerances to a specified range. First, the program determines whether the dimension passed to it is angular. If it is, the method gets the dimension value and adds or subtracts the range to it to get the upper and lower limits.The program then initializes a DimTolLimits tolerance object and assigns it to the dimension.

Because the BaseParameter used in the example is a dimension, you know that its ParamValue object must contain a double value. Therefore, you do not have to check the ParamValueType using the method pfcModelItem.pfcParamValue.Getdiscr.

19

Relations

This chapter describes how to access relations on all models and model items in PTC Creo Parametric using the methods provided in J-Link.

Accessing Relations

In J-Link, the set of relations on any model or model item is represented by the `pfcModelItem.RelationOwner`. Models, features, surfaces, and edges inherit from this interface, because each object can be assigned relations in PTC Creo Parametric.

Methods Introduced:

- **pfcModelItem.RelationOwner.RegenerateRelations**
- **pfcModelItem.RelationOwner.DeleteRelations**
- **pfcModelItem.RelationOwner.GetRelations**
- **pfcModelItem.RelationOwner.SetRelations**
- **pfcModelItem.RelationOwner.EvaluateExpression**

The method `pfcModelItem.RelationOwner.RegenerateRelations` regenerates the relations assigned to the owner item. It also determines whether the specified relation set is valid.

The method `pfcModelItem.RelationOwner.DeleteRelations` deletes all the relations assigned to the owner item.

The method `pfcModelItem.RelationOwner.GetRelations` returns the list of initial relations assigned to the owner item as a sequence of strings.

The method `pfcModelItem.RelationOwner.SetRelations` assigns the sequence of strings as the new relations to the owner item.

The method `pfcModelItem.RelationOwner.EvaluateExpression` evaluates the given relations-based expression, and returns the resulting value in the form of the `pfcModelItem.ParamValue` object. Refer to the section The ParamValue Object on page 256 in the chapter Dimensions and Parameters on page 255 for more information on this object.

Example 1: Adding Relations between Parameters in a Solid Model

The sample code in the file `pfcRelationExamples.java` located at `<creo_jlink_loadpoint>/jlink_appls/jlinkexamples` demonstrates how to add relations between parameters in a solid model.

Accessing Post Regeneration Relations

Method Introduced:

- **pfcModel.Model.GetPostRegenerationRelations**
- **pfcModel.Model.RegeneratePostRegenerationRelations**
- **pfcModel.Model.DeletePostRegenerationRelations**

The method `pfcModel.Model.GetPostRegenerationRelations` lists the post-regeneration relations assigned to the model. It can be `NULL`, if not set.

Note

To work with post-regeneration relations, use the post-regeneration relations attribute in the methods `pfcModelItem.RelationOwner.SetRelations`, `pfcModelItem.RelationOwner.RegenerateRelations` and `pfcModelItem.RelationOwner.DeleteRelations`.

You can regenerate the relation sets post-regeneration in a model using the method `fcModel.Model.RegeneratePostRegenerationRelations`.

To delete all the post-regeneration relations in the specified model, call the method `pfcModel.Model.DeletePostRegenerationRelations`.

20

Assemblies and Components

This chapter describes the J-Link functions that access the functions of a PTC Creo Parametric assembly. You must be familiar with the following before you read this section:

- The Selection Object
- Coordinate Systems
- The Geometry section

Structure of Assemblies and Assembly Objects

The object `Assembly` is an instance of `Solid`. The `Assembly` object can therefore be used as input to any of the `Solid` and `Model` methods applicable to assemblies. However assemblies do not contain solid geometry items. The only geometry in the assembly is datums (points, planes, axes, coordinate systems, curves, and surfaces). Therefore solid assembly features such as holes and slots will not contain active surfaces or edges in the assembly model.

The solid geometry of an assembly is contained in its components. A component is a feature of type `pfcComponentFeat.ComponentFeat`, which is a reference to a part or another assembly, and a set of parametric constraints for determining its geometrical location within the parent assembly.

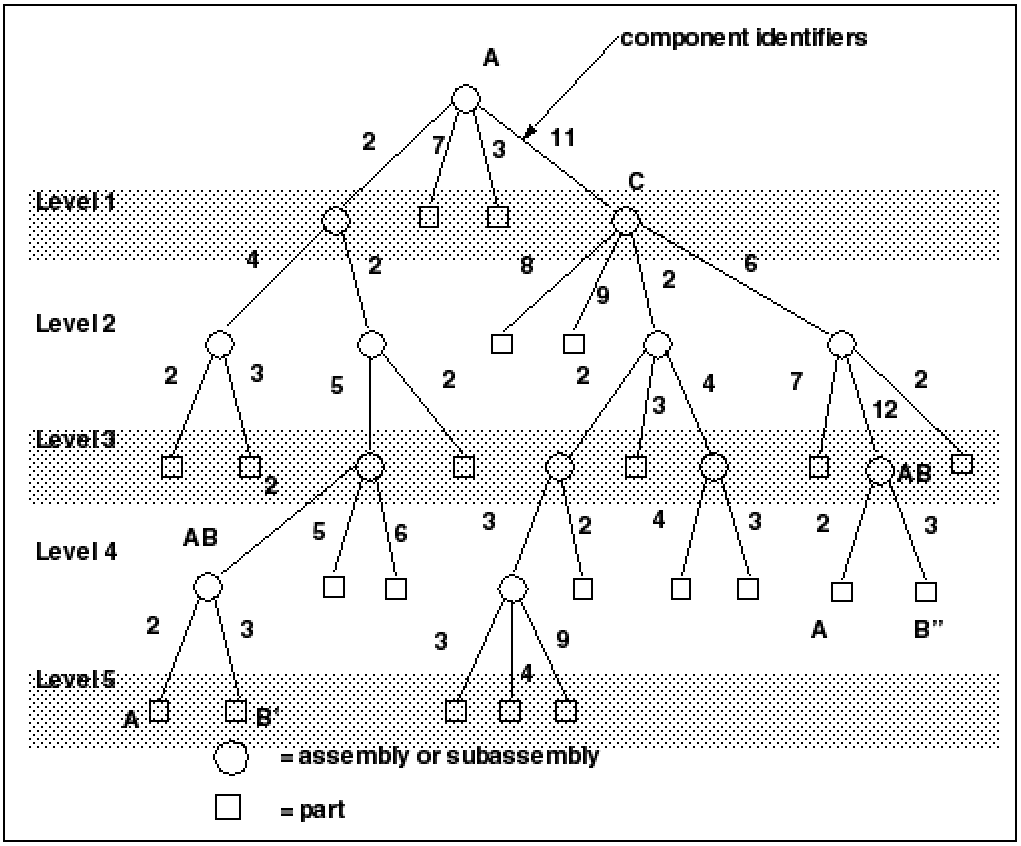
Assembly features that are solid, such as holes and slots, and therefore affect the solid geometry of parts in the assembly hierarchy, do not themselves contain the geometry items that describe those modifications. These items are always contained in the parts whose geometry is modified, within local features created for that purpose.

The important J-Link functions for assemblies are those that operate on the components of an assembly. The object `ComponentFeat`, which is an instance of `Feature` is defined for that purpose. Each assembly component is treated as a variety of feature, and the integer identifier of the component is also the feature identifier.

An assembly can contain a hierarchy of assemblies and parts at many levels, in which some assemblies and parts may appear more than once. To identify the role of any database item in the context of the root assembly, it is not sufficient to have the integer identifier of the item and the handle to its owning part or assembly, as would be provided by its `Feature` description.

It is also necessary to give the full path of the assembly-component references down from the root assembly to the part or assembly that owns the database item. This is the purpose of the object `ComponentPath`, which is used as the input to J-Link assembly functions.

The following figure shows an assembly hierarchy with two examples of the contents of a `ComponentPath` object.



In the assembly shown in the figure, subassembly C is component identifier 11 within assembly A, Part B is component identifier 3 within assembly AB, and so on. The subassembly AB occurs twice. To refer to the two occurrences of part B, use the following:

```
(?)Component B'              Component B"
ComponentIds.get(0) = 2   ComponentIds.get(1) = 11
ComponentIds.get(1) = 2   ComponentIds.get(2) = 6
ComponentIds.get(2) = 5   ComponentIds.get(3) = 12
ComponentIds.get(3) = 2   ComponentIds.get(4) = 3
ComponentIds.get(4) = 3
```

The object `ComponentPath` is one of the main portions of the `Selection` object.

Assembly Components

Methods Introduced:

- **pfcComponentFeat.ComponentFeat.GetIsBulkitem**
- **pfcComponentFeat.ComponentFeat.GetIsSubstitute**
- **pfcComponentFeat.ComponentFeat.GetCompType**
- **pfcComponentFeat.ComponentFeat.SetCompType**

- **pfcComponentFeat.ComponentFeat.GetModelDescr**
- **pfcComponentFeat.ComponentFeat.GetIsPlaced**
- **pfcComponentFeat.ComponentFeat.SetIsPlaced**
- **pfcComponentFeat.ComponentFeat.GetIsPackaged**
- **pfcComponentFeat.ComponentFeat.GetIsUnderconstrained**
- **pfcComponentFeat.ComponentFeat.GetIsFrozen**
- **pfcComponentFeat.ComponentFeat.GetPosition**
- **pfcComponentFeat.ComponentFeat.CopyTemplateContents**
- **pfcComponentFeat.ComponentFeat.CreateReplaceOp**

The method `pfcComponentFeat.ComponentFeat.GetIsBulkitem` identifies whether an assembly component is a bulk item. A bulk item is a non-geometric assembly feature that should appear in an assembly bill of materials.

The method `pfcComponentFeat.ComponentFeat.GetIsSubstitute` returns a true value if the component is substituted, else it returns a false. When you substitute a component in a simplified representation, you temporarily exclude the substituted component and superimpose the substituting component in its place.

The method `pfcComponentFeat.ComponentFeat.GetCompType` returns the type of the assembly component.

The method `pfcComponentFeat.ComponentFeat.SetCompType` enables you to set the type of the assembly component. The component type identifies the purpose of the component in a manufacturing assembly.

The method `pfcComponentFeat.ComponentFeat.GetModelDescr` returns the model descriptor of the component part or subassembly.

Note

From Pro/ENGINEER Wildfire 4.0 onwards, the method `pfcComponentFeat.ComponentFeat.GetModelDescr` throws an exception `pfcExceptions.XtoolkitCantOpen` if called on an assembly component whose immediate generic is not in session. Handle this exception and typecast the assembly component as `pfcSolid.Solid`, which in turn can be typecast as `pfcFamily.FamilyMember`, and use the method `pfcFamily.FamilyMember.GetImmediateGenericInfo` to get the model descriptor of the immediate generic model. If you wish to switch off this behavior and continue to run legacy applications in the pre-Wildfire 4.0 mode, set the configuration option `retrieve_instance_dependencies` to `instance_and_generic_deps`.

The method `pfcCompontentFeat.ComponentFeat.GetIsPlaced` determines whether the component is placed.

The method `pfcCompontentFeat.ComponentFeat.SetIsPlaced` forces the component to be considered placed. The value of this parameter is important in assembly Bill of Materials.

Note

Once a component is constrained or packaged, it cannot be made unplaced again.

A component of an assembly that is either partially constrained or unconstrained is known as a packaged component. Use the method `pfcCompontentFeat.ComponentFeat.GetIsPackaged` to determine if the specified component is packaged.

The method `pfcCompontentFeat.ComponentFeat.GetIsUnderconstrained` determines if the specified component is underconstrained, that is, it possesses some constraints but is not fully constrained.

The method `pfcCompontentFeat.ComponentFeat.GetIsFrozen` determines if the specified component is frozen. The frozen component behaves similar to the packaged component and does not follow the constraints that you specify.

The method `pfcCompontentFeat.ComponentFeat.GetPosition` retrieves the component's initial position before constraints and movements have been applied. If the component is packaged this position is the same as the constraint's actual position. This method modifies the assembly component data but does not regenerate the assembly component. To regenerate the component, use the method `pfcComponentFeat.ComponentFeat.Regenerate`.

The method `pfcComponentFeat.ComponentFeat.CopyTemplateContents` copies the template model into the model of the specified component.

The method `pfcCompontentFeat.ComponentFeat.CreateReplaceOp` creates a replacement operation used to swap a component automatically with a related component. The replacement operation can be used as an argument to `pfcSolid.Solid.ExecuteFeatureOps`.

Example Code: Replacing Instances

The sample code in the file `pfcComponentFeatExamples.java` located at <creo_jlink_loadpoint>/jlink_appls/jlinkexamples contains a single static utility method. This method takes an assembly for an argument. It searches through the assembly for all components that are instances of the model "bolt". It then replaces all such occurrences with a different instance of bolt.

Regenerating an Assembly Component

Method Introduced:

- **pfcComponentFeat.ComponentFeat.Regenerate**

The method `pfcComponentFeat.ComponentFeat.Regenerate` regenerates an assembly component. The method regenerates the assembly component just as in an interactive PTC Creo Parametric session.

Creating a Component Path

Methods Introduced

- **pfcAssembly.pfcAssembly.CreateComponentPath**

The method `pfcAssembly.pfcAssembly.CreateComponentPath` returns a component path object, given the Assembly model and the integer id path to the desired component.

Component Path Information

Methods Introduced:

- **pfcAssembly.ComponentPath.GetRoot**
- **pfcAssembly.ComponentPath.SetRoot**
- **pfcAssembly.ComponentPath.GetComponentIds**
- **pfcAssembly.ComponentPath.SetComponentIds**
- **pfcAssembly.ComponentPath.GetLeaf**
- **pfcAssembly.ComponentPath.GetTransform**
- **pfcAssembly.ComponentPath.SetTransform**
- **pfcAssembly.ComponentPath.GetIsVisible**

The method `pfcAssembly.ComponentPath.GetRoot` returns the assembly at the head of the component path object.

The method `pfcAssembly.ComponentPath.SetRoot` setsthe assembly at the head of the component path object as the root assembly.

The method `pfcAssembly.ComponentPath.GetComponentIds` returns the sequence of ids which is the path to the particular component.

The method `pfcAssembly.ComponentPath.SetComponentIds` sets the path from the root assembly to the component through various subassemblies containing this component.

The method `pfcAssembly.ComponentPath.GetLeaf` returns the solid model at the end of the component path.

The method `pfcAssembly.ComponentPath.GetTransform` returns the coordinate system transformation between the assembly and the particular component. It has an option to provide the transformation from bottom to top, or from top to bottom. This method describes the current position and the orientation of the assembly component in the root assembly.

The method `pfcAssembly.ComponentPath.SetTransform` applies a temporary transformation to the assembly component, similar to the transformation that takes place in an exploded state. The transformation will only be applied if the assembly is using DynamicPositioning.

The method `pfcAssembly.ComponentPath.GetIsVisible` identifies if a particular component is visible in any simplified representation.

Assembling Components

Methods Introduced:

- **pfcAssembly.Assembly.AssembleComponent**
- **pfcAssembly.Assembly.AssembleByCopy**
- **pfcComponentFeat.ComponentFeat.GetConstraints**
- **pfcComponentFeat.ComponentFeat.SetConstraints**

The method `pfcAssembly.Assembly.AssembleComponent` adds a specified component model to the assembly at the specified initial position. The position is specified in the format defined by the interface`pfcBase.Transform3D`. Specify the orientation of the three axes and the position of the origin of the component coordinate system, with respect to the target assembly coordinate system.

The method `pfcAssembly.Assembly.AssembleByCopy` creates a new component in the specified assembly by copying from the specified component. If no model is specified, then the new component is created empty. The input parameters for this method are:

- *LeaveUnplaced*—If true the component is unplaced. If false the component is placed at a default location in the assembly. Unplaced components belong to an assembly without being assembled or packaged. These components appear in the model tree, but not in the graphic window. Unplaced components can be constrained or packaged by selecting them from the model tree for redefinition. When its parent assembly is retrieved into memory, an unplaced component is also retrieved.

- *ModelToCopy*—Specify the model to be copied into the assembly
- *NewModelName*—Specify a name for the copied model

The method `pfcComponentFeat.ComponentFeat.GetConstraints` retrieves the constraints for a given assembly component.

The method `pfcComponentFeat.ComponentFeat.SetConstraints` allows you to set the constraints for a specified assembly component. The input parameters for this method are:

- *Constraints*—Constraints for the assembly component. These constraints are explained in detail in the later sections.
- *ReferenceAssembly*—The path to the owner assembly, if the constraints have external references to other members of the top level assembly. If the constraints are applied only to the assembly component then the value of this parameter should be null.

This method modifies the component feature data but does not regenerate the assembly component. To regenerate the assembly use the method `pfcSolid.Solid.Regenerate`.

Constraint Attributes

Methods Introduced:

- **pfcComponentFeat.pfcComponentFeat.ConstraintAttributes_Create**
- **pfcComponentFeat.ConstraintAttributes.GetForce**
- **pfcComponentFeat.ConstraintAttributes.SetForce**
- **pfcComponentFeat.ConstraintAttributes.GetIgnore**
- **pfcComponentFeat.ConstraintAttributes.SetIgnore**

The method `pfcComponentFeat.pfcComponentFeat.ConstraintAttributes_Create` returns the constraint attributes object based on the values of the following input parameters:

- *Ignore*—Constraint is ignored during regeneration. Use this capability to store extra constraints on the component, which allows you to quickly toggle between different constraints.
- *Force*—Constraint has to be forced for line and point alignment.
- *None*—No constraint attributes. This is the default value.

Use the `Get` methods to retrieve the values of the input parameters specified above and the `Set` methods to modify the values of these input parameters.

Assembling a Component Parametrically

You can position a component relative to its neighbors (components or assembly features) so that its position is updated as its neighbors move or change. This is called parametric assembly. PTC Creo Parametric allows you to specify constraints to determine how and where the component relates to the assembly. You can add as many constraints as you need to make sure that the assembly meets the design intent.

Methods Introduced:

- **pfcComponentFeat.pfcComponentFeat.ComponentConstraint_Create**
- **pfcComponentFeat.ComponentConstraint.GetType**
- **pfcComponentFeat.ComponentConstraint.SetType**
- **pfcComponentFeat.ComponentConstraint.SetAssemblyReference**
- **pfcComponentFeat.ComponentConstraint.GetAssemblyReference**
- **pfcComponentFeat.ComponentConstraint.SetAssemblyDatumSide**
- **pfcComponentFeat.ComponentConstraint.GetAssemblyDatumSide**
- **pfcComponentFeat.ComponentConstraint.SetComponentReference**
- **pfcComponentFeat.ComponentConstraint.GetComponentReference**
- **pfcComponentFeat.ComponentConstraint.SetComponentDatumSide**
- **pfcComponentFeat.ComponentConstraint.GetComponentDatumSide**
- **pfcComponentFeat.ComponentConstraint.SetOffset**
- **pfcComponentFeat.ComponentConstraint.GetOffset**
- **pfcComponentFeat.ComponentConstraint.SetAttributes**
- **pfcComponentFeat.ComponentConstraint.GetAttributes**
- **pfcComponentFeat.ComponentConstraint.SetUserDefinedData**
- **pfcComponentFeat.ComponentConstraint.GetUserDefinedData**

The method `pfcComponentFeat.pfcComponentFeat.ComponentConstraint_Create` returns the component constraint object having the following parameters:

- *ComponentConstraintType*—Using the `TYPE` options, you can specify the placement constraint types. They are as follows:
  - `ASM_CONSTRAINT_MATE`—Use this option to make two surfaces touch one another, that is coincident and facing each other.

- ASM_CONSTRAINT_MATE_OFF—Use this option to make two planar surfaces parallel and facing each other.
- ASM_CONSTRAINT_ALIGN—Use this option to make two planes coplanar, two axes coaxial and two points coincident. You can also align revolved surfaces or edges.
- ASM_CONSTRAINT_ALIGN_OFF—Use this option to align two planar surfaces at an offset.
- ASM_CONSTRAINT_INSERT—Use this option to insert a "male" revolved surface into a ``female" revolved surface, making their respective axes coaxial.
- ASM_CONSTRAINT_ORIENT—Use this option to make two planar surfaces to be parallel in the same direction.
- ASM_CONSTRAINT_CSYS—Use this option to place a component in an assembly by aligning the coordinate system of the component with the coordinate system of the assembly.
- ASM_CONSTRAINT_TANGENT—Use this option to control the contact of two surfaces at their tangents.
- ASM_CONSTRAINT_PNT_ON_SRF—Use this option to control the contact of a surface with a point.
- ASM_CONSTRAINT_EDGE_ON_SRF—Use this option to control the contact of a surface with a straight edge.
- ASM_CONSTRAINT_DEF_PLACEMENT—Use this option to align the default coordinate system of the component to the default coordinate system of the assembly.
- ASM_CONSTRAINT_SUBSTITUTE—Use this option in simplified representations when a component has been substituted with some other model
- ASM_CONSTRAINT_PNT_ON_LINE—Use this option to control the contact of a line with a point.
- ASM_CONSTRAINT_FIX—Use this option to force the component to remain in its current packaged position.
- ASM_CONSTRAINT_AUTO—Use this option in the user interface to allow an automatic choice of constraint type based upon the references.

- *AssemblyReference*—A reference in the assembly.
- *AssemblyDatumSide*—Orientation of the assembly. This can have the following values:
  - Yellow—The primary side of the datum plane which is the default direction of the arrow.

- `Red`—The secondary side of the datum plane which is the direction opposite to that of the arrow.

- *ComponentReference*—A reference on the placed component.
- *ComponentDatumSide*—Orientation of the assembly component. This can have the following values:
  - `Yellow`—The primary side of the datum plane which is the default direction of the arrow.
  - `Red`—The secondary side of the datum plane which is the direction opposite to that of the arrow.
- *Offset*—The mate or align offset value from the reference.
- *Attributes*—Constraint attributes for a given constraint
- *UserDefinedData*—A string that specifies user data for the given constraint.

Use the `Get` methods to retrieve the values of the input parameters specified above and the `Set` methods to modify the values of these input parameters.

Redefining and Rerouting Assembly Components

These functions enable you to reroute previously assembled components, just as in an interactive PTC Creo Parametric session.

Methods Introduced:

- **pfcComponentFeat.ComponentFeat.RedefineThroughUI**
- **pfcComponentFeat.ComponentFeat.MoveThroughUI**

The method `pfcComponentFeat.ComponentFeat.RedefineThroughUI` must be used in interactive J-Link applications. This method displays the PTC Creo Parametric **Constraint** dialog box. This enables the end user to redefine the constraints interactively. The control returns to J-Link application when the user selects **OK** or **Cancel** and the dialog box is closed.

The method `pfcComponentFeat.ComponentFeat.MoveThroughUI` invokes a dialog box that prompts the user to interactively reposition the components. This interface enables the user to specify the translation and rotation values. The control returns to J-Link application when the user selects **OK** or **Cancel** and the dialog box is closed.

Example: Component Constraints

The sample code in the file pfcComponentFeatExamples.java located at <creo_jlink_loadpoint>/jlink_appls/jlinkexamples displays each constraint of the component visually on the screen, and includes a text explanation for each constraint.

Example: Assembling Components

The sample code in the file pfcComponentFeatExamples.java located at <creo_jlink_loadpoint>/jlink_appls/jlinkexamples demonstrates how to assemble a component into an assembly, and how to constrain the component by aligning datum planes. If the complete set of datum planes is not found, the function will show the component constraint dialog to the user to allow them to adjust the placement.

Exploded Assemblies

These methods enable you to determine and change the explode status of the assembly object.

Methods Introduced:

- **pfcAssembly.Assembly.GetIsExploded**
- **pfcAssembly.Assembly.Explode**
- **pfcAssembly.Assembly.UnExplode**
- **pfcAssembly.Assembly.GetActiveExplodedState**
- **pfcAssembly.Assembly.GetDefaultExplodedState**
- **pfcAssembly.ExplodedState.Activate**

The methods pfcAssembly.Assembly.Explode and pfcAssembly.Assembly.UnExplode enable you to determine and change the explode status of the assembly object.

The method pfcAssembly.Assembly.GetIsExploded reports whether the specified assembly is currently exploded. Use this method in the assembly mode only. The exploded status of an assembly depends on the mode. If an assembly is opened in the drawing mode, the state of the assembly in the drawing view is displayed. The drawing view does not represent the actual exploded state of the assembly.

The method pfcAssembly.Assembly.GetActiveExplodedState returns the current active explode state.

The method pfcAssembly.Assembly.GetDefaultExplodedState returns the default explode state.

The method `pfcAssembly.ExplodedState.Activate` activates the specified explode state representation.

Skeleton Models

Skeleton models are a 3-dimensional layout of the assembly. These models are holders or distributors of critical design information, and can represent space requirements, important mounting locations, and motion.

Methods Introduced:

- **pfcAssembly.Assembly.AssembleSkeleton**
- **pfcAssembly.Assembly.AssembleSkeletonByCopy**
- **pfcAssembly.Assembly.GetSkeleton**
- **pfcAssembly.Assembly.DeleteSkeleton**
- **pfcSolid.Solid.GetIsSkeleton**

The method `pfcAssembly.Assembly.AssembleSkeleton` adds an existing skeleton model to the specified assembly.

The method `pfcAssembly.Assembly.GetSkeleton` returns the skeleton model of the specified assembly.

The method `pfcAssembly.Assembly.DeleteSkeleton` deletes a skeleton model component from the specified assembly.

The method `pfcAssembly.Assembly.AssembleSkeletonByCopy` adds a specified skeleton model to the assembly. The input parameters for this method are:

- *SkeletonToCopy*—Specify the skeleton model to be copied into the assembly
- *NewSkeletonName*—Specify a name for the copied skeleton model

The method `pfcSolid.Solid.GetIsSkeleton` determines if the specified part model is a skeleton model or a concept model. It returns a true if the model is a skeleton else it returns a false.

21

Family Tables

This chapter describes how to use J-Link classes and methods to access and manipulate family table information.

Working with Family Tables

J-Link provides several methods for accessing family table information. Because every model inherits from the `pfcFamily.FamilyMember`, every model can have a family table associated with it.

Accessing Instances

Methods Introduced:

- **pfcFamily.FamilyMember.GetParent**
- **pfcFamily.FamilyMember.GetImmediateGenericInfo**
- **pfcFamily.FamilyMember.GetTopGenericInfo**
- **pfcFamily.FamilyTableRow.CreateInstance**
- **pfcFamily.FamilyMember.ListRows**
- **pfcFamily.FamilyMember.GetRow**
- **pfcFamily.FamilyMember.RemoveRow**
- **pfcFamily.FamilyTableRow.GetInstanceName**
- **pfcFamily.FamilyTableRow.GetIsLocked**
- **pfcFamily.FamilyTableRow.SetIsLocked**

To get the generic model for an instance, call the method `pfcFamily.FamilyMember.GetParent`.

From Pro/ENGINEER Wildfire 4.0 onwards, the behavior of the method `pfcFamily.FamilyMember.GetParent` has changed as a result of performance improvement in family table retrieval mechanism. When you now call the method `pfcFamily.FamilyMember.GetParent`, it throws an exception `pfcExceptions.XToolkitCantOpen`, if the immediate generic of a model instance in a nested family table is currently not in session. Handle this exception and use the method `pfcFamily.FamilyMember.GetImmediateGenericInfo` to get the model descriptor of the immediate generic model. This information can be used to retrieve the immediate generic model.

If you wish to switch off the above behavior and continue to run legacy applications in the pre-Wildfire 4.0 mode, set the configuration option `retrieve_instance_dependencies` to `instance_and_generic_deps`.

To get the model descriptor of the top generic model, call the method `pfcFamily.FamilyMember.GetTopGenericInfo`.

Similarly, the method `pfcFamily.FamilyTableRow.CreateInstance` returns an instance model created from the information stored in the `FamilyTableRow` object.

The method `pfcFamily.FamilyMember.ListRows` returns a sequence of all rows in the family table, whereas `pfcFamily.FamilyMember.GetRow` gets the row object with the name you specify.

Use the method `pfcFamily.FamilyMember.RemoveRow` to permanently delete the row from the family table.

The method `pfcFamily.FamilyTableRow.GetInstanceName` returns the name that corresponds to the invoking row object.

To control whether the instance can be changed or removed, call the methods `pfcFamily.FamilyTableRow.GetIsLocked` and `pfcFamily.FamilyTableRow.SetIsLocked`.

Accessing Columns

Methods Introduced:

- **pfcFamily.FamilyMember.ListColumns**
- **pfcFamily.FamilyMember.GetColumn**
- **pfcFamily.FamilyMember.RemoveColumn**
- **pfcFamily.FamilyTableColumn.GetSymbol**
- **pfcFamily.FamilyTableColumn.GetType**
- **pfcFamily.FamColModelItem.GetRefItem**
- **pfcFamily.FamColParam.GetRefParam**

The method `pfcFamily.FamilyMember.ListColumns` returns a sequence of all columns in the family table.

The method `pfcFamily.FamilyMember.GetColumn` returns a family table column, given its symbolic name.

To permanently delete the column from the family table and all changed values in all instances, call the method `pfcFamily.FamilyMember.RemoveColumn`.

The method `pfcFamily.FamilyTableColumn.GetSymbol` returns the string symbol at the top of the column, such as `D4` or `F5`.

The method `pfcFamily.FamilyTableColumn.GetType` returns an enumerated value indicating the type of parameter governed by the column in the family table.

The method `pfcFamily.FamColModelItem.GetRefItem` returns the `ModelItem` (`Feature` or `Dimension`) controlled by the column, whereas `pfcFamily.FamColParam.GetRefParam` returns the `Parameter` controlled by the column.

Accessing Cell Information

Methods Introduced:

- **pfcFamily.FamilyMember.GetCell**
- **pfcFamily.FamilyMember.GetCellIsDefault**
- **pfcFamily.FamilyMember.SetCell**
- **pfcModelItem.ParamValue.GetStringValue**
- **pfcModelItem.ParamValue.GetIntValue**
- **pfcModelItem.ParamValue.GetDoubleValue**
- **pfcModelItem.ParamValue.GetBoolValue**

The method `pfcFamily.FamilyMember.GetCell` returns a string `ParamValue` that corresponds to the cell at the intersection of the row and column arguments. Use the method `pfcFamily.FamilyMember.GetCellIsDefault` to check if the value of the specified cell is the default value, which is the value of the specified cell in the generic model.

The method `pfcFamily.FamilyMember.SetCell` assigns a value to a column in a particular family table instance.

The `pfcModelItem.ParamValue.GetStringValue`, `pfcModelItem.ParamValue.GetIntValue`, `pfcModelItem.ParamValue.GetDoubleValue`, and `pfcModelItem.ParamValue.GetBoolValue` methods are used to get the different types of parameter values.

Creating Family Table Instances

Methods Introduced:

- **pfcFamily.FamilyMember.AddRow**
- **pfcModelItem.pfcModelItem.CreateStringParamValue**
- **pfcModelItem.pfcModelItem.CreateIntParamValue**
- **pfcModelItem.pfcModelItem.CreateDoubleParamValue**
- **pfcModelItem.pfcModelItem.CreateBoolParamValue**

Use the method `pfcFamily.FamilyMember.AddRow` to create a new instance with the specified name, and, optionally, the specified values for each column. If you do not pass in a set of values, the value `*` will be assigned to each column. This value indicates that the instance uses the generic value.

Creating Family Table Columns

Methods Introduced:

- **pfcFamily.FamilyMember.CreateDimensionColumn**
- **pfcFamily.FamilyMember.CreateParamColumn**
- **pfcFamily.FamilyMember.CreateFeatureColumn**
- **pfcFamily.FamilyMember.CreateComponentColumn**
- **pfcFamily.FamilyMember.CreateCompModelColumn**
- **pfcFamily.FamilyMember.CreateGroupColumn**
- **pfcFamily.FamilyMember.CreateMergePartColumn**
- **pfcFamily.FamilyMember.CreateColumn**
- **pfcFamily.FamilyMember.AddColumn**
- **pfcModelItem.pfcModelItem.CreateStringParamValue**
- **pfcModelItem.ParamValues.create**

The above methods initialize a column based on the input argument. These methods assign the proper symbol to the column header.

The method `pfcFamily.FamilyMember.CreateColumn` creates a new column given a properly defined symbol and column type. The results of this call should be passed to the method `pfcFamily.FamilyMember.AddColumn` to add the column to the model's family table.

The method `pfcFamily.FamilyMember.AddColumn` adds the column to the family table. You can specify the values; if you pass nothing for the values, the method assigns the value `*` to each instance to accept the column’s default value.

Example Code: Adding Dimensions to a Family Table

The sample code in the file `pfcFamilyMemberExamples.java` located at `<creo_jlink_loadpoint>/jlink_appls/jlinkexamples` shows a utility method that adds all the dimensions to a family table. The program lists the dependencies of the assembly and loops through each dependency, assigning the model to a new `FamColDimension` column object. All the dimensions, parameters, features, and components could be added to the family table using a similar method.

22

Action Listeners

This chapter describes the J-Link methods that enable you to use action listeners.

J-Link Action Listeners

An `ActionListener` in Java is a class that is assigned to respond to certain events. In `J-Link`, you can assign action listeners to respond to events involving the following tasks:

- Changing windows
- Changing working directories
- Model operations
- Regenerating
- Creating, deleting, and redefining features
- Checking for regeneration failures

All action listeners in J-Link are defined by these classes:

- Interface—Named <Object>ActionListener. This interface defines the methods that can respond to various events.
- Default class—Named Default<Object>ActionListener. This class has every available method overridden by an empty implementation. You create your own action listeners by extending the default class and overriding the methods for events that interest you.

Note

When notifications are set in J-Link applications, every time an event is triggered, notification messages are added to the trail files. From Creo Parametric 2.0 M210 onward, a new environment variable `PROTK_LOG_DISABLE` enables you to disable this behavior. When set to `true`, the notifications messages are not added to the trail files.

Creating an ActionListener Implementation

You can create a proper `ActionListener` class using either of the following methods:

Define a separate class within the java file.

```
Example:
    public class MyApp {
        session.AddActionListener (new SolidAL1());
    }

    class SolidAL1 extends DefaultSolidActionListener {
```

```
        // Include overridden methods here
    }
```

To use your action listener in different Java applications, define it in a separate file.

```
Example:
    MyApp.java:
    import solidAL1;

    public class MyApp {
        session.AddActionListener (new SolidAL1());
    }

    SolidAL1.java:
    public class SolidAL1 extends DefaultSolidActionListener {
        // Include overridden methods here.
    }
```

Action Sources

Methods introduced:

- **pfcBase.ActionSource.AddActionListener**
- **pfcBase.ActionSource.RemoveActionListener**

Many J-Link classes inherit the `ActionSource` interface, but only the following classes currently make calls to the methods of registered `ActionListeners`:

- `pfcSession.Session`
  - Session Action Listener
  - Model Action Listener
  - Solid Action Listener
  - Model Event Action Listener
  - Feature Action Listener
- `pfcCommand.UICommand`
  - UI Action Listener
- `pfcModel.Model` (and it’s subclasses)
  - Model Action Listener
  - Parameter Action Listener
- `pfcSolid.Solid` (and it’s subclasses)
  - Solid Action Listener

- Feature Action Listener
- `pfcFeature.Feature` (and it's subclasses)
    - Feature Action Listener

> **Note**
>
> Assigning an action listener to a source not related to it will not cause an error but the listener method will never be called.

Types of Action Listeners

The following sections describe the different kinds of action listeners: session, UI command, solid, and feature.

Session Level Action Listeners

Methods introduced:

- **pfcSession.SessionActionListener.OnAfterDirectoryChange**
- **pfcSession.SessionActionListener.OnAfterWindowChange**
- **pfcSession.SessionActionListener.OnAfterModelDisplay**
- **pfcSession.SessionActionListener.OnBeforeModelErase**
- **pfcSession.SessionActionListener.OnBeforeModelDelete**
- **pfcSession.SessionActionListener.OnBeforeModelRename**
- **pfcSession.SessionActionListener.OnBeforeModelSave**
- **pfcSession.SessionActionListener.OnBeforeModelPurge**
- **pfcSession.SessionActionListener.OnBeforeModelCopy**
- **pfcSession.SessionActionListener.OnAfterModelPurge**

The `pfcSession.SessionActionListener.OnAfterDirectoryChange` method activates after the user changes the working directory. This method takes the new directory path as an argument.

The `pfcSession.SessionActionListener.OnAfterWindowChange` method activates when the user activates a window other than the current one. Pass the new window to the method as an argument.

The `pfcSession.SessionActionListener.OnAfterModelDisplay` method activates every time a model is displayed in a window.

Note

Model display events happen when windows are moved, opened and closed, repainted, or the model is regenerated. The event can occur more than once in succession.

The methods `pfcSession.SessionActionListener.OnBeforeModelErase`, `pfcSession.SessionActionListener.OnBeforeModelRename`, `pfcSession.SessionActionListener.OnBeforeModelSave`, and `pfcSession.SessionActionListener.OnBeforeModelCopy` take special arguments. They are designed to allow you to fill in the arguments and pass this data back to PTC Creo Parametric. The model names placed in the descriptors will be used by PTC Creo Parametric as the default names in the user interface.

UI Command Action Listeners

Methods introduced:

- **pfcSession.Session.UICreateCommand**
- **pfcCommand.UICommandActionListener.OnCommand**

The `pfcSession.Session.UICreateCommand` method takes a `UICommandActionListener` argument and returns a `UICommand` action source with that action listener already registered. This `UICommand` object is subsequently passed as an argument to the `Session.AddUIButton` method that adds a command button to a PTC Creo Parametric menu. The `pfcCommand.UICommandActionListener.OnCommand` method of the registered `IpfcUICommandActionListener` is called whenever the command button is clicked.

Model Level Action listeners

Methods introduced:

- **pfcModel.ModelActionListener.OnAfterModelSave**
- **pfcModel.ModelEventActionListener.OnAfterModelCopy**
- **pfcModel.ModelEventActionListener.OnAfterModelRename**
- **pfcModel.ModelEventActionListener.OnAfterModelErase**
- **pfcModel.ModelEventActionListener.OnAfterModelDelete**
- **pfcModel.ModelActionListener.OnAfterModelRetrieve**

- **pfcModel.ModelActionListener.OnBeforeModelDisplay**
- **pfcModel.ModelActionListener.OnAfterModelCreate**
- **pfcModel.ModelActionListener.OnAfterModelSaveAll**
- **pfcModel.ModelEventActionListener.OnAfterModelCopyAll**
- **pfcModel.ModelActionListener.OnAfterModelEraseAll**
- **pfcModel.ModelActionListener.OnAfterModelDeleteAll**
- **pfcModel.ModelActionListener.OnAfterModelRetrieveAll**

Methods ending in `All` are called after any event of the specified type. The call is made even if the user did not explicitly request that the action take place. Methods that do not end in `All` are only called when the user specifically requests that the event occurs.

The method `pfcModel.ModelActionListener.OnAfterModelSave` is called after successfully saving a model.

The method `pfcModel.ModelEventActionListener.OnAfterModelCopy` is called after successfully copying a model.

The method `pfcModel.ModelEventActionListener.OnAfterModelRename` is called after successfully renaming a model.

The method `pfcModel.ModelEventActionListener.OnAfterModelErase` is called after successfully erasing a model.

The method `pfcModel.ModelEventActionListener.OnAfterModelDelete` is called after successfully deleting a model.

The method `pfcModel.ModelActionListener.OnAfterModelRetrieve` is called after successfully retrieving a model.

The method `pfcModel.ModelActionListener.OnBeforeModelDisplay` is called before displaying a model.

Note

The method `pfcModel.ModelActionListener.OnBeforeModelDisplay` is not supported in asynchronous mode.

The method `pfcModel.ModelActionListener.OnAfterModelCreate` is called after the successful creation of a model.

Solid Level Action Listeners

Methods introduced:

- **pfcSolid.SolidActionListener.OnBeforeRegen**
- **pfcSolid.SolidActionListener.OnAfterRegen**
- **pfcSolid.SolidActionListener.OnBeforeUnitConvert**
- **pfcSolid.SolidActionListener.OnAfterUnitConvert**
- **pfcSolid.SolidActionListener.OnBeforeFeatureCreate**
- **pfcSolid.SolidActionListener.OnAfterFeatureCreate**
- **pfcSolid.SolidActionListener.OnAfterFeatureDelete**

The `pfcSolid.SolidActionListener.OnBeforeRegen` and `pfcSolid.SolidActionListener.OnAfterRegen` methods occur when the user regenerates a solid object within the `ActionSource` to which the listener is assigned. These methods take the first feature to be regenerated and a handle to the `Solid` object as arguments. In addition, the method `pfcSolid.SolidActionListener.OnAfterRegenerate` includes a Boolean argument that indicates whether regeneration was successful.

Note

- It is not recommended to modify geometry or dimensions using the `pfcSolid.SolidActionListener.OnBeforeRegenerate` method call.
- A regeneration that did not take place because nothing was modified is identified as a regeneration failure.

The `pfcSolid.SolidActionListener.OnBeforeUnitConvert` and `pfcSolid.SolidActionListener.OnAfterUnitConvert` methods activate when a user modifies the unit scheme (by selecting the PTC Creo Parametric command **Set Up**, **Units**). The methods receive the `Solid` object to be converted and a Boolean flag that identifies whether the conversion changed the dimension values to keep the object the same size.

Note

SolidActionListeners can be registered with the session object so that its methods are called when these events occur for any solid model that is in session.

The pfcSolid.SolidActionListener.OnBeforeFeatureCreate method activates when the user starts to create a feature that requires the **Feature Creation** dialog box. Because this event occurs only after the dialog box is displayed, it will not occur at all for datums and other features that do not use this dialog box. This method takes two arguments: the solid model that will contain the feature and the ModelItem identifier.

The pfcSolid.SolidActionListener.OnAfterFeatureCreate method activates after any feature, including datums, has been created. This method takes the new Feature object as an argument.

The pfcSolid.SolidActionListener.OnAfterFeatureDelete method activates after any feature has been deleted. The method receives the solid that contained the feature and the (now defunct) ModelItem identifier.

Feature Level Action Listeners

Methods introduced:

- **pfcFeature.FeatureActionListener.OnBeforeDelete**
- **pfcFeature.FeatureActionListener.OnBeforeSuppress**
- **pfcFeature.FeatureActionListener.OnAfterSuppress**
- **pfcFeature.FeatureActionListener.OnBeforeRegen**
- **pfcFeature.FeatureActionListener.OnAfterRegen**
- **pfcFeature.FeatureActionListener.OnRegenFailure**
- **pfcFeature.FeatureActionListener.OnBeforeRedefine**
- **pfcFeature.FeatureActionListener.OnAfterCopy**
- **pfcFeature.FeatureActionListener.OnBeforeParameterDelete**

Each method in FeatureActionListener takes as an argument the feature that triggered the event.

FeatureActionListeners can be registered with the object so that the action listener's methods are called whenever these events occur for any feature that is in session or with a solid model to react to changes only in that model.

The method `pfcFeature.FeatureActionListener.OnBeforeDelete` is called before a feature is deleted.

The method `pfcFeature.FeatureActionListener.OnBeforeSuppress` is called before a feature is suppressed.

The method `pfcFeature.FeatureActionListener.OnAfterSuppress` is called after a successful feature suppression.

The method `pfcFeature.FeatureActionListener.OnBeforeRegen` is called before a feature is regenerated.

The method `pfcFeature.FeatureActionListener.OnAfterRegen` is called after a successful feature regeneration.

The method `pfcFeature.FeatureActionListener.OnRegenFailure` is called when a feature fails regeneration.

The method `pfcFeature.FeatureActionListener.OnBeforeRedefine` is called before a feature is redefined.

The method `pfcFeature.FeatureActionListener.OnAfterCopy` is called after a feature has been successfully copied.

The method `pfcFeature.FeatureActionListener.OnBeforeParameterDelete` is called before a feature parameter is deleted.

Cancelling an ActionListener Operation

J-Link allows you to cancel certain notification events, registered by the action listeners.

Methods Introduced:

- **pfcExceptions.XCancelProEAction.Throw**

The static method `pfcExceptions.XCancelProEAction.Throw` must be called from the body of an action listener to cancel the impending PTC Creo Parametric operation. This method will throw a J-Link exception signalling to PTC Creo Parametric to cancel the listener event.

Note: Your application should not catch the J-Link exception, or should rethrow it if caught, so that PTC Creo Parametric is forced to handle it.

The following events can be cancelled using this technique:

- `pfcSession.SessionActionListener.OnBeforeModelErase`
- `pfcSession.SessionActionListener.OnBeforeModelDelete`
- `pfcSession.SessionActionListener.OnBeforeModelRename`
- `pfcSession.SessionActionListener.OnBeforeModelSave`
- `pfcSession.SessionActionListener.OnBeforeModelPurge`
- `pfcSession.SessionActionListener.OnBeforeModelCopy`
- `pfcModel.ModelActionListener.OnBeforeParameterCreate`
- `pfcModel.ModelActionListener.OnBeforeParameterDelete`
- `pfcModel.ModelActionListener.OnBeforeParameterModify`
- `pfcFeature.FeatureActionListener.OnBeforeDelete`
- `pfcFeature.FeatureActionListener.OnBeforeSuppress`
- `pfcFeature.FeatureActionListener.OnBeforeParameterDelete`
- `pfcFeature.FeatureActionListener.OnBeforeParameterCreate`
- `pfcFeature.FeatureActionListener.OnBeforeRedefine`

23

Interface

This chapter describes various methods of importing and exporting files in J-Link.

Exporting Files and 2D Models

Method Introduced:

- **pfcModel.Model.Export**

The method `pfcModel.Model.Export` exports model data to a file. The exported files are placed in the current PTC Creo Parametric working directory. The input parameters are:

- *filename*—Output file name including extensions
- *exportdata*—The `pfcModel.ExportInstructions` object that controls the export operation. The type of data that is exported is given by the `pfcModel.ExportType` object.

There are four general categories of files to which you can export models:

- File types whose instructions inherit from `pfcModel.GeomExportInstructions`.

  These instructions export files that contain precise geometric information used by other CAD systems.
- File types whose instructions inherit from `pfcModel.CoordSysExportInstructions`.

  These instructions export files that contain coordinate information describing faceted, solid models (without datums and surfaces).
- File types whose instructions inherit from `pfcModel.FeatIdExportInstructions`.

  These instructions export information about a specific feature.
- General file types that inherit only from `pfcModel.ExportInstructions`.

  These instructions provide conversions to file types such as BOM (bill of materials).

For information on exporting to a specific format, see the J-Link APIWizard and online help for the PTC Creo Parametric interface.

Export Instructions

Methods Introduced:

- **pfcModel.pfcModel.RelationExportInstructions_Create**
- **pfcModel.pfcModel.ModelInfoExportInstructions_Create**
- **pfcModel.pfcModel.ProgramExportInstructions_Create**
- **pfcModel.pfcModel.IGESFileExportInstructions_Create**

- **pfcModel.pfcModel.DXFExportInstructions_Create**
- **pfcModel.pfcModel.RenderExportInstructions_Create**
- **pfcModel.pfcModel.STLASCIIExportInstructions_Create**
- **pfcModel.pfcModel.STLBinaryExportInstructions_Create**
- **pfcModel.pfcModel.BOMExportInstructions_Create**
- **pfcModel.pfcModel.DWGSetupExportInstructions_Create**
- **pfcModel.pfcModel.FeatInfoExportInstructions_Create**
- **pfcModel.pfcModel.MFGFeatCLExportInstructions_Create**
- **pfcModel.pfcModel.MFGOperCLExportInstructions_Create**
- **pfcModel.pfcModel.MaterialExportInstructions_Create**
- **pfcModel.pfcModel.CGMFILEExportInstructions_Create**
- **pfcModel.pfcModel.InventorExportInstructions_Create**
- **pfcModel.pfcModel.FIATExportInstructions_Create**
- **pfcModel.pfcModel.ConnectorParamExportInstructions_Create**
- **pfcModel.pfcModel.CableParamsFileInstructions_Create**
- **pfcModel.pfcModel.CATIAFacetsExportInstructions_Create**
- **pfcModel.pfcModel.VRMLModelExportInstructions_Create**
- **pfcModel.pfcModel.STEP2DExportInstructions_Create**
- **pfcModel.pfcModel.MedusaExportInstructions_Create**
- **pfcExport.pfcExport.CADDSExportInstructions_Create**
- **pfcModel.pfcModel.SliceExportData_Create**
- **pfcExport.pfcExport.NEUTRALFileExportInstructions_Create**
- **pfcExport.pfcExport.ProductViewExportInstructions_Create**
- **pfcSession.BaseSession.ExportDirectVRML**

Export Instructions Table

	Used to Export
`RelationExportInstructions`	A list of the relations and parameters in a part or assembly
`ModelInfoExportInstructions`	Information about a model, including units information, features, and children
`ProgramExportInstructions`	A program file for a part or assembly that can be edited to change the model
`IGESExportInstructions`	A drawing in `IGES` format
`DXFExportInstructions`	A drawing in `DXF` format
`RenderExportInstructions`	A part or assembly in `RENDER` format

	Used to Export
STLASCIIExportInstructions	A part or assembly to an ASCII STL file
STLBinaryExportInstructions	A part or assembly in a binary STL file
BOMExportInstructions	A BOM for an assembly
DWGSetupExportInstructions	A drawing setup file
FeatInfoExportInstructions	Information about one feature in a part or assembly
MfgFeatCLExportInstructions	A cutter location (CL) file for one NC sequence in a manufacturing assembly
MfgOperClExportInstructions	A cutter location (CL) file for all the NC sequences in a manufacturing assembly
MaterialExportInstructions	A material from a part
CGMFILEExportInstructions	A drawing in CGM format
InventorExportInstructions	A part or assembly in Inventor format
FIATExportInstructions	A part or assembly in FIAT format
ConnectorParamExportInstructions	The parameters of a connector to a text file
CableParamsFileInstructions	Cable parameters from an assembly
CATIAFacetsExportInstructions	A part or assembly in CATIA format (as a faceted model)
VRMLModelExportInstructions	A part or assembly in VRML format
STEP2DExportInstructions	A two-dimensional STEP format file
MedusaExportInstructions	A drawing in MEDUSA file
CADDSExportInstructions	A CADDS5 solid model
NEUTRALFileExportInstructions	A PTC Creo Parametric part to neutral format
ProductViewExportInstructions	A part, assembly, or drawing in PTC Creo View format
Export.SliceExportData	A slice export format

Note

The New Instruction Classes replace the following Deprecated Classes:

Deprecated Classes	New Instruction Classes
STEPExportInstructions	STEP3DExportInstructions
VDAExportInstructions	VDA3DExportInstructions
IGES3DExportInstructions	IGES3DNewExportInstructions

Exporting Drawing Sheets

The options required to export multiple sheets of a drawing are given by the `pfcModel.Export2DOption` object.

Methods Introduced:

- **pfcModel.pfcModel.Export2DOption_Create**
- **pfcModel.Export2DOption.SetExportSheetOption**
- **pfcModel.Export2DOption.SetModelSpaceSheet**
- **pfcModel.Export2DOption.SetSheets**

The method `pfcModel.pfcModel.Export2DOptions_Create` creates a new instance of the `pfcModel.Export2DOption` object. This object contains the following options:

- *ExportSheetOption*—Specifies the option for exporting multiple drawing sheets. Use the method `pfcModel.Export2DOption.SetExportSheetOption` to set the option for exporting multiple drawing sheets. The options are given by the `pfcModel.Export2DSheetOption` class and can be of the following types:
    - `EXPORT_CURRENT_TO_MODEL_SPACE`—Exports only the drawing's current sheet as model space to a single file. This is the default type.
    - `EXPORT_CURRENT_TO_PAPER_SPACE`—Exports only the drawing's current sheet as paper space to a single file. This type is the same as `EXPORT_CURRENT_TO_MODEL_SPACE` for formats that do not support the concept of model space and paper space.
    - `EXPORT_ALL`—Exports all the sheets in a drawing to a single file as paper space, if applicable for the format type.
    - `EXPORT_SELECTED`—Exports selected sheets in a drawing as paper space and one sheet as model space.
- *ModelSpaceSheet*—Specifies the sheet number that needs be exported as model space. This option is applicable only if the export formats support the concept of model space and paper space and if *ExportSheetOption* is set to `EXPORT_SELECTED`. Use the method `pfcModel.Export2DOption.SetModelSpaceSheet` to set this option.
- *Sheets*—Specifies the sheet numbers that need to be exported as paper space. This option is applicable only if *ExportSheetOption* is set to `EXPORT_SELECTED`. Use the method `pfcModel.Export2DOption.SetSheets` to set this option.

Exporting to Faceted Formats

The methods described in this section support the export of PTC Creo Parametric drawings and solid models to faceted formats like `CATIA CGR`.

Methods Introduced:

- **pfcExport.TriangulationInstructions.GetAngleControl**
- **pfcExport.TriangulationInstructions.SetAngleControl**
- **pfcExport.TriangulationInstructions.GetChordHeight**
- **pfcExport.TriangulationInstructions.SetChordHeight**
- **pfcExport.TriangulationInstructions.GetStepSize**
- **pfcExport.TriangulationInstructions.SetStepSize**
- **pfcExport.TriangulationInstructions.GetFacetControlOptions**
- **pfcExport.TriangulationInstructions.SetFacetControlOptions**

The methods `pfcExport.TriangulationInstructions.GetAngleControl` and `pfcExport.TriangulationInstructions.SetAngleControl` gets and sets the angle control for the exported facet drawings and models. You can set the value between 0.0 to 1.0.

Use the methods `pfcExport.TriangulationInstructions.GetChordHeight` and `pfcExport.TriangulationInstructions.SetChordHeight` to get and set the chord height for the exported facet drawings and models.

The methods `pfcExport.TriangulationInstructions.GetStepSize` and `pfcExport.TriangulationInstructions.SetStepSize` allow you to control the step size for the exported files. The default value is 0.0.

Note

You must pass the value of Step Size value as `NULL`, if you specify the Quality value.

The methods `pfcModel.CoordSysExportInstructions.GetStepSize` and `pfcModel.CoordSysExportInstructions.SetStepSize` control the step size for the exported files. The default value is 0.0.

Note

You must pass the value of Step Size value as `NULL`, if you specify the Quality value.

The methods `pfcExport.TriangulationInstructions.GetFacetControlOptions` and pfcExport.TriangulationInstructions.SetFacetControlOptions control the facet export options using bit flags. You can set the bit flags using the pfcModel.FacetControlFlag object. It has the following values:

- FACET_STEP_SIZE_ADJUST—Adjusts the step size according to the component size.
- FACET_CHORD_HEIGHT_ADJUST—Adjusts the chord height according to the component size.
- FACET_USE_CONFIG—If this flag is set, values of the flags FACET_STEP_SIZE_OFF, FACET_STEP_SIZE_ADJUST, and FACET_CHORD_HEIGHT_ADJUST are ignored and the configuration settings from the PTC Creo Parametric user interface are used during the export operation.
- FACET_CHORD_HEIGHT_DEFAULT—Uses the default value set in the PTC Creo Parametric user interface for the chord height.
- FACET_ANGLE_CONTROL_DEFAULT—Uses the default value set in the PTC Creo Parametric user interface for the angle control.
- FACET_STEP_SIZE_DEFAULT—Uses the default value set in the PTC Creo Parametric user interface for the step size.
- FACET_STEP_SIZE_OFF—Switches off the step size control.
- FACET_FORCE_INTO_RANGE—Forces the out-of-range parameters into range. If any of the FACET_*_DEFAULT option is set, then the option pfcFACET_FORCE_INTO_RANGE is not applied on that parameter.
- FACET_STEP_SIZE_FACET_INCLUDE_QUILTS—Includes quilts in the export of PTC Creo Parametric model to the specified format.
- EXPORT_INCLUDE_ANNOTATIONS—Includes annotations in the export of PTC Creo Parametric model to the specified format.

Note

To include annotations, during the export of PTC Creo Parametric model, you must call the method pfcModel.Model.Display before calling pfcModel.Model.Export.

Exporting Using Coordinate System

The methods described in this section support the export of files with information about the faceted solid models (without datums and surfaces). The files are exported in reference to the coordinate-system feature in the model being exported.

Methods Introduced:

- **pfcModel.CoordSysExportInstructions.GetCsysName**
- **pfcModel.CoordSysExportInstructions.SetCsysName**
- **pfcModel.CoordSysExportInstructions.GetQuality**
- **pfcModel.CoordSysExportInstructions.SetQuality**
- **pfcModel.CoordSysExportInstructions.GetMaxChordHeight**
- **pfcModel.CoordSysExportInstructions.SetMaxChordHeight**
- **pfcModel.CoordSysExportInstructions.GetAngleControl**
- **pfcModel.CoordSysExportInstructions.SetAngleControl**
- **pfcModel.CoordSysExportInstructions.GetSliceExportData**
- **pfcModel.CoordSysExportInstructions.SetSliceExportData**
- **pfcModel.CoordSysExportInstructions.GetStepSize**
- **pfcModel.CoordSysExportInstructions.SetStepSize**
- **pfcModel.CoordSysExportInstructions.GetFacetControlOptions**
- **pfcModel.CoordSysExportInstructions.SetFacetControlOptions**

The method `pfcModel.CoordSysExportInstructions.GetCsysName` returns the name of the the name of a coordinate system feature in the model being exported. It is recommended to use the coordinate system that places the part or assembly in its upper-right quadrant, so that all position and distance values of the exported assembly or part are positive. The method `pfcModel.CoordSysExportInstructions.SetCsysName` allows you to set the coordinate system feature name.

The methods `pfcModel.CoordSysExportInstructions.GetQuality` and `pfcModel.CoordSysExportInstructions.SetQuality` can be used instead of `pfcModel.CoordSysExportInstructions.GetMaxChordHeight` and `pfcModel.CoordSysExportInstructions.GetMaxChordHeight` and `pfcModel.CoordSysExportInstructions.GetAngleControl` and `pfcModel.CoordSysExportInstructions.SetAngleControl`.

You can set the value between 1 and 10. The higher the value you pass, the lower is the Maximum Chord Height setting and higher is the Angle Control setting the method uses. The default Quality value is 1.0.

Note

You must pass the value of Quality as `NULL`, if you use Maximum Chord Height and Angle Control values. If Quality, Maximum Chord Height, and Angle Control are all `NULL`, then the Quality setting of 3 is used.

Use the methods `pfcModel.CoordSysExportInstructions.GetMaxChordHeight` and `pfcModel.CoordSysExportInstructions.SetMaxChordHeight` to work with the maximum chord height for the exported files. The default value is 0.1.

Note

You must pass the value of Maximum Chord Height as `NULL`, if you specify the Quality value.

The methods `pfcModel.CoordSysExportInstructions.GetAngleControl` and `pfcModel.CoordSysExportInstructions.SetAngleControl` allow you to work with the angle control setting for the exported files. The default value is 0.1.

Note

You must pass the value of Angle Control value as `NULL`, if you specify the Quality value.

The methods `pfcModel.CoordSysExportInstructions.GetSliceExportData` and `pfcModel.CoordSysExportInstructions.SetSliceExportData` get and set the `pfcModel.SliceExportData` data object that specifies data for the slice export. The options in this object are described as follows:

- *CompIds*—Specifies the sequence of integers that identify the components that form the path from the root assembly down to the component part or assembly being referred to. Use the methods pfcModel.SliceExportData.GetCompIds and pfcModel.SliceExportData.SetCompIds to work with the component IDs.

The methods pfcModel.CoordSysExportInstructions.GetStepSize and pfcModel.CoordSysExportInstructions.SetStepSize control the step size for the exported files. The default value is 0.0.

Note

You must pass the value of Step Size value as NULL, if you specify the Quality value.

The methods pfcModel.CoordSysExportInstructions.GetFacetControlOptions and pfcModel.CoordSysExportInstructions.SetFacetControlOptions control the facet export options using bit flags. You can set the bit flags using the pfcModel.FacetControlFlag object. For more information on the bit flag values, please refer to the section Exporting to Faceted Formats on page 305.

Exporting to PDF and U3D

The methods described in this section support the export of PTC Creo Parametric drawings and solid models to Portable Document Format (PDF) and U3D format. You can export a drawing or a 2D model as a 2D raster image embedded in a PDF file. You can export PTC Creo Parametric solid models in the following ways:

- As a U3D model embedded in a one-page PDF file
- As 2D raster images embedded in the pages of a PDF file representing saved views
- As a standalone U3D file

While exporting multiple sheets of a PTC Creo Parametric drawing to a PDF file, you can choose to export all sheets, the current sheet, or selected sheets.

These methods also allow you to insert a variety of non-geometric information to improve document content, navigation, and search.

Methods Introduced:

- **pfcExport.pfcExport.PDFExportInstructions_Create**
- **pfcExport.PDFExportInstructions.GetFilePath**

- **pfcExport.PDFExportInstructions.SetFilePath**
- **pfcExport.PDFExportInstructions.GetOptions**
- **pfcExport.PDFExportInstructions.SetOptions**
- **pfcExport.PDFExportInstructions.GetProfilePath**
- **pfcExport.PDFExportInstructions.SetProfilePath**
- **pfcExport.pfcExport.PDFOption_Create**
- **pfcExport.PDFOption.SetOptionType**
- **pfcExport.PDFOption.SetOptionValue**

The method `pfcExport.pfcExport.PDFExportInstructions_Create` creates a new instance of the `pfcExport.PDFExportInstructions` data object that describes how to export PTC Creo Parametric drawings or solid models to the `PDF` and `U3D` formats. The options in this object are described as follows:

- *FilePath*—Specifies the name of the output file. Use the method `pfcExport.PDFExportInstructions.SetFilePath` to set the name of the output file.
- *Options*—Specifies a collection of PDF export options of the type `pfcExport.PDFOption`. Create a new instance of this object using the method `pfcExport.pfcExport.PDFOption_Create`. This object contains the following attributes:
  - *OptionType*—Specifies the type of option in terms of the `pfcExport.PDFOptionType` class. Set this option using the method `pfcExport.PDFOption.SetOptionType`.
  - *OptionValue*—Specifies the value of the option in terms of the `pfcArgument.ArgValue` object. Set this option using the method `pfcExport.PDFOption.SetOptionValue`.

  Use the method `pfcExport.PDFExportInstructions.SetOptions` to set the collection of PDF export options.
- *ProfilePath*—Specifies the export profile path. Use the method `pfcExport.PDFExportInstructions.SetProfilePath` to set the profile path. When you set the profile path, the PDF export options set in the data object `pfcExport.PDFExportInstructions` data object are ignored when the method `pfcModel.Model.Export` is called. You can set the profile path as `NULL`.

> **Note**
>
> You can specify the profile path only for drawings.

The types of options (given by the EpfcExport.PDFOptionType class) available for export to PDF and U3D formats are described as follows:

- PDFOPT_FONT_STROKE—Allows you to switch between using TrueType fonts or "stroking" text in the resulting document. This option is given by the pfcExport.PDFFontStrokeMode class and takes the following values:
  - PDF_USE_TRUE_TYPE_FONTS—Specifies TrueType fonts. This is the default type.
  - PDF_STROKE_ALL_FONTS—Specifies the option to stroke all fonts.
- PDFOPT_COLOR_DEPTH—Allows you to choose between color, grayscale, or monochrome output. This option is given by the pfcExport.PDFColorDepth class and takes the following values:
  - PDF_CD_COLOR—Specifies color output. This is the default value.
  - PDF_CD_GRAY—Specifies grayscale output.
  - PDF_CD_MONO—Specifies monochrome output.
- PDFOPT_HIDDENLINE_MODE—Enables you to set the style for hidden lines in the resulting PDF document. This option is given by the pfcExport.PDFHiddenLineMode class and takes the following values:
  - PDF_HLM_SOLID—Specifies solid hidden lines.
  - PDF_HLM_DASHED—Specifies dashed hidden lines. This is the default type.
- PDFOPT_SEARCHABLE_TEXT—If true, stroked text is searchable. The default value is true.
- PDFOPT_RASTER_DPI—Allows you to set the resolution for the output of any shaded views in DPI. It can take a value between 100 and 600. The default value is 300.
- PDFOPT_LAUNCH_VIEWER—If true, launches the Adobe Acrobat Reader. The default value is true.
- PDFOPT_LAYER_MODE—Enables you to set the availability of layers in the document. It is given by the pfcExport.PDFLayerMode class and takes the following values:
  - PDF_LAYERS_ALL—Exports the visible layers and entities. This is the default.

- PDF_LAYERS_VISIBLE—Exports only visible layers in a drawing.
- PDF_LAYERS_NONE—Exports only the visible entities in the drawing, but not the layers on which they are placed.

- PDFOPT_PARAM_MODE—Enables you to set the availability of model parameters as searchable metadata in the PDF document. It is given by the pfcExport.PDFParameterMode class and takes the following values:
  - PDF_PARAMS_ALL—Exports the drawing and the model parameters to PDF. This is the default.
  - PDF_PARAMS_DESIGNATED—Exports only the specified model parameters in the PDF metadata.
  - PDF_PARAMS_NONE—Exports the drawing to PDF without the model parameters.
- PDFOPT_HYPERLINKS—Sets hyperlinks to be exported as label text only or sets the underlying hyperlink URLs as active. The default value is true, specifying that the hyperlinks are active.
- PDFOPT_BOOKMARK_ZONES—If true, adds bookmarks to the PDF showing zoomed in regions or zones in the drawing sheet. The zone on an A4-size drawing sheet is ignored.
- PDFOPT_BOOKMARK_VIEWS—If true, adds bookmarks to the PDF document showing zoomed in views on the drawing.
- PDFOPT_BOOKMARK_SHEETS—If true, adds bookmarks to the PDF document showing each of the drawing sheets.
- PDFOPT_BOOKMARK_FLAG_NOTES—If true, adds bookmarks to the PDF document showing the text of the flag note.
- PDFOPT_TITLE—Specifies a title for the PDF document.
- PDFOPT_AUTHOR—Specifies the name of the person generating the PDF document.
- PDFOPT_SUBJECT—Specifies the subject of the PDF document.
- PDFOPT_KEYWORDS—Specifies relevant keywords in the PDF document.
- PDFOPT_PASSWORD_TO_OPEN—Sets a password to open the PDF document. By default, this option is NULL, which means anyone can open the PDF document without a password.
- PDFOPT_MASTER_PASSWORD—Sets a password to restrict or limit the operations that the viewer can perform on the opened PDF document. By default, this option is NULL, which means you can make any changes to the PDF document regardless of the settings of the modification flags PDFOPT_ALLOW_*.

- PDFOPT_RESTRICT_OPERATIONS—If true, enables you to restrict or limit operations on the PDF document. By default, is is false.
- PDFOPT_ALLOW_MODE—Enables you to set the security settings for the PDF document. This option must be set if PDFOPT_RESTRICT_OPERATIONS is set to true. It is given by the pfcExport.PDFRestrictOperationsMode class and takes the following values:
  - PDF_RESTRICT_NONE—Specifies that the user can perform any of the permitted viewer operations on the PDF document. This is the default value.
  - PDF_RESTRICT_FORMS_SIGNING—Restricts the user from adding digital signatures to the PDF document.
  - PDF_RESTRICT_INSERT_DELETE_ROTATE—Restricts the user from inserting, deleting, or rotating the pages in the PDF document.
  - PDF_RESTRICT_COMMENT_FORM_SIGNING—Restricts the user from adding or editing comments in the PDF document.
  - PDF_RESTRICT_EXTRACTING—Restricts the user from extracting pages from the PDF document.
- PDFOPT_ALLOW_PRINTING—If true, allows you to print the PDF document. By default, it is true.
- PDFOPT_ALLOW_PRINTING_MODE—Enables you to set the print resolution. It is given by the pfcExport.PDFPrintingMode class and takes the following values:
  - PDF_PRINTING_LOW_RES—Specifies low resolution for printing.
  - PDF_PRINTING_HIGH_RES—Specifies high resolution for printing. This is the default value.
- PDFOPT_ALLOW_COPYING—If true, allows you to copy content from the PDF document. By default, it is true.
- PDFOPT_ALLOW_ACCESSIBILITY—If true, enables visually-impaired screen reader devices to extract data independent of the value given by the pfcExport.PDFRestrictOperationsMode class. The default value is true.
- PDFOPT_PENTABLE—If true, uses the standard PTC Creo Parametric pentable to control the line weight, line style, and line color of the exported geometry. The default value is false.
- PDFOPT_LINECAP—Enables you to control the treatment of the ends of the geometry lines exported to PDF. It is given by the pfcExport.PDFLinecap class and takes the following values:

- PDF_LINECAP_BUTT—Specifies the butt cap square end. This is the default value.
- PDF_LINECAP_ROUND—Specifies the round cap end.
- PDF_LINECAP_PROJECTING_SQUARE—Specifies the projecting square cap end.

- PDFOPT_LINEJOIN—Enables you to control the treatment of the joined corners of connected lines exported to PDF. It is given by the pfcExport.PDFLinejoin class and takes the following values:
  - PDF_LINEJOIN_MITER—Specifies the miter join. This is the default.
  - PDF_LINEJOIN_ROUND—Specifies the round join.
  - PDF_LINEJOIN_BEVEL—Specifies the bevel join.
- PDFOPT_SHEETS—Allows you to specify the sheets from a PTC Creo Parametric drawing that are to be exported to PDF. It is given by the pfcExport.PrintSheets enumerated class and takes the following values:
  - PRINT_CURRENT_SHEET—Only the current sheet is exported to PDF
  - PRINT_ALL_SHEETS—All the sheets are exported to PDF. This is the default value.
  - PRINT_SELECTED_SHEETS—Sheets of a specified range are exported to PDF. If this value is assigned, then the value of the option PDFOPT_SHEET_RANGE must also be known.
- PDFOPT_SHEET_RANGE—Specifies the range of sheets in a drawing that are to be exported to PDF. If this option is set, then the option PDFOPT_SHEETS must be set to the value PRINT_SELECTED_SHEETS.
- PDFOPT_EXPORT_MODE—Enables you to select the object to be exported to PDF and the export format. It is given by the pfcExport.PDFExportMode class and takes the following values:
  - PDF_2D_DRAWING—Only drawings are exported to PDF. This is the default value.
  - PDF_3D_AS_NAMED_VIEWS—3D models are exported as 2D raster images embedded in PDF files.
  - PDF_3D_AS_U3D_PDF—3D models are exported as U3D models embedded in one-page PDF files.
  - PDF_3D_AS_U3D—A 3D model is exported as a U3D (.u3d) file. This value ignores the options set for the pfcExport.PDFOptionType class.

- PDFOPT_LIGHT_DEFAULT—Enables you to set the default lighting style used while exporting 3D models in the U3D format to a one-page PDF file, that is when the option PDFOPT_EXPORT_MODE is set to PDF_3D_AS_U3D. The values for this option are given by the pfcExport.PDFU3DLightingMode class.
- PDFOPT_RENDER_STYLE_DEFAULT—Enables you to set the default rendering style used while exporting PTC Creo Parametric models in the U3D format to a one-page PDF file, that is when the option PDFOPT_EXPORT_MODE is set to PDF_3D_AS_U3D. The values for this option are given by the pfcModel.PDFU3DRenderMode class.
- PDFOPT_SIZE—Allows you to specify the page size of the exported PDF file. The values for this option are given by the pfcExport.PlotPaperSize class. If the value is set to VARIABLESIZEPLOT, you also need to set the options PDFOPT_HEIGHT and PDFOPT_WIDTH.
- PDFOPT_HEIGHT—Enables you to set the height for a user-defined page size of the exported PDF file. The default value is 0.0.
- PDFOPT_WIDTH—Enables you to set the width for a user-defined page size of the exported PDF file. The default value is 0.0.
- PDFOPT_ORIENTATION—Enables you to specify the orientation of the pages in the exported PDF file. It is given by the pfcSheet.SheetOrientation class.
  - ORIENT_PORTRAIT—Exports the pages in portrait orientation. This is the default value.
  - ORIENT_LANDSCAPE—Exports the pages in landscape orientation.
- PDFOPT_TOP_MARGIN—Allows you to specify the top margin of the view port. The default value is 0.0.
- PDFOPT_LEFT_MARGIN—Allows you to specify the left margin of the view port. The default value is 0.0.
- PDFOPT_BACKGROUND_COLOR_RED—Specifies the default red background color that appears behind the U3D model. You can set any value within the range of 0.0 to 1.0. The default value is 1.0.
- PDFOPT_BACKGROUND_COLOR_GREEN—Specifies the default green background color that appears behind the U3D model. You can set any value within the range of 0.0 to 1.0. The default value is 1.0.
- PDFOPT_BACKGROUND_COLOR_BLUE—Specifies the default blue background color that appears behind the U3D model. You can set any value within the range of 0.0 to 1.0. The default value is 1.0.

- PDFOPT_ADD_VIEWS—If true, allows you to add view definitions to the U3D model from a file. By default, it is true.
- PDFOPT_VIEW_TO_EXPORT—Specifies the view or views to be exported to the PDF file. It is given by the pfcExport.PDFSelectedViewMode class and takes the following values:
  - PDF_VIEW_SELECT_CURRENT—Exports the current graphical area to a one-page PDF file.
  - PDF_VIEW_SELECT_ALL—Exports all the views to a multi-page PDF file. Each page contains one view with the view name displayed at the bottom center of the view port.
  - PDF_VIEW_SELECT_BY_NAME—Exports the selected view to a one-page PDF file with the view name printed at the bottom center of the view port. If this value is assigned, then the option PDFOPT_SELECTED_VIEW must also be set.
- PDFOPT_SELECTED_VIEW—Sets the option PDFOPT_VIEW_TO_EXPORT to the value PDF_VIEW_SELECT_BY_NAME, if the corresponding view is successfully found.
- PDFOPT_PDF_SAVE—Specifies the PDF save options. It is given by the pfcExport.PDFSaveMode class and takes the following values:
  - PDF_ARCHIVE_1—Applicable only for the value PDF_2D_DRAWING. Saves the drawings as PDF with the following conditions:
    - The value of pfcExport.PDFLayerMode is set to PDF_LAYERS_NONE.
    - The value of PDFOPT_HYPERLINKS is set to FALSE.
    - The shaded views in the drawings will not have transparency and may overlap other data in the PDF.
    - The value of PDFOPT_PASSWORD_TO_OPEN is set to NULL.
    - The value of PDFOPT_MASTER_PASSWORD is set to NULL.
  - PDF_FULL—Saves the PDF with the values set by you. This is the default value.

Exporting 3D Geometry

J-Link allows you to export three dimensional geometry to various formats. Pass the instructions object containing information about the desired export file to the method pfcModel.Model.Export.

Export Instructions

Methods Introduced:

- **pfcExport.Export3DInstructions.GetConfiguration**
- **pfcExport.Export3DInstructions.SetConfiguration**
- **pfcExport.Export3DInstructions.GetReferenceSystem**
- **pfcExport.Export3DInstructions.SetReferenceSystem**
- **pfcExport.Export3DInstructions.GetGeometry**
- **pfcExport.Export3DInstructions.SetGeometry**
- **pfcExport.Export3DInstructions.GetIncludedEntities**
- **pfcExport.Export3DInstructions.SetIncludedEntities**
- **pfcExport.Export3DInstructions.GetLayerOptions**
- **pfcExport.Export3DInstructions.SetLayerOptions**
- **pfcExport.pfcExport.GeometryFlags_Create**
- **pfcExport.pfcExport.InclusionFlags_Create**
- **pfcExport.pfcExport.LayerExportOptions_Create**
- **pfcExport.pfcExport.STEP3DExportInstructions_Create**
- **pfcExport.pfcExport.VDA3DExportInstructions_Create**
- **pfcExport.pfcExport.IGES3DNewExportInstructions_Create**
- **pfcExport.pfcExport.CATIAModel3DExportInstructions_Create**
- **pfcExport.pfcExport.ACIS3DExportInstructions_Create**
- **pfcExport.pfcExport.CatiaPart3DExportInstructions_Create**
- **pfcExport.pfcExport.CatiaProduct3DExportInstructions_Create**
- **pfcExport.pfcExport.CatiaCGR3DExportInstructions_Create**
- **pfcExport.pfcExport.DXF3DExportInstructions_Create**
- **pfcExport.pfcExport.DWG3DExportInstructions_Create**
- **pfcExport.pfcExport.JT3DExportInstructions_Create**
- **pfcExport.pfcExport.ParaSolid3DExportInstructions_Create**
- **pfcExport.pfcExport.UG3DExportInstructions_Create**
- **pfcExport.pfcExport.TriangulationInstructions_Create**

The `pfcExport.Export3DInstructions` contains data to export a part or an assembly to a specifed 3D format. The fields of this are:

- `AssemblyConfiguration`—While exporting an assembly you can specify the structure and contents of the output files. The options are:

- EXPORT_ASM_FLAT_FILE—Exports all the geometry of the assembly to a single file as if it were a part.
- EXPORT_ASM_SINGLE_FILE—Exports an assembly structure to a file with external references to component files. This file contains only top-level geometry.
- EXPORT_ASM_MULTI_FILE—Exports an assembly structure to a single file and the components to component files. It creates component parts and subassemblies with their respective geometry and external references. This option supports all levels of hierarchy.
- EXPORT_ASM_ASSEMBLY_FILE—Exports an assembly as multiple files containing geometry information of its components and assembly features.

- CoordSystem—The reference coordinate system used for export. If this value is null, the system uses the default coordinate system.
- GeometryFlags—The object describing the type of geometry to export. The pfcExport.pfcExport.GeometryFlags_Create returns this instruction object. The types of geometry supported by the export operation are:
  - Wireframe—Export edges only.
  - Solid—Export surfaces along with topology.
  - Surfaces—Export all model surfaces.
  - Quilts—Export as quilt.
- InclusionFlags—The object returned by the method pfcExport.pfcExport.InclusionFlags_Create that determines whether to include certain entities. The entities are:
  - Datums—Determines whether datum curves are included when exporting files. If true the datum curve information is included during export. The default value is false.
  - Blanked—Determines whether entities on blanked layers are exported. If true entities on blanked layers are exported. The default value is false.
- LayerExportOptions—The instructions object returned by the method pfcExport.pfcExport.LayerExportOptions_Create that describes how to export layers. To export layers you can specify the following:
  - *UseAutoId*—Enables you to set or remove an interface layer ID. A layer is recognized with this ID when exporting the file to a specified output format. If true, automatically assigns interface IDs to layers not assigned IDs and exports them. The default value is false.

- *LayerSetupFile*—Specifies the name and complete path of the layer setup file. This file contains the layer assignment information which includes the name of the layer, its display status, the interface ID and number of sub layers.

The method `pfcExport.pfcExport.TriangulationInstructions_Create` creates a object that will be used to define the parameters for faceted exports.

Export 3D Instructions Table

	Used to Export
`STEP3DExportInstructions`	A part or assembly in `STEP` format
`VDA3DExportInstructions`	A part or assembly in `VDA` format
`IGES3DNewExportInstructions`	A part or assembly in `IGES` format
`CATIAModel3DExportInstructions`	A part or assembly in `CATIA MODEL` format
`ACIS3DExportInstructions`	A part or assembly in `ACIS` format
`CatiaPart3DExportInstructions`	A part or assembly in `CATIA PART` format
`CatiaProduct3DExportInstructions`	A part or assembly in `CATIA PRODUCT` format
`CatiaCGR3DExportInstructions`	A part or assembly in `CATIA CGR` format
`JT3DExportInstructions`	A part or assembly in `JT` format
`ParaSolid3DExportInstructions`	A part or assembly in `PARASOLID` format
`UG3DExportInstructions`	A part or assembly in `UG` format
`DWG3DExportInstructions`	A part or assembly in `DWG` format
`DXF3DExportInstructions`	A part or assembly in `DXF` format
`TriangulationInstructions`	A part or assembly in faceted format

Export Utilities

Methods Introduced:

- **pfcSession.BaseSession.IsConfigurationSupported**
- **pfcSession.BaseSession.IsGeometryRepSupported**

The method `pfcSession.BaseSession.IsConfigurationSupported` checks whether the specified assembly configuration is valid for a particular model and the specified export format. The input parameters for this method are:

- *Configuration*—Specifies the structure and content of the output files.
- *Type*—Specifies the output file type to create.

The method returns a true value if the configuration is supported for the specified export type.

The method pfcSession.BaseSession.IsGeometryRepSupported checks whether the specified geometric representation is valid for a particular export format. The input parameters are :

- *Flags*—The type of geometry supported by the export operation.
- *Type*—The output file type to create.

The method returns a true value if the geometry combination is valid for the specified model and export type.

The methods pfcSession.BaseSession.IsConfigurationSupported and pfcSession.BaseSession.IsGeometryRepSupported must be called before exporting an assembly to the specified export formats except for the CADDS and STEP2D formats. The return values of both the methods must be true for the export operation to be successful.

Use the method Model.Model.Export to export the assembly to the specified output format.

Shrinkwrap Export

To improve performance in a large assembly design, you can export lightweight representations of models called shrinkwrap models. A shrinkwrap model is based on the external surfaces of the source part or asssembly model and captures the outer shape of the source model.

You can create the following types of nonassociative exported shrinkwrap models:

- Surface Subset—This type consists of a subset of the original model's surfaces.
- Faceted Solid—This type is a faceted solid representing the original solid.
- Merged Solid—The external components from the reference assembly model are merged into a single part representing the solid geometry in all collected components.

Methods Introduced:

- **pfcSolid.Solid.ExportShrinkwrap**

You can export the specified solid model as a shrinkwrap model using the method pfcSolid.Solid.ExportShrinkwrap. This method takes the ShrinkwrapExportInstructions object as an argument.

Use the appropriate given in the following table to create the required type of shrinkwrap. All the have their own static method to create an object of the specified type. The object created by these interfaces can be used as an object of type ShrinkwrapExportInstructions or ShrinkwrapModelExportInstructions.

Type of Shrinkwrap Model	to Use
Surface Subset	`ShrinkwrapSurfaceSubset Instructions`
Faceted Part	`ShrinkwrapFacetedPart Instructions`
Faceted VRML	`ShrinkwrapFacetedVRML Instructions`
Faceted STL	`ShrinkwrapFacetedSTLInstructions`
Merged Solid	`ShrinkwrapMergedSolidInstructions`

Setting Shrinkwrap Options

The `ShrinkwrapModelExportInstructions` contains the general methods available for all the types of shrinkwrap models. The object created by any of the interfaces specified in the preceeding table can be used with these methods.

MethodsIntroduced:

- **pfcShrinkwrap.ShrinkwrapModelExportInstructions.GetMethod**
- **pfcShrinkwrap.ShrinkwrapModelExportInstructions.GetQuality**
- **pfcShrinkwrap.ShrinkwrapModelExportInstructions.SetQuality**
- **pfcShrinkwrap.ShrinkwrapModelExportInstructions.GetAutoHoleFilling**
- **pfcShrinkwrap.ShrinkwrapModelExportInstructions.SetAutoHoleFilling**
- **pfcShrinkwrap.ShrinkwrapModelExportInstructions.GetIgnoreSkeleton**
- **pfcShrinkwrap.ShrinkwrapModelExportInstructions.SetIgnoreSkeleton**
- **pfcShrinkwrap.ShrinkwrapModelExportInstructions.GetIgnoreQuilts**
- **pfcShrinkwrap.ShrinkwrapModelExportInstructions.SetIgnoreQuilts**
- **pfcShrinkwrap.ShrinkwrapModelExportInstructions.GetAssignMassProperties**
- **pfcShrinkwrap.ShrinkwrapModelExportInstructions.SetAssignMassProperties**
- **pfcShrinkwrap.ShrinkwrapModelExportInstructions.GetIgnoreSmallSurfaces**
- **pfcShrinkwrap.ShrinkwrapModelExportInstructions.SetIgnoreSmallSurfaces**
- **pfcShrinkwrap.ShrinkwrapModelExportInstructions.GetSmallSurfPercentage**
- **pfcShrinkwrap.ShrinkwrapModelExportInstructions.SetSmallSurfPercentage**

- **pfcShrinkwrap.ShrinkwrapModelExportInstructions.GetDatumReferences**
- **pfcShrinkwrap.ShrinkwrapModelExportInstructions.SetDatumReferences**

The method `pfcShrinkwrap.ShrinkwrapModelExportInstructions.GetMethod` returns the method used to create the shrinkwrap. The types of shrinkwrap methods are:

- `SWCREATE_SURF_SUBSET`—Surface Subset
- `SWCREATE_FACETED_SOLID`—Faceted Solid
- `SWCREATE_MERGED_SOLID`—Merged Solid

The method `pfcShrinkwrap.ShrinkwrapModelExportInstructions.GetQuality` specifies the quality level for the system to use when identifying surfaces or components that contribute to the shrinkwrap model. Quality ranges from 1 which produces the coarsest representation of the model in the fastest time, to 10 which produces the most exact representation. Use the method `pfcShrinkwrap.ShrinkwrapModelExportInstructions.SetQuality` to set the quality level for the system during the shrinkwrap export. The default value is 1.

The method `pfcShrinkwrap.ShrinkwrapModelExportInstructions.GetAutoHoleFilling` returns true if auto hole filling is enabled during Shrinkwrap export. The method `pfcShrinkwrap.ShrinkwrapModelExportInstructions.SetAutoHoleFilling` sets a flag that forces PTC Creo Parametric to identify all holes and surfaces that intersect a single surface and fills those holes during shrinkwrap. The default value is true.

The methods `pfcShrinkwrap.ShrinkwrapModelExportInstructions.GetIgnoreSkeleton` and `pfcShrinkwrap.ShrinkwrapModelExportInstructions.SetIgnoreSkeleton` determine whether the skeleton model geometry must be included in the shrinkwrap model.

The methods `pfcShrinkwrap.ShrinkwrapModelExportInstructions.GetIgnoreQuilts` and `pfcShrinkwrap.ShrinkwrapModelExportInstructions.SetIgnoreQuilts` determine whether external quilts must be included in the shrinkwrap model.

The method `pfcShrinkwrap.ShrinkwrapModelExportInstructions.GetAssignMassProperties` determines the mass property of the model. The method `pfcShrinkwrap.ShrinkwrapModelExportInstructions.SetAssignMassProperties` assign mass properties to the shrinkwrap model. The default value is false and the mass properties of the original model is assigned to the shrinkwrap model. If the value is set to true, the user must assign a value for the mass properties.

The method `pfcShrinkwrap.ShrinkwrapModelExportInstructions.GetIgnoreSmallSurfaces` specifies whether small surfaces are ignored during the creation of a shrinkwrap model. The method `pfcShrinkwrap.ShrinkwrapModelExportInstructions.SetIgnoreSmallSurfaces` sets a flag that forces PTC Creo Parametric to skip surfaces smaller than a certain size. The default value is false. The size of the surface is specified as a percentage of the model's size. This size can be modified using the methods `pfcShrinkwrap.ShrinkwrapModelExportInstructions.GetSmallSurfPercentage` and `pfcShrinkwrap.ShrinkwrapModelExportInstructions.SetSmallSurfPercentage`.

The methodproperty `pfcShrinkwrap.ShrinkwrapModelExportInstructions.GetDatumReferences` and `pfcShrinkwrap.ShrinkwrapModelExportInstructions.SetDatumReferences` specify and select the datum planes, points, curves, axes, and coordinate system references to be included in the shrinkwrap model.

Surface Subset Options

Methods Introduced:

- **pfcShrinkwrap.pfcShrinkwrap.ShrinkwrapSurfaceSubsetInstructions_Create**
- **pfcShrinkwrap.ShrinkwrapSurfaceSubsetInstructions.GetAdditionalSurfaces**
- **pfcShrinkwrap.ShrinkwrapSurfaceSubsetInstructions.SetAdditionalSurfaces**
- **pfcShrinkwrap.ShrinkwrapSurfaceSubsetInstructions.GetOutputModel**
- **pfcShrinkwrap.ShrinkwrapSurfaceSubsetInstructions.SetOutputModel**

The static method
pfcShrinkwrap.Shrinkwrap.ShrinkwrapSurfaceSubsetInstructions
_Create returns an object used to create a shrinkwrap model of surface subset type. Specify the name of the output model in which the shrinkwrap is to be created as an input to this method.

The method
pfcShrinkwrap.ShrinkwrapSurfaceSubsetInstructions.GetAdditionalSurfaces specifies the surfaces included in the shrinkwrap model while the method
pfcShrinkwrap.ShrinkwrapSurfaceSubsetInstructions.SetAdditionalSurfaces selects individual surfaces to be included in the shrinkwrap model.

The method
pfcShrinkwrap.ShrinkwrapSurfaceSubsetInstructions.GetOutputModel returns the template model where the shrinkwrap geometry is to be created while the method
pfcShrinkwrap.ShrinkwrapSurfaceSubsetInstructions.SetOutputModel sets the template model.

Faceted Solid Options

The ShrinkwrapFacetedFormatInstructions consists of the following types:

- SWFACETED_PART—PTC Creo Parametric part with normal geometry. This is the default format type.
- SWFACETED_STL—An STL file.
- SWFACETED_VRML—A VRML file.

Use the Create method to create the object of the specified type. Upcast the object to use the general methods available in this .

Methods Intoduced:

- **pfcShrinkwrap.ShrinkwrapFacetedFormatInstructions.GetFormat**
- **pfcShrinkwrap.ShrinkwrapFacetedFormatInstructions.GetFramesFile**
- **pfcShrinkwrap.ShrinkwrapFacetedFormatInstructions.SetFramesFile**

The method
pfcShrinkwrap.ShrinkwrapFacetedFormatInstructions.GetFormat returns the the output file format of the shrinkwrap model.

The methods
pfcShrinkwrap.ShrinkwrapFacetedFormatInstructions.GetFramesFile and

pfcShrinkwrap.ShrinkwrapFacetedFormatInstructions.SetFramesFile enable you to select a frame file to create a faceted solid motion envelope model that represents the full motion of the mechanism captured in the frame file. Specify the name and complete path of the frame file.

Faceted Part Options

Methods Introduced:

- **pfcShrinkwrap.pfcShrinkwrap.ShrinkwrapFacetedPartInstructions_Create**
- **pfcShrinkwrap.ShrinkwrapFacetedPartInstructions.GetLightweight**
- **pfcShrinkwrap.ShrinkwrapFacetedPartInstructions.SetLightweight**

The static method pfcShrinkwrap.Shrinkwrap.ShrinkwrapFacetedPartInstructions_Create returns an object used to create a shrinkwrap model of shrinkwrap faceted type. The input parameters of this method are:

- *OutputModel*—Specify the output model where the shrinkwrap must be created.
- *Lightweight*—Specify this value as True if the shrinkwrap model is a Lightweight PTC Creo Parametric part.

The method pfcShrinkwrap.ShrinkwrapFacetedPartInstructions.GetLightweight returns a true value if the output file format of the shrinkwrap model is a Lightweight PTC Creo Parametric part. The method pfcShrinkwrap.ShrinkwrapFacetedPartInstructions.SetLightweight specifies if the PTC Creo Parametric part is exported as a light weight faceted geometry.

VRML Export Options

Methods Introduced:

- **pfcShrinkwrap.pfcShrinkwrap.ShrinkwrapVRMLInstructions_Create**
- **pfcShrinkwrap.ShrinkwrapVRMLInstructions.GetOutputFile**
- **pfcShrinkwrap.ShrinkwrapVRMLInstructions.SetOutputFile**

The static method pfcShrinkwrap.Shrinkwrap.ShrinkwrapVRMLInstructions_Create returns an object used to create a shrinkwrap model of shrinkwrap VRML format. Specify the name of the output model as an input to this method.

The method `pfcShrinkwrap.ShrinkwrapVRMLInstructions.GetOutputFile` returns the name of the output file to be created and the method `pfcShrinkwrap.ShrinkwrapVRMLInstructions.SetOutputFile` specifies the name of the output file to be created.

STL Export Options

Methods Introduced:

- **pfcShrinkwrap.pfcShrinkwrap.ShrinkwrapVRMLInstructions_Create**
- **pfcShrinkwrap.ShrinkwrapVRMLInstructions.GetOutputFile**
- **pfcShrinkwrap.ShrinkwrapVRMLInstructions.SetOutputFile**

The static method `pfcShrinkwrap.Shrinkwrap.ShrinkwrapVRMLInstructions_Create` returns an object used to create a shrinkwrap model of shrinkwrap `STL` format. Specify the name of the output model as an input to this method.

The method `pfcShrinkwrap.ShrinkwrapSTLInstructions.GetOutputFile` returns the name of the output file to be created and the method `pfcShrinkwrap.ShrinkwrapSTLInstructions.SetOutputFile` specifies the name of the output file to be created.

Merged Solid Options

Methods Introduced:

- **pfcShrinkwrap.pfcShrinkwrap.ShrinkwrapMergedSolidInstructions_Create**
- **pfcShrinkwrap.ShrinkwrapMergedSolidInstructions.GetAdditionalComponents**
- **pfcShrinkwrap.ShrinkwrapMergedSolidInstructions.SetAdditionalComponents**

The static method `pfcShrinkwrap.Shrinkwrap.ShrinkwrapMergedSolidInstructions_Create` returns an object used to create a shrinkwrap model of merged solids format. Specify the name of the output model as an input to this method.

The methods `pfcShrinkwrap.ShrinkwrapMergedSolidInstructions.GetAdditional`

Components specifies individual components of the assembly to be merged into the shrinkwrap model. Use the method pfcShrinkwrap.ShrinkwrapMergedSolidInstructions.SetAdditional
Components to select individual components of the assembly to be merged into the shrinkwrap model.

Importing Files

Method Introduced:

- **pfcModel.Model.Import**

The method pfcModel.Model.Import reads a file into PTC Creo Parametric. The format must be the same as it would be if these files were created by PTC Creo Parametric. The parameters are:

- *FilePath*—Absolute path of the file to be imported along with its extension.
- *ImportData*—The ImportInstructions object that controls the import operation.

Import Instructions

Methods Introduced:

- **pfcModel.pfcModel.RelationImportInstructions_Create**
- **pfcModel.pfcModel.IGESSectionImportInstructions_Create**
- **pfcModel.pfcModel.ProgramImportInstructions_Create**
- **pfcModel.pfcModel.ConfigImportInstructions_Create**
- **pfcModel.pfcModel.DWGSetupImportInstructions_Create**
- **pfcModel.pfcModel.SpoolImportInstructions_Create**
- **pfcModel.pfcModel.ConnectorParamsImportInstructions_Create**
- **pfcModel.pfcModel.ASSEMTreeCFGImportInstructions_Create**
- **pfcModel.pfcModel.WireListImportInstructions_Create**
- **pfcModel.pfcModel.CableParamsImportInstructions_Create**
- **pfcModel.pfcModel.STEPImport2DInstructions_Create**
- **pfcModel.pfcModel.IGESImport2DInstructions_Create**
- **pfcModel.pfcModel.DXFImport2DInstructions_Create**
- **pfcModel.pfcModel.DWGImport2DInstructions_Create**

The methods described in this section create an instructions data object to import a file of a specified type into PTC Creo Parametric. The details are as shown in the table below:

	Used to Import
`RelationImportInstructions`	A list of relations and parameters in a part or assembly.
`IGESSectionImportInstructions`	A section model in `IGES` format.
`ProgramImportInstructions`	A program file for a part or assembly that can be edited to change the model.
`ConfigImportInstructions`	Configuration instructions.
`DWGSetupImportInstructions`	A drawing s/u file.
`SpoolImportInstructions`	Spool instructions.
`ConnectorParamsImportInstructions`	Connector parameter instructions.
`ASSEMTreeCFGImportInstructions`	Assembly tree `CFG` instructions.
`WireListImportInstructions`	Wirelist instructions.
`CableParamsImportInstructions`	Cable parameters from an assembly.
`STEPImport2DInstructions`	A part or assembly in `STEP` format.
`IGESImport2DInstructions`	A part or assembly in `IGES` format.
`DXFImport2DInstructions`	A drawing in `DXF` format.
`DWGImport2DInstructions`	A drawing in `DWG` format.

Note

- The method `pfcModel.Model.Import` does not support importing of `CADAM` type of files.
- If a model or the file type `STEP`, `IGES`, `DWX`, or `SET` already exists, the imported model is appended to the current model. For more information on methods that return models of the types `STEP`, `IGES`, `DWX`, and `SET`, refer to Getting a Model Object on page 110.

Importing 2D Models

Method Introduced:

- **pfcSession.BaseSession.Import2DModel**

The method `pfcSession.BaseSession.Import2DModel` imports a two dimensional model based on the following parameters:

- *NewModelName*—Specifies the name of the new model.
- *Type*—Specifies the type of the model. The type can be one of the following:
  - `STEP`

- IGES
- DXF
- DWG
- SET

- *FilePath*—Specifies the location of the file to be imported along with the file extension
- *Instructions*—Specifies the `pfcModel.Import2DInstructions` object that controls the import operation.

  The `pfcModel.Import2DInstructions` contains the following attributes:

  - *Import2DViews*—Defines whether to import 2D drawing views.
  - *ScaleToFit*—If the current model has a different sheet size than that specified by the imported file, set the parameter to true to retain the current sheet size. Set the parameter to false to retain the sheet size of the imported file.
  - *FitToLeftCorner*—If this parameter is set to true, the bottom left corner of the imported file is adjusted to the bottom left corner of the current model. If it is set to false, the size of imported file is retained.

Note

The method `pfcSession.BaseSession.Import2DModel` does not support importing of `CADAM` type of files.

Importing 3D Geometry

Methods Introduced:

- **pfcSession.BaseSession.GetImportSourceType**
- **pfcSession.BaseSession.ImportNewModel**
- **pfcImport.LayerImportFilter.OnLayerImport**

For some input formats, the method `pfcSession.BaseSession.GetImportSourceType` returns the type of model that can be imported using a designated file. The input parameters of this method are:

- *FileToImport*—Specifies the path of the file along with its name and extension.
- *NewModelImportType*—Specifies the type of model to be imported.

The method `pfcSession.BaseSession.ImportNewModel` is used to import an external 3D format file and creates a new model or set of models of type `pfcModel.Model`. The input parameters of this method are:

- *FileToImport*—Specifies the path to the file along with its name and extension
- `pfcNewModelImportType`—Specifies the type of model to be imported. The types of models that can be imported are as follows:
  - `IMPORT_NEW_IGES`
  - `IMPORT_NEW_VDA`
  - `IMPORT_NEW_NEUTRAL`
  - `IMPORT_NEW_CADDS`
  - `IMPORT_NEW_STEP`
  - `IMPORT_NEW_STL`
  - `IMPORT_NEW_VRML`
  - `IMPORT_NEW_POLTXT`
  - `IMPORT_NEW_CATIA_SESSION`
  - `IMPORT_NEW_CATIA_MODEL`
  - `IMPORT_NEW_DXF`
  - `IMPORT_NEW_ACIS`
  - `IMPORT_NEW_PARASOLID`
  - `IMPORT_NEW_ICEM`
  - `IMPORT_NEW_DESKTOP`
  - `IMPORT_NEW_CATIA_PART`
  - `IMPORT_NEW_CATIA_PRODUCT`
  - `IMPORT_NEW_UG`
  - `IMPORT_NEW_PRODUCTVIEW`
  - `IMPORT_NEW_CATIA_CGR`
  - `IMPORT_NEW_JT`
  - `IMPORT_NEW_SW_PART`
  - `IMPORT_NEW_SW_ASSEM`
  - `pfcIMPORT_NEW_INVENTOR_PART`
  - `pfcIMPORT_NEW_INVENTOR_ASSEM`

- ModelType—Specifies the type of the model. It can be a part, assembly or drawing.
- *NewModelName*—Specifies a name for the imported model.
- LayerImportFilter—Specifies the layer filter. This parameter is optional.

The interface pfcImport.LayerImportFilter has a call back function pfcImport.LayerImportFilter.OnLayerImport. PTC Creo Parametric passes the object pfcImport.ImportedLayer describing each imported layer to the layer filter to allow you to perform changes on each layer as it is imported.

The method pfcExceptions.XCancelProEAction.Throw can be called from the body of the method pfcImport.LayerImportFilter.OnLayerImport to end the filtering of the layers.

Modifying the Imported Layers

Layers help you organize model items so that you can perform operations on those items collectively. These operations primarily include ways of showing the items in the model, such as displaying or blanking, selecting, and suppressing. The methods described in this section modify the attributes of the imported layers.

Methods Introduced:

- **pfcImport.ImportedLayer.GetName**
- **pfcImport.ImportedLayer.SetNewName**
- **pfcImport.ImportedLayer.GetSurfaceCount**
- **pfcImport.ImportedLayer.GetCurveCount**
- **pfcImport.ImportedLayer.GetTrimmedSurfaceCount**
- **pfcImport.ImportedLayer.SetAction**

Layers are identified by their names. The method pfcImport.ImportedLayer.GetName returns the name of the layer while the method pfcImport.ImportedLayer.SetNewName can be used to set the name of the layer. The name can be numeric or alphanumeric.

The method pfcImport.ImportedLayer.GetSurfaceCount returns the number of curves on the layer.

The method pfcImport.ImportedLayer.GetTrimmedSurfaceCount returns the number of trimmed surfaces on the layer and the method pfcImport.ImportedLayer.GetCurveCount returns the number of curves on the layer.

The method `pfcImport.ImportedLayer.SetAction` sets the display of the imported layers. The input parameter for this method is ImportAction. The types of actions that can be performed on the imported layers are:

- `IMPORT_LAYER_DISPLAY`—Displays the imported layer.
- `IMPORT_LAYER_SKIP`—Does not import entities on this layer.
- `IMPORT_LAYER_BLANK`—Blanks the selected layer.
- `IMPORT_LAYER_IGNORE`—Imports only entities on this layer but not the layer.

The default action type is `IMPORT_LAYER_DISPLAY`.

Plotting Files

From Pro/ENGINEER Wildfire 5.0 onwards, the `pfcModel.PlotInstructions` object containing the instructions for plotting files has been deprecated. All the methods listed below for creating and accessing the instruction attributes in `pfcModel.PlotInstructions` have also been deprecated. Use the new interface type `pfcExport.PrinterInstructions` and its methods described in the next section.

Methods Deprecated:

- **pfcModel.pfcModel.PlotInstructions_Create**
- **pfcModel.PlotInstructions.GetPlotterName**
- **pfcModel.PlotInstructions.SetPlotterName**
- **pfcModel.PlotInstructions.GetOutputQuality**
- **pfcModel.PlotInstructions.SetOutputQuality**
- **pfcModel.PlotInstructions.GetUserScale**
- **pfcModel.PlotInstructions.SetUserScale**
- **pfcModel.PlotInstructions.GetPenSlew**
- **pfcModel.PlotInstructions.SetPenSlew**
- **pfcModel.PlotInstructions.GetPenVelocityX**
- **pfcModel.PlotInstructions.SetPenVelocityX**
- **pfcModel.PlotInstructions.GetPenVelocityY**
- **pfcModel.PlotInstructions.SetPenVelocityY**
- **pfcModel.PlotInstructions.GetSegmentedOutput**
- **pfcModel.PlotInstructions.SetSegmentedOutput**
- **pfcModel.PlotInstructions.GetLabelPlot**

- **pfcModel.PlotInstructions.SetLabelPlot**
- **pfcModel.PlotInstructions.GetSeparatePlotFiles**
- **pfcModel.PlotInstructions.SetSeparatePlotFiles**
- **pfcModel.PlotInstructions.GetPaperSize**
- **pfcModel.PlotInstructions.SetPaperSize**
- **pfcModel.PlotInstructions.GetPageRangeChoice**
- **pfcModel.PlotInstructions.SetPageRangeChoice**
- **pfcModel.PlotInstructions.GetPaperSizeX**
- **pfcModel.PlotInstructions.SetPaperSizeY**
- **pfcModel.PlotInstructions.GetFirstPage**
- **pfcModel.PlotInstructions.SetFirstPage**
- **pfcModel.PlotInstructions.GetLastPage**
- **pfcModel.PlotInstructions.SetLastPage**

Printing Files

The printer instructions for printing a file are defined in `pfcExport.PrinterInstructions` data object.

Methods Introduced:

- **pfcExport.pfcExport.PrinterInstructions_Create**
- **pfcExport.PrinterInstructions.SetPrinterOption**
- **pfcExport.PrinterInstructions.SetPlacementOption**
- **pfcExport.PrinterInstructions.SetModelOption**
- **pfcExport.PrinterInstructions.SetWindowId**

The method `pfcExport.pfcExport.PrinterInstructions_Create` creates a new instance of the `pfcExport.PrinterInstructions` object. The object contains the following instruction attributes:

- *PrinterOption*—Specifies the printer settings for printing a file in terms of the `pfcExport.PrintPrinterOption` object. Set this attribute using the method `pfcExport.PrinterInstructions.SetPrinterOption`.
- *PlacementOption*—Specifies the placement options for printing purpose in terms of the `pfcExport.PrintMdlOption` object. Set this attribute using the method `pfcExport.PrinterInstructions.SetPlacementOption`.
- *ModelOption*—Specifies the model options for printing purpose in terms of the `pfcExport.PrintPlacementOption` object. Set this attribute

using the method `pfcExport.PrinterInstructions.SetModelOption`.

- *WindowId*—Specifies the current window identifier. Set this attribute using the method `pfcExport.PrinterInstructions.SetWindowId`.

Printer Options

The printer settings for printing a file are defined in the `pfcExport.PrintPrinterOption` object.

Methods Introduced:

- **pfcExport.pfcExport.PrintPrinterOption_Create**
- **pfcSession.BaseSession.GetPrintPrinterOptions**
- **pfcExport.PrintPrinterOption.SetDeleteAfter**
- **pfcExport.PrintPrinterOption.SetFileName**
- **pfcExport.PrintPrinterOption.SetPaperSize**
- **pfcExport.pfcExport.PrintSize_Create**
- **pfcExport.PrintSize.SetHeight**
- **pfcExport.PrintSize.SetWidth**
- **pfcExport.PrintSize.SetPaperSize**
- **pfcExport.PrintPrinterOption.SetPenTable**
- **pfcExport.PrintPrinterOption.SetPrintCommand**
- **pfcExport.PrintPrinterOption.SetPrinterType**
- **pfcExport.PrintPrinterOption.SetQuantity**
- **pfcExport.PrintPrinterOption.SetRollMedia**
- **pfcExport.PrintPrinterOption.SetRotatePlot**
- **pfcExport.PrintPrinterOption.SetSaveMethod**
- **pfcExport.PrintPrinterOption.SetSaveToFile**
- **pfcExport.PrintPrinterOption.SetSendToPrinter**
- **pfcExport.PrintPrinterOption.SetSlew**
- **pfcExport.PrintPrinterOption.SetSwHandshake**
- **pfcExport.PrintPrinterOption.SetUseTtf**

The method `pfcExport.pfcExport.PrintPrinterOption_Create` creates a new instance of the `pfcExport.PrintPrinterOption` object.

The method `pfcSession.BaseSession.GetPrintPrinterOptions` retrieves the printer settings.

The `pfcExport.PrintPrinterOption` object contains the following options:

- *DeleteAfter*—Determines if the file is deleted after printing. Set it to true to delete the file after printing. Use the method `pfcExport.PrintPrinterOption.SetDeleteAfter` to assign this option.
- *FileName*—Specifies the name of the file to be printed. Use the method `pfcExport.PrintPrinterOption.SetFileName` to set the name.

> **Note**
>
> If the method `pfcModel.Model.Export` is called for `pfcModel.ExportType` object, then the argument *FileName* is ignored, and can be passed as `NULL`. You must use the method `pfcModel.Model.Export` to set the *FileName*.

- *PaperSize*—Specifies the parameters of the paper to be printed in terms of the `pfcExport.PrintSize` object. The method `pfcExport.PrintPrinterOption.SetPaperSize` assigns the *PaperSize* option. Use the method `pfcExport.Export.PrintSize_Create` to create a new instance of the `pfcExport.PrintSize` object. This object contains the following options:
  - *Height*—Specifies the height of paper. Use the method `pfcExport.PrintSize.SetHeight` to set the paper height.
  - *Width*—Specifies the width of paper. Use the method `pfcExport.PrintSize.SetWidth` to set the paper width.
  - *PaperSize*—Specifies the size of the paper used for the plot in terms of the `pfcModel.PlotPaperSize` object. Use the method `pfcExport.PrintSize.SetPaperSize` to set the paper size.
- *PenTable*—Specifies the file containing the pen table. Use the method `pfcExport.PrintPrinterOption.SetPenTable` to set this option.
- *PrintCommand*—Specifies the command to be used for printing. Use the method `pfcExport.PrintPrinterOption.SetPrintCommand` to set the command.
- *PrinterType*—Specifies the printer type. Use the method `pfcExport.PrintPrinterOption.SetPrinterType` to assign the type.

- *Quantity*—Specifies the number of copies to be printed. Use the method `pfcExport.PrintPrinterOption.SetQuantity` to assign the quantity.
- *RollMedia*—Determines if roll media is to be used for printing. Set it to true to use roll media. Use the method `pfcExport.PrintPrinterOption.SetRollMedia` to assign this option.
- *RotatePlot*—Determines if the plot is rotated by 90 degrees. Set it to true to rotate the plot. Use the method `pfcExport.PrintPrinterOption.SetRotatePlot` to set this option.
- *SaveMethod*—Specifies the save method in terms of the `pfcExport.PrintSaveMethod` class. Use the method `pfcExport.PrintPrinterOption.SetSaveMethod` to specify the save method. The available methods are as follows:
  - `PRINT_SAVE_SINGLE_FILE`—Plot is saved to a single file.
  - `PRINT_SAVE_MULTIPLE_FILE`—Plot is saved to multiple files.
  - `PRINT_SAVE_APPEND_TO_FILE`—Plot is appended to a file.
- *SaveToFile*—Determines if the file is saved after printing. Set it to true to save the file after printing. Use the method `pfcExport.PrintPrinterOption.SetSaveToFile` to assign this option.
- *SendToPrinter*—Determines if the plot is directly sent to the printer. Set it to true to send the plot to the printer. Use the method `pfcExport.PrintPrinterOption.SetSendToPrinter` to set this option.
- *Slew*—Specifies the speed of the pen in centimeters per second in X and Y direction. Use the method `pfcExport.PrintPrinterOption.SetSlew` to set this option.
- *SwHandshake*—Determines if the software handshake method is to be used for printing. Set it to true to use the software handshake method. Use the method `pfcExport.PrintPrinterOption.SetSwHandshake` to set this option.
- *UseTtf*—Specifies whether `TrueType` fonts or stroked text is used for printing. Set this option to true to use `TrueType` fonts and to false to stroke all text. Use the method `pfcExport.PrintPrinterOption.SetUseTtf` to set this option.

Placement Options

The placement options for printing purpose are defined in the `pfcExport.PrintPlacementOption` object.

Methods Introduced:

- **pfcExport.pfcExport.PrintPlacementOption_Create**
- **pfcSession.BaseSession.GetPrintPlacementOptions**
- **pfcExport.PrintPlacementOption.SetBottomOffset**
- **pfcExport.PrintPlacementOption.SetClipPlot**
- **pfcExport.PrintPlacementOption.SetKeepPanzoom**
- **pfcExport.PrintPlacementOption.SetLabelHeight**
- **pfcExport.PrintPlacementOption.SetPlaceLabel**
- **pfcExport.PrintPlacementOption.SetScale**
- **pfcExport.PrintPlacementOption.SetShiftAllCorner**
- **pfcExport.PrintPlacementOption.SetSideOffset**
- **pfcExport.PrintPlacementOption.SetX1ClipPosition**
- **pfcExport.PrintPlacementOption.SetX2ClipPosition**
- **pfcExport.PrintPlacementOption.SetY1ClipPosition**
- **pfcExport.PrintPlacementOption.SetY2ClipPosition**

The method `pfcExport.pfcExport.PrintPlacementOption_Create` creates a new instance of the `pfcExport.PrintPlacementOption` object.

The method `pfcSession.BaseSession.GetPrintPlacementOptions` retrieves the placement options.

The `pfcExport.PrintPlacementOption` object contains the following options:

- *BottomOffset*—Specifies the offset from the lower-left corner of the plot. Use the method `pfcExport.PrintPlacementOption.SetBottomOffset` to set this option.
- *ClipPlot*—Specifies whether the plot is clipped. Set this option to true to clip the plot or to false to avoid clipping of plot. Use the method `pfcExport.PrintPlacementOption.SetClipPlot` to set this option.
- *KeepPanzoom*—Determines whether pan and zoom values of the window are used. Set this option to true use pan and zoom and false to skip them. Use the

method pfcExport.PrintPlacementOption.SetKeepPanzoom to set this option.

- *LabelHeight*—Specifies the height of the label in inches. Use the method pfcExport.PrintPlacementOption.SetLabelHeight to set this option.
- *PlaceLabel*—Specifies whether you want to place the label on the plot. Use the method pfcExport.PrintPlacementOption.SetPlaceLabel to set this option.
- *Scale*—Specifies the scale used for the plot. Use the method pfcExport.PrintPlacementOption.SetScale to set this option.
- *ShiftAllCorner*—Determines whether all corners are shifted.Set this option to true to shift all corners or to false to skip shifting of corners. Use the method pfcExport.PrintPlacementOption.SetShiftAllCorner to set this option.
- *SideOffset*—Specifies the offset from the sides. Use the method pfcExport.PrintPlacementOption.SetSideOffset to set this option.
- *X1ClipPosition*—Specifies the first X parameter for defining the clip position. Use the method pfcExport.PrintPlacementOption.SetX1ClipPosition to set this option.
- *X2ClipPosition*—Specifies the second X parameter for defining the clip position. Use the method pfcExport.PrintPlacementOption.SetX2ClipPosition to set this option.
- *Y1ClipPosition*—Specifies the first Y parameter for defining the clip position. Use the method pfcExport.PrintPlacementOption.SetY1ClipPosition to set this option.
- *Y2ClipPosition*—Specifies the second Y parameter for defining the clip position. Use the method pfcExport.PrintPlacementOption.SetY2ClipPosition to set this option.

Model Options

The model options for printing purpose are defined in the pfcExport.PrintMdlOption object.

Methods Introduced:

- **pfcExport.pfcExport.PrintMdlOption_Create**
- **pfcSession.BaseSession.GetPrintMdlOptions**
- **pfcExport.PrintMdlOption.SetDrawFormat**
- **pfcExport.PrintMdlOption.SetFirstPage**
- **pfcExport.PrintMdlOption.SetLastPage**
- **pfcExport.PrintMdlOption.SetLayerName**
- **pfcExport.PrintMdlOption.SetLayerOnly**
- **pfcExport.PrintMdlOption.SetMdl**
- **pfcExport.PrintMdlOption.SetQuality**
- **pfcExport.PrintMdlOption.SetSegmented**
- **pfcExport.PrintMdlOption.SetSheets**
- **pfcExport.PrintMdlOption.SetUseDrawingSize**
- **pfcExport.PrintMdlOption.SetUseSolidScale**

The method `pfcExport.pfcExport.PrintMdlOption_Create` creates a new instance of the `pfcExport.PrintMdlOption` object.

The method `pfcSession.BaseSession.GetPrintMdlOptions` retrieves the model options.

The `pfcExport.PrintMdlOption` object contains the following options:

- *DrawFormat*—Displays the drawing format used for printing. Use the method `pfcExport.PrintMdlOption.SetDrawFormat` to set this option.
- *FirstPage*—Specifies the first page number. Use the method `pfcExport.PrintMdlOption.SetFirstPage` to set this option.
- *LastPage*—Specifies the last page number. Use the method `pfcExport.PrintMdlOption.SetLastPage` to set this option.
- *LayerName*—Specifies the name of the layer. Use the method `pfcExport.PrintMdlOption.SetLayerName` to set the name.
- *LayerOnly*—Prints the specified layer only. Set this option to `true` to print the specified layer. Use the method `pfcExport.PrintMdlOption.SetLayerOnly` to set this option.
- *Mdl*—Specifies the model to be printed. Use the method `pfcExport.PrintMdlOption.SetMdl` to set this option.
- *Quality*—Determines the quality of the model to be printed. It checks for no line, no overlap, simple overlap, and complex overlap. Use the method `pfcExport.PrintMdlOption.SetQuality` to set this option.
- *Segmented*—If set to `true`, the printer prints the drawing in full size, but in segments that are compatible with the selected paper size. This option is

available only if you are plotting a single page. Use the method `pfcExport.PrintMdlOption.SetSegmented` to set this option.

- *Sheets*—Specifies the sheets that need to be printed in terms of the `pfcExport.PrintSheets` class. Use the method `pfcExport.PrintMdlOption.SetSheets` to specify the sheets. The sheets can be of the following types:
  - `PRINT_CURRENT_SHEET`—Only the current sheet is printed.
  - `PRINT_ALL_SHEETS`—All the sheets are printed.
  - `PRINT_SELECTED_SHEETS`—Sheets of a specified range are printed.
- *UseDrawingSize*—Overrides the paper size specified in the printer options with the drawing size. Set this option to `true` to use the drawing size. Use the method `pfcExport.PrintMdlOption.SetUseDrawingSize` to set this option.
- *UseSolidScale*—Prints with the scale used in the solid model. Set this option to `true` to use solid scale. Use the method `pfcExport.PrintMdlOption.SetUseSolidScale` to set this option.

Plotter Configuration File (PCF) Options

The printing options for PCF file are defined in the `Export.PrinterPCFOptions` object.

Methods Introduced:

- **pfcExport.pfcExport.PrinterPCFOptions_Create**
- **pfcExport.PrinterPCFOptions.SetPrinterOption**
- **pfcExport.PrinterPCFOptions.SetPlacementOption**
- **pfcExport.PrinterPCFOptions.SetModelOption**

The method `pfcExport.pfcExport.PrinterPCFOptions_Create` creates a new instance of the `pfcExport.PrinterPCFOptions` object.

The `pfcExport.PrinterPCFOptions` object contains the following options:

- *PrinterOption*—Specifies the printer settings for printing a file in terms of the `pfcExport.PrintPrinterOption` object. Set this attribute using the method `pfcExport.PrinterPCFOptions.SetPrinterOption`.
- *PlacementOption*—Specifies the placement options for printing purpose in terms of the `pfcExport.PrintMdlOption` object. Set this attribute using

the method `pfcExport.PrinterPCFOptions.SetPlacementOption`.

- *ModelOption*—Specifies the model options for printing purpose in terms of the `pfcExport.PrintPlacementOption` object. Set this attribute using the method `pfcExport.PrinterPCFOptions.SetModelOption`.

Solid Operations

Method Introduced:

- **pfcSolid.Solid.CreateImportFeat**

The method `pfcSolid.Solid.CreateImportFeat` creates a new import feature in the solid and takes the following input arguments:

- *IntfData*—The source of data from which to create the import feature. It is given by the `pfcModel.IntfDataSource` object. The type of source data that can be imported is given by the `pfcModel.IntfType` class and can be of the following types:
  - `INTF_NEUTRAL`
  - `INTF_NEUTRAL_FILE`
  - `INTF_IGES`
  - `INTF_STEP`
  - `INTF_VDA`
  - `INTF_ICEM`
  - `INTF_ACIS`
  - `INTF_DXF`
  - `INTF_CDRS`
  - `INTF_STL`
  - `INTF_VRML`
  - `INTF_PARASOLID`
  - `INTF_AI`
  - `INTF_CATIA_PART`
  - `INTF_UG`
  - `INTF_PRODUCTVIEW`
  - `INTF_CATIA_CGR`

    ○ INTF_JT

- *CoordSys*—The pointer to a reference coordinate system. If this is NULL, the function uses the default coordinate system.
- *FeatAttr*—The attributes for creation of the new import feature given by the pfcModel.ImportFeatAttr object. If this pointer is NULL, the function uses the default attributes.

Example Code: Returning a Feature Object

The sample code in the file pfcImportFeatureExample.java located at <creo_jlink_loadpoint>/jlink_appls/jlinkexamples return a feature object when provided with a solid coordinate system name and an import feature's file name. The method will find the coordinate system in the model, set the Import Feature Attributes, and create an import feature. The feature is then returned.

Window Operations

Method Introduced:

- **pfcWindow.Window.ExportRasterImage**

The method pfcWindow.Window.ExportRasterImage outputs a standard PTC Creo Parametric raster output file.

24

Simplified Representations

J-Link gives programmatic access to all the simplified representation functionality of PTC Creo Parametric. Create simplified representations either permanently or on the fly and save, retrieve, or modify them by adding or deleting items.

Overview

Using J-Link, you can create and manipulate assembly simplified representations just as you can using PTC Creo Parametric interactively.

Note

J-Link supports simplified representation of assemblies only, not parts.

Simplified representations are identified by the `pfcSimpRep.SimRep` class. This class is a child of `pfcModelItem.ModelItem`, so you can use the methods dealing with `pfcModelItems` to collect, inspect, and modify simplified representations.

The information required to create and modify a simplified representation is stored in a class called `pfcSimpRep.SimpRepInstructions` which contains several data objects and fields, including:

- `String`—The name of the simplified representation
- `pfcSimpRep.SimpRepAction`—The rule that controls the default treatment of items in the simplified representation.
- `pfcSimpRep.SimpRepItem`—An array of assembly components and the actions applied to them in the simplified representation.

A `pfcSimpRep.SimpRepItem` is identified by the assembly component path to that item. Each `pfcSimpRep.SimpRepItem` has it's own `pfcSimpRep.SimpRepAction` assigned to it. `pfcSimpRep.SimpRepAction` is a visible data object that includes a field of type `pfcSimpRep.SimpRepActionType`. You can use the method `pfcSimpRep.SimpRepAction.Action()` to set the actions. To delete an existing item, you must set the action as NULL.

`pfcSimpRep.SimpActionType` is an enumerated type that specifies the possible treatment of items in a simplified representation. The possible values are as follows

Values	Action
SIMPREP_NONE	No action is specified.
SIMPREP_REVERSE	Reverse the default rule for this component (for example, include it if the default rule is exclude).
SIMPREP_INCLUDE	Include this component in the simplified representation.
SIMPREP_EXCLUDE	Exclude this component from the simplified representation.
SIMPREP_SUBSTITUTE	Substitute the component in the simplified representation.
SIMPREP_GEOM	Use only the geometrical representation of the

Values	Action
	component.
SIMPREP_GRAPHICS	Use only the graphics representation of the component.
SIMPREP_SYMB	Use the symbolic representation of the component.

Retrieving Simplified Representations

Methods Introduced:

- **pfcSession.BaseSession.RetrieveAssemSimpRep**
- **pfcSession.BaseSession.RetrieveGeomSimpRep**
- **pfcSession.BaseSession.RetrieveGraphicsSimpRep**
- **pfcSession.BaseSession.RetrieveSymbolicSimpRep**
- **pfcSimpRep.pfcSimpRep.RetrieveExistingSimpRepInstructions_Create**

You can retrieve a named simplified representation from a model using the method `pfcSession.BaseSession.RetrieveAssemSimpRep`, which is analogous to the Assembly mode option **Retrieve Rep** in the **SIMPLFD REP** menu. This method retrieves the object of an existing simplified representation from an assembly without fetching the generic representation into memory. The method takes two arguments, the name of the assembly and the simplified representation data.

To retrieve an existing simplified representation, pass an instance of `pfcSimpRep.pfcSimpRep.RetrieveExistingSimpRepInstructions_Create` and specify its name as the second argument to this method. PTC Creo Parametric retrieves that representation and any active submodels and returns the object to the simplified representation as a `pfcAssembly.Assembly` object.

You can retrieve geometry, graphics, and symbolic simplified representations into session using the methods `pfcSession.BaseSession.RetrieveGeomSimpRep`, `pfcSession.BaseSession.RetrieveGraphicsSimpRep`, and `pfcSession.BaseSession.RetrieveSymbolicSimpRep` respectively. Like `pfcSession.BaseSession.RetrieveAssemSimpRep`, these methods retrieve the simplified representation without bringing the master representation into memory. Supply the name of the assembly whose simplified representation is to be retrieved as the input parameter for these methods. The methods output the assembly. They do not display the simplified representation.

Creating and Deleting Simplified Representations

Methods Introduced:

- **pfcSimpRep.pfcSimpRep.CreateNewSimpRepInstructions_Create**
- **pfcSolid.Solid.CreateSimpRep**
- **pfcSolid.Solid.DeleteSimpRep**

To create a simplified representation, you must allocate and fill a `pfcSimpRep.SimpRepInstructions` object by calling the method `pfcSimpRep.pfcSimpRep.CreateNewSimpRepInstructions_Create`. Specify the name of the new simplified representation as an input to this method. You should also set the default action type and add `SimpRepItems` to the object.

To generate the new simplified representation, call `pfcSolid.Solid.CreateSimpRep`. This method returns the `pfcSimpRep.SimpRep` object for the new representation.

The method `pfcSolid.Solid.DeleteSimpRep` deletes a simplified representation from its model owner. The method requires only the `pfcSimpRep.SimpRep` object as input.

Extracting Information About Simplified Representations

Methods Introduced:

- **pfcSimpRep.SimpRep.GetInstructions**
- **pfcSimpRep.SimpRepInstructions.GetDefaultAction**
- **pfcSimpRep.CreateNewSimpRepInstructions.GetNewSimpName**
- **pfcSimpRep.SimpRepInstructions.GetIsTemporary**
- **pfcSimpRep.SimpRepInstructions.GetItems**

Given the object to a simplified representation, `pfcSimpRep.SimpRep.GetInstructions` fills out the `pfcSimpRep.SimpRepInstructions` object.

The `pfcSimpRep.SimpRepInstructions.GetDefaultAction`, `pfcSimpRep.CreateNewSimpRepInstructions.GetNewSimpName`, and `pfcSimpRep.SimpRepInstructions.GetIsTemporary` methods return the associated values contained in the `pfcSimpRep.SimpRepInstructions`object.

The method pfcSimpRep.SimpRepInstructions.GetItems returns all the items that make up the simplified representation.

Modifying Simplified Representations

Methods Introduced:

- **pfcSimpRep.SimpRep.GetInstructions**
- **pfcSimpRep.SimpRep.SetInstructions**
- **pfcSimpRep.SimpRepInstructions.SetDefaultAction**
- **pfcSimpRep.CreateNewSimpRepInstructions.SetNewSimpName**
- **pfcSimpRep.SimpRepInstructions.SetIsTemporary**

Using J-Link, you can modify the attributes of existing simplified representations. After you create or retrieve a simplified representation, you can make calls to the methods listed in this section to designate new values for the fields in the pfcSimpRep.SimpRepInstructions object.

To modify an existing simplified representation retrieve it and then get the pfcSimpRep.SimpRepInstructions object by calling pfcSimpRep.SimpRep.GetInstructions. If you created the representation programmatically within the same application, the pfcSimpRep.SimpRepInstructions object is already available. Once you have modified the data object, reassign it to the corresponding simplified representation by calling the method pfcSimpRep.SimpRep.SetInstructions.

Adding Items to and Deleting Items from a Simplified Representation

Methods Introduced:

- **pfcSimpRep.SimpRepInstructions.SetItems**
- **pfcSimpRep.pfcSimpRep.SimpRepItem_Create**
- **pfcSimpRep.SimpRep.SetInstructions**
- **pfcSimpRep.pfcSimpRep.SimpRepReverse_Create**
- **pfcSimpRep.pfcSimpRep.SimpRepInclude_Create**
- **pfcSimpRep.pfcSimpRep.SimpRepExclude_Create**
- **pfcSimpRep.pfcSimpRep.SimpRepSubstitute_Create**
- **pfcSimpRep.pfcSimpRep.SimpRepGeom_Create**
- **pfcSimpRep.pfcSimpRep.SimpRepGraphics_Create**

You can add and delete items from the list of components in a simplified representation using J-Link. If you created a simplified representation using the option **Exclude** as the default rule, you would generate a list containing the items you want to include. Similarly, if the default rule for a simplified representation is **Include**, you can add the items that you want to be excluded from the simplified representation to the list, setting the value of the `pfcSimpRep.SimpRepActionType` to `SIMPREP_EXCLUDE`.

How to Add Items

1. Get the `pfcSimpRep.SimpRepInstructions` object, as described in the previous section.
2. Specify the action to be applied to the item with a call to one of following methods.
3. Initialize a `pfcSimpRep.SimpRepItem` object for the item by calling the method `pfcSimpRep.pfcSimpRep.SimpRepItem_Create`.
4. Add the item to the `pfcSimpRep.SimpRepItem` sequence. Put the new `pfcSimpRep.SimpRepInstructions` using `pfcSimpRep.SimpRepInstructions.SetItems`.
5. Reassign the `pfcSimpRep.SimpRepInstructions` object to the corresponding `pfcSimpRep.SimpRep` object by calling `pfcSimpRep.SimpRep.SetInstructions`

How to Remove Items

Follow the procedure above, except remove the unwanted `pfcSimpRep.SimpRepItem` from the sequence.

Simplified Representation Utilities

Methods Introduced:

- **pfcModelItem.ModelItemOwner.ListItems**
- **pfcModelItem.ModelItemOwner.GetItemById**
- **pfcSolid.Solid.GetSimpRep**
- **pfcSolid.Solid.SelectSimpRep**
- **pfcSolid.Solid.ActivateSimpRep**
- **pfcSolid.Solid.GetActiveSimpRep**

This section describes the utility methods that relate to simplified representations.

The method `pfcModelItem.ModelItemOwner.ListItems` can list all of the simplified representations in a Solid.

The method `pfcModelItem.ModelItemOwner.GetItemById` initializes a `pfcSimpRep.SimpRep` object. It takes an integer id.

Note

J-Link supports simplified representation of Assemblies only, not Parts.

The method `pfcSolid.Solid.GetSimpRep` initializes a `pfcSimpRep.SimpRep` object. The method takes the following arguments:

- *SimpRepname* The name of the simplified representation in the solid. If you specify this argument, the method ignores the *rep_id*.

The method `pfcSolid.Solid.SelectSimpRep` creates a PTC Creo Parametric menu to enable interactive selection. The method takes the owning solid as input, and outputs the object to the selected simplified representation. If you choose the **Quit** menu button, the method throws an exception `XToolkitUserAbort`

The methods `pfcSolid.Solid.GetActiveSimpRep` and `pfcSolid.Solid.ActivateSimpRep` enable you to find and get the currently active simplified representation, respectively. Given an assembly object, `pfcSolid.Solid.GetActiveSimpRep` returns the object to the currently active simplified representation. If the current representation is the master representation, the return is null.

The method `pfcSolid.Solid.ActivateSimpRep` activates the requested simplified representation.

To set a simplified representation to be the currently displayed model, you must also call `pfcModel.ModelDisplay`.

25

Asynchronous Mode

This chapter explains how to use J-Link in Asynchronous Mode.

Overview

Asynchronous mode is a multiprocess mode in which the J-Link application and PTC Creo Parametric can perform concurrent operations. Unlike the synchronous modes, asynchronous mode uses JNI (Java Native Interface) and RPC (remote procedure calls) as the means of communication between the application and PTC Creo Parametric.

Another important difference between synchronous and asynchronous modes is in the startup of the J-Link application. In synchronous mode, the application is started by PTC Creo Parametric, based on information contained in the registry file. In asynchronous mode, the application (containing its own main() method) is started independently of PTC Creo Parametric and subsequently either starts or connects to a PTC Creo Parametric process.

> **Note**
>
> An asynchronous application that starts PTC Creo Parametric will not appear in the **Auxiliary Applications** dialog box.

The use of RPC causes asynchronous mode to perform more slowly than synchronous mode. For this reason, apply asynchronous mode only when it is needed.

An asynchronous mode is not the only mode in which your application has explicit control over PTC Creo Parametric. Because PTC Creo Parametric calls a Java start method when an application starts, your synchronous application can take control by initiating all operations in Java start method (thus interrupting any user interaction). This technique is important when you want to run PTC Creo Parametric in batch mode.

Depending on how your asynchronous application handles messages from PTC Creo Parametric, your application can be classified as either `simple` or `full`. The following sections describe simple and full asynchronous mode.

Setting up an Asynchronous J-Link Application

For your asynchronous application to communicate with PTC Creo Parametric, you must set the environment variable `PRO_COMM_MSG_EXE` to the full path of the executable `pro_comm_msg`.

On Windows systems, set `PRO_COMM_MSG_EXE` in the **Environment** section of the **System** window that you access from the **Control Panel**.

To support the asynchronous mode, use the jar file `pfcasync.jar` in your CLASSPATH. This file is available at <creo_loadpoint>\<datecode>\Common Files\text\java. This file contains all required classes for running with asynchronous J-Link.

Note

Asynchronous applications are incompatible with the classes in the synchronous `.jar` files. You must build and run your application classes specifically to run in asynchronous mode.

You must add the asynchronous library, `pfcasyncmt`, to your environment that launches the J-Link application. This library is stored in <creo_loadpoint>\<datecode>\Common Files\<machine type>\<lib>.

Note

The library has prefix and extension specifiers for a dynamically loaded library for the platform being used.

System	Library	Path
i486_nt, x86e_win64	pfcasyncmt.dll	set PATH=<creo_loadpoint>\<datecode>\Common Files\<machine Type>\lib;%PATH%

Asynchronous J-Link applications must load the library prior to calls made to the asynchronous methods. This can be accomplished by adding the following line to your application.

`System.loadLibrary` (“`pfcasyncmt`”)

Simple Asynchronous Mode

A `simple` asynchronous application does not implement a way to handle requests from PTC Creo Parametric. Therefore, J-Link cannot plant listeners to be notified when events happen in PTC Creo Parametric. Consequently, PTC Creo Parametric cannot invoke the methods that must be supplied when you add, for example, menu buttons to PTC Creo Parametric.

Despite this limitation, a simple asynchronous mode application can be used to automate processes in PTC Creo Parametric. The application may either start or connect to an existing PTC Creo Parametric session, and may access PTC Creo

Parametric in interactive or in a non graphical, non interactive mode. When PTC Creo Parametric is running with graphics, it is an interactive process available to the user.

When you design a J-Link application to run in simple asynchronous mode, keep the following points in mind:

- The PTC Creo Parametric process and the application perform operations concurrently.
- None of the application's listener methods can be invoked by PTC Creo Parametric.

Simple asynchronous mode supports normal J-Link methods but does not support ActionListeners. These considerations imply that the J-Link application does not know the state (the current mode, for example) of the PTC Creo Parametric process at any moment.

Starting and Stopping PTC Creo Parametric

The following methods are used to start and stop PTC Creo Parametric when using J-Link applications.

Methods Introduced:

- **pfcAsyncConnection.pfcAsyncConnection.AsyncConnection_Start**
- **pfcAsyncConnection.AsyncConnection.End**

A simple asynchronous application can spawn and connect to a PTC Creo Parametric process with the method `pfcAsyncConnection.pfcAsyncConnection.AsyncConnection_Start`. The PTC Creo Parametric process listens for requests from the application and acts on the requests at suitable breakpoints, usually between commands.

Unlike applications running in synchronous mode, asynchronous applications are not terminated when PTC Creo Parametric terminates. This is useful when the application needs to perform PTC Creo Parametric operations intermittently, and therefore, must start and stop PTC Creo Parametric more than once during a session.

The application can connect to or start only one PTC Creo Parametric session at any time. If the J-Link application spawns a second session, connection to the first session is lost.

To end any PTC Creo Parametric process that the application is connected to, call the method `pfcAsyncConnection.AsyncConnection.End`.

Setting Up a Noninteractive Session

You can spawn a PTC Creo Parametric session that is both noninteractive and nongraphical. In asynchronous mode, include the following strings in the PTC Creo Parametric start or connect call to `pfcAsyncConnection.pfcAsyncConnection.AsyncConnection_Start`:

- `-g:no_graphics`—Turn off the graphics display.
- `-i:rpc_input`—Causes PTC Creo Parametric to expect input from your asynchronous application only.

> **Note**
>
> Both of these arguments are required, but the order is not important.

The syntax of the call for a noninteractive, nongraphical session is as follows:

```
pfcAsyncConnection.pfcAsyncConnection.AsyncConnection_Start
("pro -g:no_graphics -i:rpc_input",<text_dir>);
```

where `pro` is the command to start PTC Creo Parametric.

Example Code

The sample code in the file `pfcAsyncStartExample.java` located at `<creo_jlink_loadpoint>/jlink_appls/jlinkasyncexamples` demonstrates how to use J-Link in asynchronous mode. The method starts PTC Creo Parametric asynchronously, retrieves a Session, and opens a model in PTC Creo Parametric.

Connecting to a PTC Creo Parametric Process

Methods Introduced:

- **pfcAsyncConnection.pfcAsyncConnection.AsyncConnection_Connect**
- **pfcAsyncConnection.pfcAsyncConnection.AsyncConnection_GetActiveConnection**
- **pfcAsyncConnection.AsyncConnection.Disconnect**

A simple asynchronous application can also connect to a PTC Creo Parametric process that is already running on a local computer. The method `pfcAsyncConnection.pfcAsyncConnection.AsyncConnection_Connect` performs this connection. This method fails to connect if multiple PTC

Creo Parametric sessions are running. If several versions of PTC Creo Parametric are running on the same computer, try to connect by specifying user and display parameters. However, if several versions of PTC Creo Parametric are running in the same user and display parameters, the connection may not be possible.

pfcAsyncConnection.pfcAsyncConnection.AsyncConnection_GetActiveConnection returns the current connection to a PTC Creo Parametric session.

To disconnect from a PTC Creo Parametric process, call the method pfcAsyncConnection.AsyncConnection.Disconnect. This method can be called only if you used the method pfcAsyncConnection.pfcAsyncConnection.AsyncConnection_Connect to get the connection.

The connection to a PTC Creo Parametric process uses information provided by the name service daemon. The name service daemon accepts and supplies information about the processes running on the specified hosts. The application manager, for example, uses the name service when it starts up PTC Creo Parametric and other processes. The name service daemon is set up as part of the PTC Creo Parametric installation.

Connecting Via Connection ID

Methods Introduced:

- **pfcAsyncConnection.AsyncConnection.GetConnectionId**
- **pfcAsyncConnection.ConnectionId.GetExternalRep**
- **pfcSession.BaseSession.GetConnectionId**
- **pfcAsyncConnection.pfcAsyncConnection.ConnectionId_Create**
- **pfcAsyncConnection.pfcAsyncConnection.AsyncConnection_ConnectById**

Each PTC Creo Parametric process maintains a unique identity for communications purposes. Use this ID to reconnect to a PTC Creo Parametric process.

The method pfcAsyncConnection.AsyncConnection.GetConnectionId returns a data structure containing the connection ID.

If the connection id must be passed to some other application the method pfcAsyncConnection.ConnectionId.GetExternalRep provides the string external representation for the connection ID.

The method pfcSession.BaseSession.GetConnectionId provides access to the asynchronous connection ID for the current PTC Creo Parametric session. This ID can be passed to any asynchronous mode application that needs to connect to the current session of PTC Creo Parametric.

The method pfcAsyncConnection.pfcAsyncConnection.ConnectionId_Create takes a string representation and creates a ConnectionId data object. The method pfcAsyncConnection.pfcAsyncConnection.AsyncConnection_ConnectById connects to PTC Creo Parametric at the specified connection ID.

Note

Connection IDs are unique for each PTC Creo Parametric process and are not maintained after you quit PTC Creo Parametric.

Status of a PTC Creo Parametric Process

Method Introduced:

- **pfcAsyncConnection.AsyncConnection.IsRunning**

To find out whether a PTC Creo Parametric process is running, use the method pfcAsyncConnectionAsyncConnection.IsRunning.

Getting the Session Object

Method Introduced:

- **pfcAsyncConnection.AsyncConnection.GetSession**

The method pfcAsyncConnection.AsyncConnection.GetSession returns the session object representing the PTC Creo Parametric session. Use this object to access the contents of the PTC Creo Parametric session. See the Session Objects on page 65 chapter for additional information.

Full Asynchronous Mode

Full asynchronous mode is identical to the simple asynchronous mode except in the way the J-Link application handles requests from PTC Creo Parametric. In simple asynchronous mode, it is not possible to process these requests. In full asynchronous mode, the application implements a control loop that “listens” for messages from PTC Creo Parametric. As a result, PTC Creo Parametric can call functions in the application, including callback functions for menu buttons and notifications.

Note

Using full asynchronous mode requires starting or connecting to PTC Creo Parametric using the methods described in the previous sections. The difference is that the application must provide an event loop to process calls from menu buttons and listeners.

Methods Introduced:

- **pfcAsyncConnection.AsyncConnection.EventProcess**
- **pfcAsyncConnection.AsyncConnection.WaitForEvents**
- **pfcAsyncConnection.AsyncConnection.InterruptEventProcessing**
- **pfcAsyncConnection.AsyncActionListener.OnTerminate**

The control loop of an application running in full asynchronous mode must contain a call to the method `pfcAsyncConnection.AsyncConnection.EventProcess`, which takes no arguments. This method allows the application to respond to messages sent from PTC Creo Parametric. For example, if the user selects a menu button that is added by your application, `pfcAsyncConnection.AsyncConnection.EventProcess` processes the call to your listener and returns when the call completes. For more information on listeners and adding menu buttons, see the Session Objects on page 65 chapter.

The method `pfcAsyncConnection.AsyncConnection.WaitForEvents` provides an alternative to the development of an event processing loop in a full asynchronous mode application. Call this function to have the application wait in a loop for events to be passed from PTC Creo Parametric. No other processing takes place while the application is waiting. The loop continues until `pfcAsyncConnection.AsyncConnection.InterruptEventProcessing` is called from a J-Link callback action, or until the application detects the termination of PTC Creo Parametric.

It is often necessary for your full asynchronous application to be notified of the termination of the PTC Creo Parametric process. In particular, your control loop need not continue to listen for PTC Creo Parametric messages if PTC Creo Parametric is no longer running.

An `AsyncConnection` object can be assigned an Action Listener to bind a termination action that is executed upon the termination of PTC Creo Parametric. The method `pfcAsyncConnection.AsyncActionListener.OnTerminate` handles the termination that you must override. It sends a member of the class `pfcAsyncConnection.TerminationStatus`, which is one of the following:

- `TERM_EXIT`—Normal exit (the user clicks **Exit** on the menu).
- `TERM_ABNORMAL`—Quit with error status.
- `TERM_SIGNAL`—Fatal signal raised.

Your application can interpret the termination type and take appropriate action. For more information on Action Listeners, see the Action Listeners on page 291 chapter.

Example Code

The sample code in the file `pfcAsyncFullExample.java` located at `<creo_jlink_loadpoint>/jlink_appls/jlinkasyncexamples` is a fully asynchronous application. It follows the procedure for a full asynchronous application:

1. The application establishes listeners for PTC Creo Parametric events, in this case, the menu button and the termination listener.
2. The application goes into a control loop calling **EventProcess** which allows the application to respond to the PTC Creo Parametric events.

Message and Menu File

```
J-Link
J-Link
#
#
AsyncApp
Hit me!
#
#
AsyncAppHelp
Launch async application callback
#
#
```

Troubleshooting Asynchronous J-Link

General Problems

UnsatisfiedLinkError in System.loadLibrary ("pfcasyncmt")

Add `$PRO_DIRECTORY/$PRO_MACHINE_TYPE/lib` to your library path:

Windows and Windows XP 64bit:$PATH (separated with semicolon)

Java gives this exception when it has any trouble loading the library, not just when the library is not in the library path. If you are working in a non-standard configuration make sure that all of these libraries are in your library path (subject to your OS naming, for example `cipstdmtz.dll` on Windows):

- `pfcasyncmt`
- `jnicipjavamtz`
- `jniadaptsmtz`
- `cipstdmtz`
- `ctoolsmtz`
- `baselibmtz`
- `i18nmtz`

Look at what is printed on `stdout/stderr`. There can be unresolved symbols. Windows usually reports unresolved symbols in a pop-up dialog so you will see it immediately. If that does not help, then enable the debug output from the operating system's dynamic loader, start with reading the main page.

UnsatisfiedLinkError on first call to a JLink method

Ensure that you executed the `System.loadLibrary ("pfcasyncmt")`. Put a debug printout right after it to ensure it gets loaded.

In most cases the J-Link jar files (`pfcasync.jar`) are loaded using a non-system class loader. Java lets you load native libraries from classes loaded with either the system class loader (`pfcasync.jar`must be in the default CLASSPATH), or a signed class loader. Java will not throw an exception on `System.loadLibrary`. For this reason everything will appear to be fine until the first call to a native method. At this point you will get an `UnsatisfiedLinkError`. Add J-Link jar files to the default CLASSPATH, usually to the CLASSPATH environment or an appropriate place in your servlet engine's configuration.

NullPointerException from a JLink method early in program execution

Make sure that you have jar files from only one version of J-Link in your CLASSPATH. If you have both async and sync jar files, the VM will pick up incorrect classes.

- Sync J-Link jars:
- Async J-Link jars:

pfcExceptions.XToolkitNotFound exception on the first call to pfcAsyncConnection.pfcAsyncConnection.AsyncConnection_Start on Windows

Make sure your PTC Creo Parametric command is correct. If it's not a full path to a script/executable, make sure `$PATH` is set correctly. Try full path in the command: if it works, then your `$PATH`is incorrect.

pfcExceptions.XToolkitGeneralError or pfcExceptions.CommError on the first call to pfcAsyncConnection.pfcAsyncConnection.AsyncConnection_Start or pfcAsyncConnection.pfcAsyncConnection.AsyncConnection_Connect

- Make sure the environment variable `PRO_COMM_MSG_EXE` is set to full path to `pro_comm_msg`, including file name, including `.exe` on Windows.
- Make sure the environment variable `PRO_DIRECTORY` is set to PTC Creo Parametric installation directory.
- Make sure name service () is running.

pfcAsyncConnection.pfcAsyncConnection.AsyncConnection_Start hangs, even though PTC Creo Parametric already started

Make sure name service () is also started with PTC Creo Parametric. Open **Task Manager** and look for `nmsd.exe` in the process listing.

Problems Specific to Servlets and JSP

pfcExceptions.XToolkitGeneralError or pfcExceptions.CommError on the first call to pfcAsyncConnection.pfcAsyncConnection.AsyncConnection_Start or pfcAsyncConnection.pfcAsyncConnection.AsyncConnection_Connect

- Make sure you have `PRO_COMM_MSG_EXE` and `PRO_DIRECTORY` set correctly.
- On Windows, servlet engine deployments typically belong to the SYSTEM account and not a local user account. So, you must set `PRO_COMM_MSG_EXE` and `PRO_DIRECTORY` in your system environment and restart Windows to cause this change to take effect.

26

Task Based Application Libraries

Applications created using different PTC Creo Parametric API products are interoperable. These products use PTC Creo Parametric as the medium of interaction, eliminating the task of writing native-platform specific interactions between different programming languages.

Application interoperability allows J-Link applications to call into PTC Creo Parametric TOOLKIT from areas not covered in the native interface. It allows you to put a Java front end on legacy PTC Creo Parametric TOOLKIT applications, and also allows you to use J-Link applications and listeners in conjunction with a or asynchronous J-Link application.

J-Link can call PTC Creo Parametric web pages belonging to Web.Link, and functions in PTC Creo Parametric TOOLKIT DLLs. J-Link synchronous applications can also register tasks for use by other applications.

Managing Application Arguments

J-Link passes application data to and from tasks in other applications as members of a sequence of `pfcArgument.Argument` objects. Application arguments consist of a label and a value. The value may be of any one of the following types:

- Integer
- Double
- Boolean
- ASCII string (a non-encoded string, provided for compatibility with arguments provided from C applications)
- String (a fully encoded string)
- `pfcSelect.Selection` (a selection of an item in a PTC Creo Parametric session)
- `pfcBase.Transform3D` (a coordinate system transformation matrix)

Methods Introduced:

- **pfcArgument.pfcArgument.CreateIntArgValue**
- **pfcArgument.pfcArgument.CreateDoubleArgValue**
- **pfcArgument.pfcArgument.CreateBoolArgValue**
- **pfcArgument.pfcArgument.CreateASCIIStringArgValue**
- **pfcArgument.pfcArgument.CreateStringArgValue**
- **pfcArgument.pfcArgument.CreateSelectionArgValue**
- **pfcArgument.pfcArgument.CreateTransformArgValue**
- **pfcArgument.ArgValue.Getdiscr**
- **pfcArgument.ArgValue.GetIntValue**
- **pfcArgument.ArgValue.SetIntValue**
- **pfcArgument.ArgValue.GetDoubleValue**
- **pfcArgument.ArgValue.SetDoubleValue**
- **pfcArgument.ArgValue.GetBoolValue**
- **pfcArgument.ArgValue.SetBoolValue**
- **pfcArgument.ArgValue.GetASCIIStringValue**
- **pfcArgument.ArgValue.SetASCIIStringValue**
- **pfcArgument.ArgValue.GetStringValue**
- **pfcArgument.ArgValue.SetStringValue**
- **pfcArgument.ArgValue.GetSelectionValue**
- **pfcArgument.ArgValue.SetSelectionValue**

- **pfcArgument.ArgValue.GetTransformValue**
- **pfcArgument.ArgValue.SetTransformValue**

The class `pfcArgument ArgValue` contains one of the seven types of values. J-Link provides different methods to create each of the seven types of argument values.

The method `pfcArgument.ArgValue.Getdiscr` returns the type of value contained in the argument value object.

Use the methods listed above to access and modify the argument values.

Modifying Arguments

Methods Introduced:

- **pfcArgument.pfcArgument.Argument_Create**
- **pfcArgument.Arguments.create**
- **pfcArgument.Argument.GetLabel**
- **pfcArgument.Argument.SetLabel**
- **pfcArgument.Argument.GetValue**
- **pfcArgument.Argument.SetValue**

The method `pfcArgument.pfcArgument.Argument_Create` creates a new argument. Provide a name and value as the input arguments of this method.

The method `pfcArgument.Arguments.create` creates a new empty sequence of task arguments.

The method `pfcArgument.Argument.GetLabel` returns the label of the argument. The method `pfcArgument.Argument.SetLabel` sets the label of the argument.

The method `pfcArgument.Argument.GetValue` returns the value of the argument. The method `pfcArgument.Argument.SetValue` sets the value of the argument.

Launching a PTC Creo Parametric TOOLKIT DLL

The methods described in this section enable J-Link user to register and launch a PTC Creo Parametric TOOLKIT DLL from a J-Link application. The ability to launch and control a PTC Creo Parametric TOOLKIT application enables the following:

- Reuse of existing PTC Creo Parametric TOOLKIT code with J-Link applications.
- ATB operations.

Methods Introduced:

- **pfcSession.BaseSession.LoadProToolkitDll**
- **pfcSession.BaseSession.LoadProToolkitLegacyDll**
- **pfcSession.BaseSession.GetProToolkitDll**
- **pfcProToolkit.Dll.ExecuteFunction**
- **pfcProToolkit.Dll.GetId**
- **pfcProToolkit.Dll.IsActive**
- **pfcProToolkit.Dll.Unload**

Use the method `pfcSession.BaseSession.LoadProToolkitDll` to register and start a PTC Creo Parametric TOOLKIT DLL. The input parameters of this method are similar to the fields of a registry file and are as follows:

- *ApplicationName*—The name of the application to initialize.
- *DllPath*—The full path to the DLL binary file.
- *TextPath*—The path to the application's message and user interface text files.
- *UserDisplay*—Set this parameter to `true` to register the application in the PTC Creo Parametric user interface and to see error messages if the application fails. If this parameter is `false`, the application will be invisible to the user.

The application's `user_initialize()` function is called when the application is started. The method returns a handle to the loaded PTC Creo Parametric TOOLKIT DLL.

In order to register and start a legacy Pro/TOOLKIT DLL that is not Unicode-compliant, use the method `pfcSession.BaseSession.LoadProToolkitLegacyDll`. This method conveys to PTC Creo Parametric that the loaded DLL application is not Unicode-compliant and built in the pre-Wildfire 4.0 environment. It takes the same input parameters as the earlier method `pfcSession.BaseSession.LoadProToolkitDll`.

> **Note**
>
> The method `pfcSession.BaseSession.LoadProToolkitLegacyDll` must be used only by a pre-Wildfire 4.0 J-Link application to load a pre-Wildfire 4.0 Pro/TOOLKIT DLL.

Use the method `pfcSession.BaseSession.GetProToolkitDll` to obtain a PTC Creo Parametric TOOLKIT DLL handle. Specify the *Application_Id*, that is, the DLL's identifier string as the input parameter of this method. The method returns the DLL object or null if the DLL was not in session. The *Application_Id* can be determined as follows:

- Use the function `ProToolkitDllIdGet()` within the DLL application to get a string representation of the DLL application. Pass `NULL` to the first argument of `ProToolkitDllIdGet()` to get the string identifier for the calling application.
- Use the `Get` method for the `Id` attribute in the DLL interface. The method `pfcProToolkit.Dll.GetId()` returns the DLL identifier string.

Use the method `pfcProToolkit.Dll.ExecuteFunction` to call a properly designated function in the PTC Creo Parametric TOOLKIT DLL library. The input parameters of this method are:

- *FunctionName*—Name of the function in the PTC Creo Parametric TOOLKIT DLL application.
- *InputArguments*—Input arguments to be passed to the library function.

The method returns an object of interface `com.ptc.pfc.pfcProToolkit.FunctionReturn`. This interface contains data returned by a PTC Creo Parametric TOOLKIT function call. The object contains the return value, as integer, of the executed function and the output arguments passed back from the function call.

The method `pfcProToolkit.Dll.IsActive` determines whether a PTC Creo Parametric TOOLKIT DLL previously loaded by the method `pfcSession.BaseSession.LoadProToolkitDll` is still active.

The method `pfcProToolkit.Dll.Unload` is used to shutdown a PTC Creo Parametric TOOLKIT DLL previously loaded by the method `pfcSession.BaseSession.LoadProToolkitDll` and the application's `user_terminate()` function is called.

Creating J-Link Task Libraries

The methods described in this section allow you to setup J-Link libraries to be used and called from other custom PTC Creo Parametric applications in PTC Creo Parametric TOOLKIT or J-Link.

J-Link task libraries must be compiled using the synchronous J-Link library called pfc.jar. Each task must be registered within the application for it to be called from external applications. Provide the following information to the application to use your J-Link application as a task library:

1. The required `CLASSPATH` settings.
2. The name of the invocation class containing the static start and stop methods.
3. The name of static `Start()` and `Stop()` methods
4. The path to the text files, if the application deals with messages or menus.
5. The registration name of the task.
6. The input argument names and types.
7. The output argument names and types.

Methods Introduced:

- **pfcSession.BaseSession.RegisterTask**
- **pfcJLink.JLinkTaskListener.OnExecute**
- **pfcSession.BaseSession.UnregisterTask**

Use the method `pfcSession.BaseSession.RegisterTask` to register the task or tasks to be executed. This method has two input parameters:

- The name of the task. This name must be provided by calling applications.
- Object implementing the interface `JLinkTaskListener` that has the `pfcJLinkTaskListener.OnExecute` callback method overridden. The class `pfcLink.DefaultJLinkTaskListener` makes extending the interface easier.

The method `pfcJLinkTaskListener.OnExecute` is called when the calling application tries to invoke a registered task. This method must contain the body of the J-Link task. The method signature includes a sequence of input arguments and allows you to return a sequence of output arguments to the caller.

Use the method `pfcSession.BaseSession.UnregisterTask` to delete a task that is no longer needed. This method must be called when you exit the application using the application's stop method.

Launching Tasks from J-Link Task Libraries

The methods described in this section allow you to launch tasks from a predefined J-Link task library.

Methods Introduced:

- **pfcSession.BaseSession.StartJLinkApplication**
- **pfcJLink.JLinkApplication.ExecuteTask**
- **pfcJLink.JLinkApplication.IsActive**
- **pfcJLink.JLinkApplication.Stop**

Use the method `pfcSession.BaseSession.StartJLinkApplication` to start a J-Link application. The input parameters of this method are similar to the fields of a registry file and are as follows:

- *ApplicationName*—Assigns a unique name to this J-Link application.
- *ClassName*—Specifies the name of the Java class that contains the J-Link application's start and stop method. This should be a fully qualified Java package and class name.
- *StartMethod*—Specifies the start method of the J-Link application.
- *StopMethod*—Specifies the stop method of the J-Link application.
- *AdditionalClassPath*—Specifies the locations of packages and classes that must be loaded when starting this J-Link application. If this parameter is specified as null, the default classpath locations are used.
- *TextPath*—Specifies the application text path for menus and messages. If this parameter is specified as null, the default text locations are used.
- *UserDisplay*—Specifies whether to display the application in the **Auxiliary Applications** dialog box in PTC Creo Parametric.

Upon starting the application, the static `start()` method is invoked. The method returns a `JLink.JLinkApplication` referring to the J-Link application.

The method `pfcJLink.JLinkApplication.ExecuteTask` calls a registered task method in a J-Link application. The input parameters of this method are:

- Name of the task to be executed.
- A sequence of name value pair arguments contained by the interface `pfcArguments.Arguments`.

The method outputs an array of output arguments.These arguments are returned by the task's implementation of the `pfcJLinkTaskListener.OnExecute` call back method.

The method `pfcJLink.JLinkApplication.IsActive` returns a `True` value if the application specified by the `pfcJLink.JLinkApplication` object is active.

The method `pfcJLink.JLinkApplication.Stop` stops the application specified by the `pfcJLink.JLinkApplication` object. This method activates the application's static `Stop()` method.

27

Graphics

This chapter covers J-Link Graphics including displaying lists, displaying text and using the mouse.

Overview

The methods described in this section allow you to draw temporary graphics in a display window. Methods that are identified as 2D are used to draw entities (arcs, polygons, and text) in screen coordinates. Other entities may be drawn using the current model's coordinate system or the screen coordinate system's lines, circles, and polylines. Methods are also included for manipulating text properties and accessing mouse inputs.

Getting Mouse Input

The following methods are used to read the mouse position in screen coordinates with the mouse button depressed. Each method outputs the position and an enumerated type description of which mouse button was pressed when the mouse was at that position. These values are contained in the `pfcSession.MouseStatus`.

The enumerated values are defined in `pfcSession.MouseButton` and are as follows:

- `MOUSE_BTN_LEFT`
- `MOUSE_BTN_RIGHT`
- `MOUSE_BTN_MIDDLE`
- `MOUSE_BTN_LEFT_DOUBLECLICK`

Methods Introduced:

- **pfcSession.Session.UIGetNextMousePick**
- **pfcSession.Session.UIGetCurrentMouseStatus**

The method `pfcSession.Session.UIGetNextMousePick` returns the mouse position when you press a mouse button. The input argument is the mouse button that you expect the user to select.

The method `pfcSession.Session.UIGetCurrentMouseStatus` returns a value whenever the mouse is moved or a button is pressed. With this method a button does not have to be pressed for a value to be returned. You can use an input argument to flag whether or not the returned positions are snapped to the window grid.

Drawing a Mouse Box

This method allows you to draw a mouse box.

Method Introduced:

- **pfcSession.Session.UIPickMouseBox**

The method `pfcSession.Session.UIPickMouseBox` draws a dynamic rectangle from a specified point in screen coordinates to the current mouse position until the user presses the left mouse button. The return value for this method is of the type `pfcBase.Outline3D`.

You can supply the first corner location programmatically or you can allow the user to select both corners of the box.

Displaying Graphics

All the methods in this section draw graphics in the PTC Creo Parametric current window and use the color and linestyle set by calls to `pfcSession.BaseSession.SetStdColorFromRGB` and `pfcSession.BaseSession.SetLineStyle`. The methods draw the graphics in the PTC Creo Parametric graphics color. The default graphics color is white.

The methods in this section are called using the `pfcDisplay.Display`. This is extended by the `pfcSession.BaseSession` . This architecture allows you to call all these methods on any `Session` object.

By default graphic elements are not stored in the PTC Creo Parametric display list. Thus, they do not get redrawn by PTC Creo Parametric when the user selects **View, Repaint** or **View, Orientation**. However, if you store graphic elements in either 2-D or 3-D display lists, PTC Creo Parametric will redraw them when appropriate. See the section on Display Lists and Graphics on page for more information.

Methods Introduced:

- **pfcDisplay.Display.SetPenPosition**
- **pfcDisplay.Display.DrawLine**
- **pfcDisplay.Display.DrawPolyline**
- **pfcDisplay.Display.DrawCircle**
- **pfcDisplay.Display.DrawArc2D**
- **pfcDisplay.Display.DrawPolygon2D**

The method `pfcDisplay.Display.SetPenPosition` sets the point at which you want to start drawing a line. The function `pfcDisplay.Display.DrawLine` draws a line to the given point from the position given in the last call to either of the two functions. Call `pfcDisplay.Display.SetPenPosition` for the start of the polyline, and `pfcDisplay.Display.DrawLine` for each vertex. If you use these methods in two-dimensional modes, use screen coordinates instead of solid coordinates.

The method pfcDisplay.Display.DrawCircle uses solid coordinates for the center of the circle and the radius value. The circle will be placed to the XY plane of the model.

The method pfcDisplay.Display.DrawPolyline also draws polylines, using an array to define the polyline.

In two-dimensional models the Display Graphics methods draw graphics at the specified screen coordinates.

The method pfcDisplay.Display.DrawPolygon2D draws a polygon in screen coordinates. The method pfcDisplay.Display.DrawArc2D draws an arc in screen coordinates.

Controlling Graphics Display

MethodsIntroduced:

- **pfcDisplay.Display.GetCurrentGraphicsColor**
- **pfcDisplay.Display.SetCurrentGraphicsColor**
- **pfcDisplay.Display.GetCurrentGraphicsMode**
- **pfcDisplay.Display.SetCurrentGraphicsMode**

The method pfcDisplay.Display.GetCurrentGraphicsColor returns the PTC Creo Parametric standard color used to display graphics. The PTC Creo Parametric default is COLOR_DRAWING (white). The method pfcDisplay.Display.SetCurrentGraphicsColor allows you to change the color used to draw subsequent graphics.

The method pfcDisplay.Display.GetCurrentGraphicsMode returns the mode used to draw graphics:

- DRAW_GRAPHICS_NORMAL— PTC Creo Parametric draws graphics in the required color in each invocation.
- DRAW_GRAPHICS_COMPLEMENT—PTC Creo Parametric draws graphics normally, but will erase graphics drawn a second time in the same location. This allows you to create rubber band lines.

The method pfcDisplay.Display.GetCurrentGraphicsMode allows you to set the current graphics mode.

Example Code: Creating Graphics On Screen

The sample code in the file pfcDisplayExamples.java located at <creo_jlink_loadpoint>/jlink_appls/jlinkexamples demonstrates the use of mouse-tracking methods to draw graphics on the screen. The static method

DrawRubberbandLine prompts the user to pick a screen point. The example uses the 'complement mode' to cause the line to display and erase as the user moves the mouse around the window.

Note

This example uses the method transformPosition to convert the coordinates into the 3D coordinate system of a solid model, if one is displayed.

Display example text

%C PUSER Pick first location for rubberband line

Pick first location for rubberband line

#

#

Displaying Text in the Graphics Window

Method Introduced:

- **pfcDisplay.Display.DrawText2D**

The method pfcDisplay.Display.DrawText2D places text at a position specified in screen coordinates. If you want to add text to a particular position on the solid, you must transform the solid coordinates into screen coordinates by using the view matrix.

PTC Creo Parametric and therefore are not redrawn when you select **View, Repaint**. To notify thePTC Creo Parametric of these objects, create them inside the OnDisplay() method of the Display Listener.

Controlling Text Attributes

MethodsIntroduced:

- **pfcDisplay.Display.GetTextHeight**
- **pfcDisplay.Display.SetTextHeight**
- **pfcDisplay.Display.GetWidthFactor**
- **pfcDisplay.Display.SetWidthFactor**
- **pfcDisplay.Display.GetRotationAngle**
- **pfcDisplay.Display.SetRotationAngle**

- **pfcDisplay.Display.GetSlantAngle**
- **pfcDisplay.Display.SetSlantAngle**

These methods control the attributes of text added by calls to `pfcDisplay.Display.DrawText2D`.

You can get and set the following information:

- Text height (in screen coordinates)
- Width ratio of each character, including the gap, as a proportion of the height
- Rotation angle of the whole text, in counterclockwise degrees
- Slant angle of the text, in clockwise degrees

Controlling Text Fonts

Methods Introduced:

- **pfcDisplay.Display.GetDefaultFont**
- **pfcDisplay.Display.GetCurrentFont**
- **pfcDisplay.Display.SetCurrentFont**
- **pfcDisplay.Display.GetFontById**
- **pfcDisplay.Display.GetFontByName**

The method `pfcDisplay.Display.GetDefaultFont` returns the default PTC Creo Parametric text font. The text fonts are identified in PTC Creo Parametric by names and by integer identifiers. To find a specific font, use the methods `pfcDisplay.Display.GetFontById` or `pfcDisplay.Display.GetFontByName`.

28

External Data

This chapter explains using External Data in J-Link.

External Data

This chapter describes how to store and retrieve external data. External data enables a J-Link application to store its own data in a PTC Creo Parametric database in such a way that it is invisible to the PTC Creo Parametric user. This method is different from other means of storage accessible through the PTC Creo Parametric user interface.

Introduction to External Data

External data provides a way for the PTC Creo Parametric application to store its own private information about a PTC Creo Parametric model within the model file. The data is built and interrogated by the application as a workspace data structure. It is saved to the model file when the model is saved, and retrieved when the model is retrieved. The external data is otherwise ignored by PTC Creo Parametric; the application has complete control over form and content.

The external data for a specific PTC Creo Parametric model is broken down into classes and slots. A class is a named ‘‘bin’’ for your data, and identifies it as yours so no other PTC Creo Parametric API application (or other classes in your own application) will use it by mistake. An application usually needs only one class. The class name should be unique for each application and describe the role of the data in your application.

Each class contains a set of data slots. Each slot is identified by an identifier and optionally, a name. A slot contains a single data item of one of the following types:

J-Link Type	Data
`com.ptc.pfc.pfcExternal.EXTDATA_INTEGER`	integer
`com.ptc.pfc.pfcExternal.EXTDATA_DOUBLE`	double
`com.ptc.pfc.pfcExternal.EXTDATA_STRING`	string

The J-Link interfaces used to access external data in PTC Creo Parametric are:

J-Link Type	Data Type
`pfcExternal.ExternalDataAccess`	This is the top level object and is created when attempting to access external data.
`pfcExternal.ExternalDataClass`	This is a class of external data and is identified by a unique name.
`pfcExternal.ExternalDataSlot`	This is a container for one item of data. Each slot is stored in a class.
`pfcExternal.ExternalData`	This is a compact data structure that contains either an integer, double or string value.

Compatibility with PTC Creo Parametric TOOLKIT

J-Link and PTC Creo Parametric TOOLKIT share external data in the same manner. J-Link external data is accessible by PTC Creo Parametric TOOLKIT and the reverse is also true. However, an error will result if J-Link attempts to access external data previously stored by PTC Creo Parametric TOOLKIT as a stream.

Accessing External Data

Methods Introduced:

- **pfcModel.Model.AccessExternalData**
- **pfcModel.Model.TerminateExternalData**
- **pfcExternal.ExternalDataAccess.IsValid**

The method `pfcModel.Model.AccessExternalData` prepares PTC Creo Parametric to read external data from the model file. It returns the `pfcExternal.ExternalDataAccess` object that is used to read and write data. This method should be called only once for any given model in session.

The method `pfcModel.Model.TerminateExternalData` stops PTC Creo Parametric from accessing external data in a model. When you use this method all external data in the model will be removed. Permanent removal will occur when the model is saved.

Note

If you need to preserve the external data created in session, you must save the model before calling this function. Otherwise, your data will be lost.

The method `pfcExternal.ExternalDataAccess.IsValid` determines if the `ExternalDataAccess` object can be used to read and write data.

Storing External Data

Methods Introduced:

- **pfcExternal.ExternalDataAccess.CreateClass**
- **pfcExternal.ExternalDataClass.CreateSlot**
- **pfcExternal.ExternalDataSlot.SetValue**

The first step in storing external data in a new class and slot is to set up a class using the method `pfcExternal.ExternalDataAccess.CreateClass`, which provides the class name. The method outputs `pfcExternal.ExternalDataClass`, used by the application to reference the class.

The next step is to use `pfcExternal.ExternalDataClass.CreateSlot` to create an empty data slot and input a slot name. The method outputs a `pfcExternal.ExternalDataSlot` object to identify the new slot.

Note

Slot names cannot begin with a number.

The method `pfcExternal.ExternalDataSlot.SetValue` specifies the data type of a slot and writes an item of that type to the slot. The input is a `pfcExternal.ExternalData` object that you can create by calling any one of the methods in the next section.

Initializing Data Objects

Methods Introduced:

- **pfcExternal.pfcExternal.CreateIntExternalData**
- **pfcExternal.pfcExternal.CreateDoubleExternalData**
- **pfcExternal.pfcExternal.CreateStringExternalData**

These methods initialize a `pfcExternal.ExternalData` object with the appropriate data inputs.

Retrieving External Data

Methods Introduced:

- **pfcExternal.ExternalDataAccess.LoadAll**
- **pfcExternal.ExternalDataAccess.ListClasses**
- **pfcExternal.ExternalDataClass.ListSlots**
- **pfcExternal.ExternalDataSlot.GetValue**
- **pfcExternal.ExternalData.Getdiscr**
- **pfcExternal.ExternalData.GetIntegerValue**
- **pfcExternal.ExternalData.GetDoubleValue**
- **pfcExternal.ExternalData.GetStringValue**

For improved performance, external data is not loaded automatically into memory with the model. When the model is in session, call the method `pfcExternal.ExternalDataAccess.LoadAll` to retrieve all the external data for the specified model from the PTC Creo Parametric model file and put it in the workspace. The method needs to be called only once to retrieve all the data.

The method pfcExternal.ExternalDataAccess.ListClasses returns a sequence of ExternalDataClasses registered in the model. The method pfcExternal.ExternalDataClass.ListSlots provide a sequence of ExternalDataSlots existing for each class.

The method pfcExternal.ExternalDataSlot.GetValue reads the pfcExternal.ExternalData from a specified slot.

To find out a data type of a pfcExternal.ExternalData, call pfcExternal.ExternalData.Getdiscr and then call one of these methodsto get the data, depending on the data type:

- pfcExternal.ExternalData.GetIntegerValue
- pfcExternal.ExternalData.GetDoubleValue
- pfcExternal.ExternalData.GetStringValue

Example Code

The sample code in the file pfcExternalDataExamples.java located at <creo_jlink_loadpoint>/jlink_appls/jlinkexamples demonstrates the usage of external data in J-Link. It provides utility methods to convert a Java hashtable (java.util.Hashtable) to a model's external data, and to convert external data to a hashtable.

The conversion process makes some assumptions about the type of data to store in each data slot:

+ Short, Byte, Integer = integer external data

+ Float, Double = double external data

+ Any other Java object = String external data using the object's toString() method.

Exceptions

Most exceptions thrown by external data methods in J-Link extend pfcExceptions.XExternalDataError, which is a subclass of pfcExceptions.XToolkitError.

An additional exception thrown by external data methods is pfcExceptions.XBadExternalData. This exception signals an error accessing data. For example, external data access might have been terminated or the model might contain stream data from PTC Creo Parametric TOOLKIT.

The following table lists these exceptions.

Exception	Cause
XExternalDataInvalidObject	Generated when a model or class is invalid.
XExternalDataClassOrSlotExists	Generated when creating a class or slot and the proposed class or slot already exists.
XExternalDataNamesTooLong	Generated when a class or slot name is too long.
XExternalDataSlotNotFound	Generated when a specified class or slot does not exist.
XExternalDataEmptySlot	Generated when the slot you are attempting to read is empty.
XExternalDataInvalidSlotName	Generated when a specified slot name is invalid.
XBadGetExternalData	Generated when you try to access an incorrect data type in a External.ExternalData object.

29

PTC Windchill Connectivity APIs

PTC Creo Parametric has the capability to be directly connected to PTC Windchill solutions, including PTC WindchillProjectLink and PDMLink servers. This access allows users to manage and control the product data seamlessly from within PTC Creo Parametric.

This chapter lists J-Link APIs that support PTC Windchill servers and server operations in a connected PTC Creo Parametric session.

Introduction

The methods introduced in this chapter provide support for the basic PTC Windchill server operations from within PTC Creo Parametric. With these methods, operations such as registering a PTC Windchill server, managing workspaces, and check in or check out of objects will be possible via J-Link. The capabilities of these APIs are similar to the operations available from within the PTC Creo Parametric client, with some restrictions.

Some of these APIs are supported from a non-interactive, that is, batch mode application or asynchronous application.

Accessing a PTC Windchill Server from a PTC Creo Parametric Session

PTC Creo Parametric allows you to register PTC Windchill servers as a connection between the PTC Windchill database and PTC Creo Parametric. Although the represented PTC Windchill database can be from PTC WindchillProjectLink or PTC Windchill PDMLink all types of databases are represented in the same way.

You can use the following identifiers when referring to PTC Windchill servers in J-Link:

- Codebase URL—This is the root portion of the URL that is used to connect to a PTC Windchill server. For example `http://wcserver.company.com/Windchill`.
- Server Alias—A server alias is used to refer to the server after it has been registered. The alias is also used to construct paths to files in the server workspaces and commonspaces. The server alias is chosen by the user or application and it need not have any direct relationship to the codebase URL. An alias can be any normal name, such as `my_alias`.

Accessing Information Before Registering a Server

To start working with a PTC Windchill server, you must establish a connection by registering the server in PTC Creo Parametric. The methods described in this section allow you to connect to a PTC Windchill server and access information related to the server.

Methods Introduced:

- **pfcSession.BaseSession.AuthenticateBrowser**
- **pfcSession.BaseSession.GetServerLocation**
- **pfcServer.ServerLocation.GetClass**

- **pfcServer.ServerLocation.GetLocation**
- **pfcServer.ServerLocation.GetVersion**
- **pfcServer.ServerLocation.ListContexts**
- **pfcServer.ServerLocation.CollectWorkspaces**

Use the method `pfcSession.BaseSession.AuthenticateBrowser` to set the authentication context using a valid username and password. A successful call to this method allows the PTC Creo Parametric session to register with any server that accepts the username and password combination. A successful call to this method also ensures that an authentication dialog box does not appear during the registration process. You can call this method any number of times to set the authentication context for any number of PTC Windchill servers, provided that you register the appropriate servers or servers immediately after setting the context.

The method `pfcServer.ServerLocation.GetLocation` returns a `pfcServer.ServerLocation` object representing the codebase URL for a possible server. The server may not have been registered yet, but you can use this object and the methods it contains to gather information about the server prior to registration.

The method `pfcServer.ServerLocation.GetClass` returns the class of the server or server location. The values are:

- `Windchill`—Denotes a PTC Windchill PDMLink server.
- `ProjectLink`—Denotes PTC Windchill ProjectLink type of servers.

The method `pfcServer.ServerLocation.GetVersion` returns the version of PTC Windchill that is configured on the server or server location, for example, `9.0` or `10.0`. This method accepts the server codebase URL as the input.

Note

`pfcServer.ServerLocation.GetVersion` works only for PTC Windchill servers and throws the `pfcExceptions.XToolkitUnsupported` exception, if the server is not a PTC Windchill server.

The method `pfcServer.ServerLocation.ListContexts` gives a list of all the available contexts for a specified server. A context is used to associate a workspace with a product, project, or library.

The method `pfcServer.ServerLocation.CollectWorkspaces` returns the list of available workspaces for the specified server. The workspace objects returned contain the name of each workspace and its context.

Registering and Activating a Server

From Creo Parametric 2.0 onward, the J-Link methods call the same underlying API as PTC Creo Parametric to register and unregister servers. Hence, registering the servers using J-Link methods is similar to registering the servers using the PTC Creo Parametric user interface. Therefore, the servers registered by J-Link are available in the PTC Creo Parametric Server Registry. The servers are also available in other locations in the PTC Creo Parametric user interface such as, the **Folder Navigator** and the embedded browser.

Methods Introduced:

- **pfcSession.BaseSession.RegisterServer**
- **pfcServer.Server.Activate**
- **pfcServer.Server.Unregister**

The method `pfcSession.BaseSession.RegisterServer` registers the specified server with the codebase URL. You can automate the registration of servers in interactive mode. To preregister the servers use the standard `config.fld` setup. If you do not want the servers to be preregistered in batch mode, set the environment variable `PTC_WF_ROOT` to an empty directory before starting PTC Creo Parametric.

A successful call to `pfcSession.BaseSession.AuthenticateBrowser` with a valid username and password is essential for `pfcSession.BaseSession.RegisterServer` to register the server without launching the authentication dialog box. Registration of the server establishes the server alias. You must designate an existing workspace to use when registering the server. After the server has been registered, you may create a new workspace.

The method `pfcServer.Server.Activate` sets the specified server as the active server in the PTC Creo Parametric session.

The method `pfcServer.Server.Unregister` unregisters the specified server. This is similar to **Server Registry** ▸ **Delete** through the user interface.

Accessing Information From a Registered Server

Methods Introduced:

- **pfcServer.Server.GetIsActive**
- **pfcServer.Server.GetAlias**
- **pfcServer.Server.GetContext**

The method `pfcServer.Server.GetIsActive` specifies if the server is active.

The method `pfcServer.Server.GetAlias` returns the alias of a server if you specify the codebase URL.

The method `pfcServer.Server.GetContext` returns the active context of the active server.

Information on Servers in Session

Methods Introduced:

- **pfcSession.BaseSession.GetActiveServer**
- **pfcSession.BaseSession.GetServerByAlias**
- **pfcSession.BaseSession.GetServerByUrl**
- **pfcSession.BaseSession.ListServers**

The method `pfcSession.BaseSession.GetActiveServer` returns returns the active server handle.

The method `pfcSession.BaseSession.GetServerByAlias` returns the handle to the server matching the given server alias, if it exists in session.

The method `pfcSession.BaseSession.GetServerByUrl` returns the handle to the server matching the given server URL and workspace name, if it exists in session.

The method `pfcSession.BaseSession.ListServers` returns a list of servers registered in this session.

Accessing Workspaces

For every workspace, a new distinct storage location is maintained in the user's personal folder on the server (server-side workspace) and on the client (client-side workspace cache). Together, the server-side workspace and the client-side workspace cache make up the workspace.

Methods Introduced:

- **pfcServer.pfcServer.WorkspaceDefinition_Create**
- **pfcServer.WorkspaceDefinition.GetWorkspaceName**
- **pfcServer.WorkspaceDefinition.GetWorkspaceContext**
- **pfcServer.WorkspaceDefinition.SetWorkspaceName**
- **pfcServer.WorkspaceDefinition.SetWorkspaceContext**

The interface `pfcServer.WorkspaceDefinition` contains the name and context of the workspace. The method `pfcServer.ServerLocation.CollectWorkspaces` returns an array of workspace data. Workspace data is also required for the method `pfcServer.Server.CreateWorkspace` to create a workspace with a given name and a specific context.

The method `pfcServer.pfcServer.WorkspaceDefinition_Create` creates a new workspace definition object suitable for use when creating a new workspace on the server.

The method `pfcServer.WorkspaceDefinition.GetWorkspaceName` retrieves the name of the workspace.

The method `pfcServer.WorkspaceDefinition.GetWorkspaceContext` retrieves the context of the workspace.

The method `pfcServer.WorkspaceDefinition.SetWorkspaceName` sets the name of the workspace.

The method `pfcServer.WorkspaceDefinition.SetWorkspaceContext` sets the context of the workspace.

Creating and Modifying the Workspace

Methods Introduced:

- **pfcServer.Server.CreateWorkspace**
- **pfcServer.Server.GetActiveWorkspace**
- **pfcServer.Server.SetActiveWorkspace**
- **pfcServer.ServerLocation.DeleteWorkspace**

The method `pfcServer.Server.CreateWorkspace` creates and activates a new workspace.

The method `pfcServer.Server.GetActiveWorkspace` retrieves the name of the active workspace.

The method `pfcServer.Server.SetActiveWorkspace` sets a specified workspace as an active workspace.

The method `pfcServer.ServerLocation.DeleteWorkspace` deletes the specified workspace. This function is available only in the non-interactive mode, that is, in batch mode. The method deletes the workspace only if the following conditions are met:

- The workspace is not the active workspace.
- The workspace does not contain any checked out objects.

Use one of the following techniques to delete an active workspace:

- Make the required workspace inactive using `pfcServer.Server.SetActiveWorkspace` with the name of some other workspace and then call `pfcServer.ServerLocation.DeleteWorkspace`.
- Unregister the server using `pfcServer.Server.Unregister` and delete the workspace.

Workflow to Register a Server

To Register a Server with an Existing Workspace

Perform the following steps to register a PTC Windchill server with an existing workspace:

1. Set the appropriate authentication context using the method `pfcSession.BaseSession.AuthenticateBrowser` with a valid username and password.
2. Look up the list of workspaces using the method `pfcServer.ServerLocation.CollectWorkspaces`. If you already know the name of the workspace on the server, then ignore this step.
3. Register the workspace using the method `pfcSession.BaseSession.RegisterServer` with an existing workspace name on the server.
4. Activate the server using the method `pfcServer.Server.Activate`.

To Register a Server with a New Workspace

Perform the following steps to register a PTC Windchill server with a new workspace:

1. Perform steps 1 to 4 in the preceding section to register the PTC Windchill server with an existing workspace.
2. Use the method `pfcServer.ServerLocation.ListContexts` to choose the required context for the server.
3. Create a new workspace with the required context using the method `pfcServer.Server.CreateWorkspace`. This method automatically makes the created workspace active.

Note

You can create a workspace only after the server is registered.

Aliased URL

An aliased URL serves as a handle to the server objects. You can access the server objects in the commonspace (shared folders) and the workspace using an aliased URL. An aliased URL is a unique identifier for the server object and its format is as follows:

- Object in workspace has a prefix `wtws`
  `wtws://<server_alias>/<workspace_name>/<object_server_name>`

  where `<object_server_name>` includes `<object_name>.<object_extension>`

  For example,
  `wtws://my_server/my_workspace/abcd.prt`,
  `wtws://my_server/my_workspace/intf_file.igs`

  where

  `<server_alias>` is `my_server`

  `<workspace_name>` is `my_workspace`
- Object in commonspace has a prefix `wtpub`
  `wtpub://<server_alias>/<folder_location>/<object_server_name>`

  For example,
  `wtpub://my_server/path/to/cs_folder/abcd.prt`

  where

  `<server_alias>` is `my_server`

  `<folder_location>` is `path/to/cs_folder`

Note

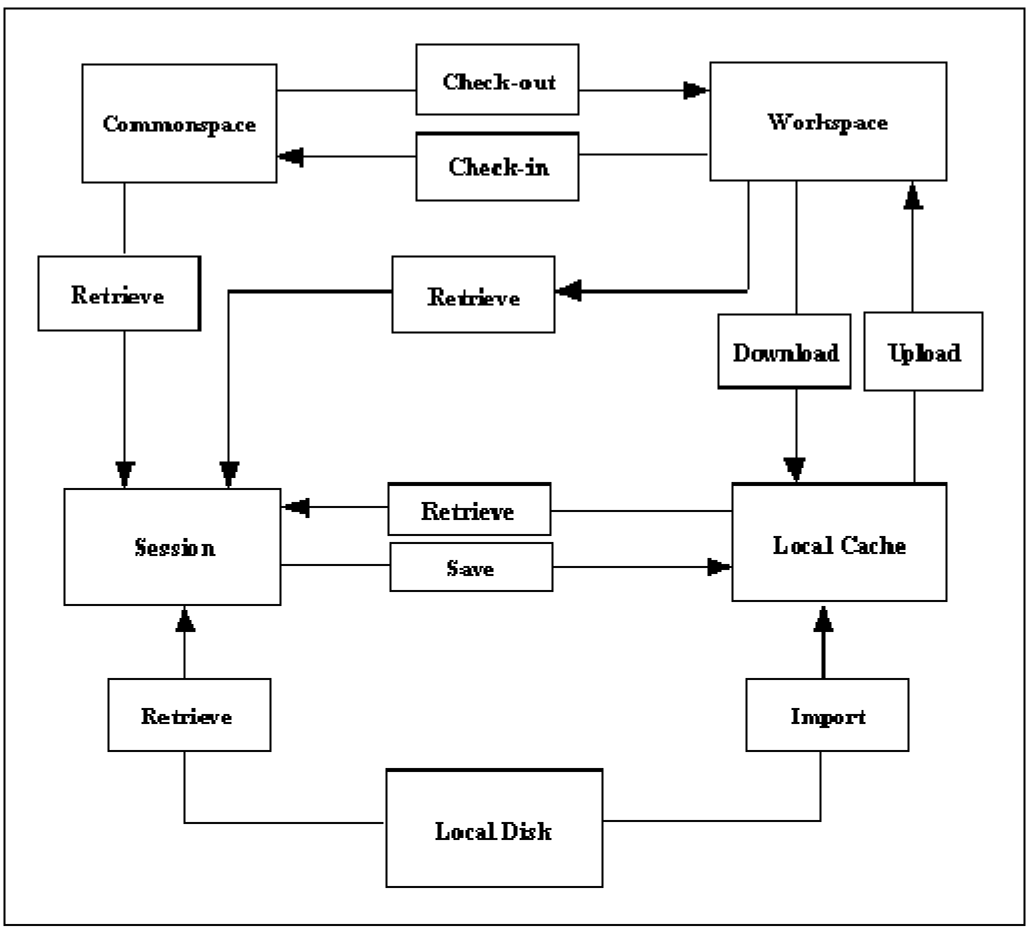
- `object_server_name` must be in lowercase.
- The APIs are case-sensitive to the aliased URL.
- `<object_extension>` should not contain PTC Creo Parametric versions, for example, .1 or .2, and so on.

Server Operations

After registering the PTC Windchill server with PTC Creo Parametric, you can start accessing the data on the PTC Windchill servers. The PTC Creo Parametric interaction with PTC Windchill servers leverages the following locations:

- Commonspace (Shared folders)
- Workspace (Server-side workspace)
- Workspace local cache (Client-side workspace)
- PTC Creo Parametric session
- Local disk

The methods described in this section enable you to perform the basic server operations. The following illustration shows how data is transferred among these locations.



Save

Methods Introduced:

- **pfcModel.Model.Save**

The method `pfcModel.Model.Save` stores the object from the session in the local workspace cache, when a server is active.

Upload

An upload transfers PTC Creo Parametric files and any other dependencies from the local workspace cache to the server-side workspace.

Methods Introduced:

- **pfcServer.Server.UploadObjects**
- **pfcServer.Server.UploadObjectsWithOptions**
- **pfcServer.pfcServer.UploadOptions_Create**

The method `pfcServer.Server.UploadObjects` uploads the object to the workspace. The object to be uploaded must be present in the current PTC Creo Parametric session. You must save the object to the workspace using `pfcModel.Model.SavepfcSession.BaseSession.ImportToCurrentWS` before attempting to upload it.

The method `pfcServer.Server.UploadObjectsWithOptions` uploads objects to the workspace using the options specified in the `pfcServer.UploadOptions` . These options allow you to upload the entire workspace, auto-resolve missing references, and indicate the target folder location for the new content during the upload. You must save the object to the workspace using `pfcModel.Model.Save`, or import it to the workspace using `pfcSession.BaseSession.ImportToCurrentWS` before attempting to upload it.

Create the `pfcServer.UploadOptions` object using the method `pfcServer.pfcServer.UploadOptions_Create`.

The methods available for setting the upload options are described in the following section.

CheckIn

After you have finished working on objects in your workspace, you can share the design changes with other users. The checkin operation copies the information and files associated with all changed objects from the workspace to the PTC Windchill database.

Methods Introduced:

- **pfcServer.Server.CheckinObjects**
- **pfcServer.pfcServer.CheckinOptions_Create**
- **pfcServer.UploadBaseOptions.SetDefaultFolder**

- **pfcServer.UploadBaseOptions.SetNonDefaultFolderAssignments**
- **pfcServer.UploadBaseOptions.SetAutoresolveOption**
- **pfcServer.CheckinOptions.SetBaselineName**
- **pfcServer.CheckinOptions.SetBaselineNumber**
- **pfcServer.CheckinOptions.SetBaselineLocation**
- **pfcServer.CheckinOptions.GetBaselineLifecycle**
- **pfcServer.CheckinOptions.SetKeepCheckedout**

The method `pfcServer.Server.CheckinObjects` checks in an object into the database. The object to be checked in must be present in the current PTC Creo Parametric session. Changes made to the object are not included unless you save the object to the workspace using the method `pfcModel.Model.Save` before you check it in.

If you pass `NULL` as the value of the *options* parameter, the checkin operation is similar to the **Auto Check-In** option in PTC Creo Parametric. For more details on **Auto Check-In**, refer to the online help for PTC Creo Parametric.

Use the method `pfcServer.pfcServer.CheckinOptions_Create` to create a new `CheckinOptions` object.

By using an appropriately constructed *options* argument, you can control the checkin operation. Use the APIs listed above to access and modify the checkin options. The checkin options are as follows:

- *DefaultFolder*—Specifies the default folder location on the server for the automatic checkin operation.
- *NonDefaultFolderAssignment*—Specifies the folder location on the server to which the objects will be checked in.
- *AutoresolveOption*—Specifies the option used for auto-resolving missing references. These options are defined in the `ServerAutoresolveOption` class, and are as follows:
  - `SERVER_DONT_AUTORESOLVE`—Model references missing from the workspace are not automatically resolved. This may result in a conflict upon checkin. This option is used by default.
  - `SERVER_AUTORESOLVE_IGNORE`—Missing references are automatically resolved by ignoring them.
  - `SERVER_AUTORESOLVE_UPDATE_IGNORE`—Missing references are automatically resolved by updating them in the database and ignoring them if not found.
- *Baseline*—Specifies the baseline information for the objects upon checkin. The baseline information for a checkin operation is as follows:

- *BaselineName*—Specifies the name of the baseline.
- *BaselineNumber*—Specifies the number of the baseline.

The default format for the baseline name and baseline number is `Username + time (GMT) in milliseconds`.

- *BaselineLocation*—Specifies the location of the baseline.
- *BaselineLifecycle*—Specifies the name of the lifecycle.

- *KeepCheckedout*—If the value specified is `true`, then the contents of the selected object are checked into the PTC Windchill server and automatically checked out again for further modification.

Retrieval

Standard J-Link provides several methods that are capable of retrieving models. When using these methods with PTC Windchill servers, remember that these methods do not check out the object to allow modifications.

Methods Introduced:

- **pfcSession.BaseSession.RetrieveModel**
- **pfcSession.BaseSession.RetrieveModelWithOpts**
- **pfcSession.BaseSession.OpenFile**

The methods `pfcSession.BaseSession.RetrieveModel`, `pfcSession.BaseSession.RetrieveModelWithOpts`, and `pfcSession.BaseSession.OpenFile` load an object into a session given its name and type. The methods search for the object in the active workspace, the local directory, and any other paths specified by the `search_path` configuration option.

Checkout and Download

To modify an object from the commonspace, you must check out the object. The process of checking out communicates your intention to modify a design to the PTC Windchill server. The object in the database is locked, so that other users can obtain read-only copies of the object, and are prevented from modifying the object while you have checked it out.

Checkout is often accompanied by a download action, where the objects are brought from the server-side workspace to the local workspace cache. In J-Link, both operations are covered by the same set of methods.

Methods Introduced:

- **pfcServer.Server.CheckoutObjects**
- **pfcServer.Server.CheckoutMultipleObjects**

- **pfcServer.pfcServer.CheckoutOptions_Create**
- **pfcServer.CheckoutOptions.SetDependency**
- **pfcServer.CheckoutOptions.SetSelectedIncludes**
- **pfcServer.CheckoutOptions.SetIncludeInstances**
- **pfcServer.CheckoutOptions.SetVersion**
- **pfcServer.CheckoutOptions.SetDownload**
- **pfcServer.CheckoutOptions.SetReadonly**

The method `pfcServer.Server.CheckoutObjects` checks out and optionally downloads the object to the workspace based on the configuration specifications of the workspace. The input arguments of this method are as follows:

- *Mdl*—Specifies the object to be checked out. This is applicable if the model has already been retrieved without checking it out.
- *File*—Specifies the top-level object to be checked out.
- *Checkout*—The checkout flag. If you specify the value of this argument as `true`, the selected object is checked out. Otherwise, the object is downloaded without being checked out. The download action enables you to bring read-only copies of objects into your workspace. This allows you to examine the object without locking it.
- *Options*—Specifies the checkout options object. If you pass `NULL` as the value of this argument,, then the default PTC Creo Parametric checkout rules apply. Use the method `pfcServer.pfcServer.CheckoutOptions_Create` to create a new `CheckoutOptions` object.

Use the method `pfcServer.Server.CheckoutMultipleObjects` to check out and download multiple objects to the workspace based on the configuration specifications of the workspace. This method takes the same input arguments as listed above, except for *Mdl* and *File*. Instead it takes the argument *Files* that specifies the sequence of the objects to check out or download.

By using an appropriately constructed *options* argument in the above functions, you can control the checkout operation. Use the APIs listed above to modify the checkout options. The checkout options are as follows:

- *Dependency*—Specifies the dependency rule used while checking out dependents of the object selected for checkout. The types of dependencies given by the `ServerDependency` class are as follows:
    - `SERVER_DEPENDENCY_ALL`—All the objects that are dependent on the selected object are downloaded, that is, they are added to the workspace.

- SERVER_DEPENDENCY_REQUIRED—All the objects that are required to successfully retrieve the selected object in the CAD application are downloaded, that is, they are added to workspace.
- SERVER_DEPENDENCY_NONE—None of the dependent objects from the selected object are downloaded, that is, they are not added to workspace.

- *IncludeInstances*—Specifies the rule for including instances from the family table during checkout. The type of instances given by the ServerIncludeInstances class are as follows:
  - SERVER_INCLUDE_ALL—All the instances of the selected object are checked out.
  - SERVER_INCLUDE_SELECTED—The application can select the family table instance members to be included during checkout.
  - SERVER_INCLUDE_NONE—No additional instances from the family table are added to the object list.
- *SelectedIncludes*—Specifies the sequence of URLs to the selected instances, if *IncludeInstances* is of type SERVER_INCLUDE_SELECTED.
- *Version*—Specifies the version of the checked out object. If this value is set to NULL, the object is checked out according to the current workspace configuration.
- *Download*—Specifies the checkout type as download or link. The value download specifies that the object content is downloaded and checked out, while link specifies that only the metadata is downloaded and checked out.
- *Readonly*—Specifies the checkout type as a read-only checkout. This option is applicable only if the checkout type is link.

The following truth table explains the dependencies of the different control factors in the method pfcServer.Server.CheckoutObjects and the effect of different combinations on the end result.

Argument *checkout* in pfcServer.Server.CheckoutObjects	**pfcServer.CheckoutOptions.SetDownload**	**pfcServer.CheckoutOptions.SetReadonly**	**Result**
true	true	NA	Object is checked out and its content is downloaded.
true	false	NA	Object is checked out but content is not downloaded.
false	NA	true	Object is downloaded without checkout and as read-only.
false	NA	false	Not supported

Undo Checkout

Method Introduced:

- **pfcServer.Server.UndoCheckout**

Use the method `pfcServer.Server.UndoCheckout` to undo a checkout of the specified object. When you undo a checkout, the changes that you have made to the content and metadata of the object are discarded and the content, as stored in the server, is downloaded to the workspace. This method is applicable only for the model in the active PTC Creo Parametric session.

Import and Export

J-Link provides you with the capability of transferring specified objects to and from a workspace. Import and export operations must take place in a session with no models. An import operation transfers a file from the local disk to the workspace.

Methods Introduced:

- **pfcSession.BaseSession.ExportFromCurrentWS**
- **pfcSession.BaseSession.ImportToCurrentWS**
- **pfcSession.WSImportExportMessage.GetDescription**
- **pfcSession.WSImportExportMessage.GetFileName**
- **pfcSession.WSImportExportMessage.GetMessageType**
- **pfcSession.WSImportExportMessage.GetResolution**
- **pfcSession.WSImportExportMessage.GetSucceeded**
- **pfcSession.BaseSession.SetWSExportOptions**
- **pfcSession.pfcSession.WSExportOptions_Create**
- **pfcSession.WSExportOptions.SetIncludeSecondaryContent**

The method `pfcSession.BaseSession.ExportFromCurrentWS` exports the specified objects from the current workspace to a disk in a linked session of PTC Creo Parametric.

The method `pfcSession.BaseSession.ImportToCurrentWS` imports the specified objects from a disk to the current workspace in a linked session of PTC Creo Parametric.

Both `pfcSession.BaseSession.ExportFromCurrentWS` and `pfcSession.BaseSession.ImportToCurrentWS` allow you to specify a dependency criterion to process the following items:

- All external dependencies
- Only required dependencies
- No external dependencies

Both pfcSession.BaseSession.ExportFromCurrentWS and pfcSession.BaseSession.ImportToCurrentWS return the messages generated during the export or import operation in the form of the pfcSession.WSImportExportMessages object. Use the APIs listed above to access the contents of a message. The message specified by the pfcSession.WSImportExportMessage object contains the following items:

- Description—Specifies the description of the problem or the message information.
- FileName—Specifies the object name or the name of the object path.
- MessageType—Specifies the severity of the message in the form of the WSImportExportMessageType class. The severity is one of the following types:
  - WSIMPEX_MSG_INFO—Specifies an informational type of message.
  - WSIMPEX_MSG_WARNING—Specifies a low severity problem that can be resolved according to the configured rules.
  - WSIMPEX_MSG_CONFLICT—Specifies a conflict that can be overridden.
  - WSIMPEX_MSG_ERROR—Specifies a conflict that cannot be overridden or a serious problem that prevents processing of an object.
- Resolution—Specifies the resolution applied to resolve a conflict that can be overridden. This is applicable when the message is of the type WSIMPEX_MSG_CONFLICT.
- Succeeded—Determines whether the resolution succeeded or not. This is applicable when the message is of the type WSIMPEX_MSG_CONFLICT.

The method pfcSession.BaseSession.SetWSExportOptions sets the export options used while exporting the objects from a workspace in the form of the pfcSession.WSExportOptions object. Create this object using the method pfcSession.pfcSession.WSExportOptions_Create. The export options are as follows:

- *Include Secondary Content*—Indicates whether or not to include secondary content while exporting the primary PTC Creo Parametric model files. Use the method pfcSession.WSExportOptions.SetIncludeSecondaryContent to set this option.

File Copy

J-Link provides you with the capability of copying a file from the workspace or target folder to a location on the disk and vice-versa.

Methods Introduced:

- **pfcSession.BaseSession.CopyFileToWS**
- **pfcSession.BaseSession.CopyFileFromWS**

Use the method pfcSession.BaseSession.CopyFileToWS to copy a file from the disk to the workspace. The file can optionally be added as secondary content to a given workspace file. If the viewable file is added as secondary content, a dependency is created between the PTC Creo Parametric model and the viewable file.

Use the method pfcSession.BaseSession.CopyFileFromWS to copy a file from the workspace to a location on disk.

When importing or exporting PTC Creo Parametric models, PTC recommends that you use methods pfcSession.BaseSession.ImportToCurrentWS and pfcSession.BaseSession.ExportFromCurrentWS, respectively, to perform the import or export operation. Methods that copy individual files do not traverse PTC Creo Parametric model dependencies, and therefore do not copy a fully retrievable set of models at the same time.

Additionally, only the methods pfcSession.BaseSession.ImportToCurrentWS and pfcSession.BaseSession.ExportFromCurrentWS provide full metadata exchange and support. That means pfcSession.BaseSession.ImportToCurrentWS can communicate all the PTC Creo Parametric designated parameters, dependencies, and family table information to a PDM system while pfcSession.BaseSession.ExportFromCurrentWS can update exported PTC Creo Parametric data with PDM system changes to designated and system parameters, dependencies, and family table information. Hence PTC recommends the use of pfcSession.BaseSession.CopyFileToWS and pfcSession.BaseSession.CopyFileFromWS to process only non-PTC Creo Parametric files.

Server Object Status

Methods Introduced:

- **pfcServer.Server.IsObjectCheckedOut**
- **pfcServer.Server.IsObjectModified**

The methods described in this section verify the current status of the object in the workspace. The method `pfcServer.Server.IsObjectCheckedOut` specifies whether the object is checked out for modification. The value `true` indicates that the specified object is checked out to the active workspace.

The value `false` indicates one of the following statuses:

- The specified object is not checked out
- The specified object is only uploaded to the workspace, but was never checked in
- The specified object is only saved to the local workspace cache, but was never uploaded

The method `pfcServer.Server.IsObjectModified` specifies whether the object has been modified in the workspace. This method returns `true` if the object was modified locally.

Delete Objects

Method Introduced:

- **pfcServer.Server.RemoveObjects**

The method `pfcServer.Server.RemoveObjects` deletes the array of objects from the workspace. When passed with the *ModelNames* array as `NULL`, this method removes all the objects in the active workspace.

Conflicts During Server Operations

Method Introduced:

- **pfcExceptions.XToolkitCheckoutConflict.GetConflictDescription**

An exception is provided to capture the error condition while performing the following server operations using the specified APIs:

Operation	API
Checkin an object or workspace	`pfcServer.Server.CheckinObjects`
Checkout an object	`pfcServer.Server.CheckoutObjects`
Undo checkout of an object	`pfcServer.Server.UndoCheckout`
Upload object	`pfcServer.Server.UploadObjects`
Download object	`pfcServer.Server.CheckoutObjects` (with download as `true`)
Delete workspace	`pfcServer.ServerLocation.DeleteWorkspace`
Remove object	`pfcServer.Server.RemoveObjects`

These APIs throw a common exception XToolkitCheckoutConflict if an error is encountered during server operations. Use the method pfcExceptions.XToolkitCheckoutConflict.GetConflictDescription to extract details of the error condition. This description is similar to the description displayed by the PTC Creo Parametric HTML user interface in the conflict report.

Utility APIs

The methods specified in this section enable you to obtain the handle to the server objects to access them. The handle may be the aliased URL or the model name of the http URL. These utilities enable the conversion of one type of handle to another.

Methods Introduced:

- **pfcServer.Server.GetAliasedUrl**
- **pfcSession.BaseSession.GetModelNameFromAliasedUrl**
- **pfcSession.BaseSession.GetAliasFromAliasedUrl**
- **pfcSession.BaseSession.GetUrlFromAliasedUrl**

The method pfcServer.Server.GetAliasedUrl enables you to search for a server object by its name. Specify the complete filename of the object as the input, for example, test_part.prt. The method returns the aliased URL for a model on the server. For more information regarding the aliased URL, refer to the section Aliased URL on page 392. During the search operation, the workspace takes precedence over the shared space.

You can also use this method to search for files that are not in the PTC Creo Parametric format. For example, my_text.txt, intf_file.stp, and so on.

The method pfcSession.BaseSession.GetModelNameFromAliasedUrl returns the name of the object from the given aliased URL on the server.

The method pfcSession.BaseSession.GetUrlFromAliasedUrl converts an aliased URL to a standard URL for the objects on the server.

For example, wtws://my_alias/Creo Parametric/abcd.prt is converted to an appropriate URL on the server as http://server.mycompany.com/Windchill.

The method pfcSession.BaseSession.GetAliasFromAliasedUrl returns the server alias from aliased URL.

A

Summary of Technical Changes

This chapter describes the critical and miscellaneous technical changes in PTC Creo Parametric 3.0 and J-Link. It also lists the new and superseded functions for this release.

Critical Technical Changes

This section describes the changes in PTC Creo Parametric 3.0 and J-Link that might require alteration of existing J-Link applications.

Accessing Member Information in a Pattern of Pattern

From Creo Parametric 2.0 M170, the method `pfcFeature.FeaturePattern.ListMembers` returns all the pattern header features created at the first level for a pattern of pattern. It does not return the pattern members under each pattern header.

APIWizard Available on PTC.com

The latest version of J-Link APIWizard is available at www.ptc.com/support/apiwizard.htm.

File Name of Applet Based APIWizard

From J-Link 3.0 M060 onward, the file name of applet based J-Link APIWizard is changed from `index.html` to `index_jlink.html`. To open the applet based APIWizard point your browser to:

`<creo_jlink_loadpoint>\jlinkdoc\index_jlink.html`

No Support for Applet Based APIWizard

The applet based APIWizard for J-Link will not be available in future releases. PTC recommends using the non-applet based version of the APIWizard.

Non-Applet Based Version of the APIWizard

To open the non-applet based version of the J-Link APIWizard, point your browser to:

`<creo_jlink_loadpoint>\jlinkdoc\manual0\loadToolkitDoc.html`

The non-applet based version of the J-Link APIWizard has enhanced search capabilities. The new search options enable you to search for only global methods and exceptions, in addition to searching for all classes and methods.

Change in Behavior of pfcServer.Server.IsObjectModified

From Creo Parametric 2.0 M040 onward, the behavior of the method `pfcServer.Server.IsObjectModified` has been fixed. The method now returns the value `false` to indicate one of the following statuses for the specified object:

- the object was only saved, but never uploaded
- the object was only uploaded, but never checked in

Change in Behavior of pfcTable.Table.CheckIfIsFromFormat

The method `pfcTable.Table.CheckIfIsFromFormat` did not correctly check if the drawing table was created using the drawing format. The method would return `FALSE`, if the first segment of the table was not on the current sheet in the Creo Parametric user interface, even though the table was created using the drawing format. This behavior has been fixed in Creo Parametric 2.0 M120. The method now checks and returns the correct boolean value.

As in the previous releases, the method `pfcTable.Table.CheckIfIsFromFormat` ignores the value provided in the input argument `SheetNumber`.

Change in Directory Structure for PTC Creo Installation

From PTC Creo 3.0 onward, the directory structure for PTC Creo installation has changed from:

- <creo_loadpoint>\Parametric to <creo_loadpoint>\<datecode>\Parametric.
- <creo_loadpoint>\Common Files\<datecode> to <creo_loadpoint>\<datecode>\Common Files

Change in Integer Values for Enumerated Data Type ComponentType

The integer values for enumerated data type `ComponentType` were shifted by 1 and did not reflect the actual values. From Creo Parametric 2.0 2.0, M120 onward, this behavior has been fixed. If your application code uses these values, you must rebuild your application.

Digital Rights Management Retired

From PTC Creo Parametric 3.0 onward, Digital Rights Management (DRM) is no longer supported. J-Link applications that check the DRM permissions will have to be updated.

Disable Notification Messages in Trail Files

When notifications are set in J-Link applications, every time an event is triggered, notification messages are added to the trail files.

From Creo Parametric 2.0 M210 onward, a new environment variable `PROTK_LOG_DISABLE` allows you to disable this behavior. When set to `true`, the notifications messages are not added to the trail files.

Documentation Updated for pfcServer.Server.IsObjectModified

The documentation for the method `pfcServer.Server.IsObjectModified` has been updated in PTC Creo 3.0 M010. This method returns `true` if the object was modified locally.

Layout Model Type

From PTC Creo 3.0 M010 onward, the object `pfcModel.ModelType` contains an additional value `MDL_CE_SOLID` that represents models of type Layout. J-Link methods will only be able to read models of type Layout, but will not be able to pass Layout models as input to other methods. To work with Layout models, you must rebuild your existing J-Link applications.

No Support for Boundary Box Type of Simplified Representation

From PTC Creo Parametric 3.0 onward, the boundary box type of representation specified by the following values are no longer supported:

- `SIMPREP_BOUNDBOX_REP` in the enumerated data type `SimpRepType`
- `SIMPREP_BOUNDBOX` in the enumerated data type `SimpRepActionType`

Retrieving Solids in a PTC Creo Parametric Session Linked to PTC Windchill

You must retrieve solid models in a PTC Creo Parametric session, which is linked to PTC Windchill, only after you create a new workspace. If you retrieve the models before creating the workspace, the models would be erased from the PTC Creo Parametric session.

Support for Constraint Creation Methods

You cannot create constraints using the J-Link applications. The following methods are not supported. These methods will be supported in a future release:

- `pfcDatumPlaneFeat.pfcDatumPlaneFeat.DatumPlaneThroughConstraint_Create`
- `pfcDatumPlaneFeat.pfcDatumPlaneFeat.DatumPlaneNormalConstraint_Create`
- `pfcDatumPlaneFeat.pfcDatumPlaneFeat.DatumPlaneParallelConstraint_Create`
- `pfcDatumPlaneFeat.pfcDatumPlaneFeat.DatumPlaneTangentConstraint_Create`
- `pfcDatumPlaneFeat.pfcDatumPlaneFeat.DatumPlaneOffsetConstraint_Create`
- `pfcDatumPlaneFeat.pfcDatumPlaneFeat.DatumPlaneOffsetCoordSysConstraint_Create`
- `pfcDatumPlaneFeat.pfcDatumPlaneFeat.DatumPlaneAngleConstraint_Create`
- `pfcDatumPlaneFeat.pfcDatumPlaneFeat.DatumPlaneSectionConstraint_Create`
- `pfcDatumPlaneFeat.pfcDatumPlaneFeat.DatumPlaneDefaultXConstraint_Create`
- `pfcDatumPlaneFeat.pfcDatumPlaneFeat.DatumPlaneDefaultYConstraint_Create`
- `pfcDatumPlaneFeat.pfcDatumPlaneFeat.DatumPlaneDefaultZConstraint_Create`
- `pfcDatumAxisFeat.pfcDatumAxisFeat.DatumAxisConstraint_Create`
- `pfcDatumAxisFeat.pfcDatumAxisFeat.DatumAxisDimensionConstraint_Create`
- `pfcDatumPointFeat.pfcDatumPointFeat.DatumPointPlacementConstraint_Create`

- pfcDatumPointFeat.pfcDatumPointFeat.DatumPointDimensionConstraint_Create
- pfcCoordSysFeat.pfcCoordSysFeat.DatumCsysOriginConstraint_Create
- pfcCoordSysFeat.pfcCoordSysFeat.DatumCsysDimensionConstraint_Create
- pfcCoordSysFeat.pfcCoordSysFeat.DatumCsysOrientMoveConstraint_Create

Support for Feature Subclasses

From PTC Creo Parametric 3.0 onward, only the following subclasses of the class pfcFeature are supported:

- pfcComponentFeat
- pfcCoordSysFeat
- pfcCurveFeat
- pfcDatumAxisFeat
- pfcDatumPlaneFeat
- pfcDatumPointFeat
- pfcRoundFeat

If your applications check the type of feature using the type of class, then you must update your existing code to use the method pfcFeature.Feature.GetFeatType. The method pfcFeature.Feature.GetFeatType returns the type of feature.

Support for Multi-CAD Assemblies

Multi-CAD assemblies are not supported in J-Link. The methods provided in PTC Creo Object TOOLKIT Java support multi-CAD assemblies. Refer to *PTC Creo Object TOOLKIT Java User's Guide* for information on multi-CAD assemblies.

New Functions

This section describes new functions for J-Link for PTC Creo Parametric 3.0.

Drawings

New Function	Description
pfcTableOwner.RetrieveTableByOrigin	Retrieves and places a drawing table in the drawing at the specified point. The origin of the table is positioned at the specified point.

Miscellaneous Technical Changes

The following changes in PTC Creo Parametric 3.0 can affect the functional behavior of J-Link. PTC does not anticipate that these changes cause critical issues with existing J-Link applications.

Configuration Flag to Include Annotations During Export of PTC Creo Parametric Models

From PTC Creo Parametric 3.0 onward, a new configuration flag `EXPORT_INCLUDE_ANNOTATIONS` has been added. The flag includes annotations during the export of PTC Creo Parametric model to the specified format.

Display Style for Views

From PTC Creo Parametric 3.0 M010 onward, a new display type `DISPSTYLE_SHADED_WITH_EDGES` has been added to the enumerated data type `DisplayStyle`. This option allows you to display the model as a shaded solid along with its edges.

B

Sample Applications

This appendix lists the sample applications provided with J-Link.

Installing J-Link

J-Link is available on the same CD as PTC Creo Parametric. When PTC Creo Parametric is installed using PTC.SetUp, one of the optional components is `API Toolkits`. This includes PTC Creo Parametric TOOLKIT, J-Link, Web.Link, and VB API.

If you select J-Link, a directory called `jlink` is created under the PTC Creo Parametric loadpoint and J-Link is automatically installed in this directory. This directory contains all the libraries, example applications, and documentation specific to J-Link.

Sample Applications

The J-Link sample applications are available in the location `<creo_jlink_loadpoint>/jlink_appls`.

> **Note**
>
> You must set the configuration option `regen_failure_handling` to `resolve_mode` in the PTC Creo Parametric session before running the sample application `install_test`. From Creo Parametric 2.0 M060 onward, a configuration file (`config.pro`) has been provided for the `install_test` application. The `config.pro` contains the `regen_failure_handling` option.

InstallTest

Location	Main Class
`<creo_jlink_loadpoint>/jlink_appls/install_test`	`StartInstallTest`

The application `StartInstallTest` is used to check the J-Link synchronous installation. It verifies the following:

- Application start and stop functions.
- Menubar functions.
- Custom UI functions.
- Sequences, arrays, exceptions, and action listener functions.

Testing the J-Link Synchronous Installation

After the system administrator has installed J-Link, compile, link, and run a simple J-Link application on the machine you intend to use for development. Test if the installation of J-Link is present, complete, and visible from your machine.

To test the synchronous J-Link installation:

1. Set the `path` and `CLASSPATH` variables to include the `Java Development Kit` as described in Java Options and Debugging on page 421.
2. Set the `CLASSPATH` to include the J-Link synchronous archive and the current directory.

   On `Windows` set the `CLASSPATH` as:
   ```
   set CLASSPATH=<creo_loadpoint>\<datecode>\Common Files
   \text\java\pfc.jar;%CLASSPATH%
   ```
3. Compile the `java` files in the directory using the command `javac *.java`.

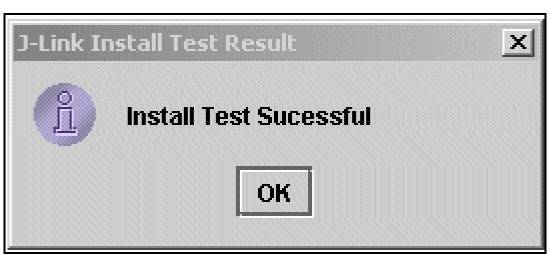
> **Note**
>
> The `java` file `AsyncInstallTest.java` is not compiled because it is used in the asynchronous mode only. Before compiling, rename this file to a non-`Java` file, that is, `AsyncInstallTest.bak`.

4. Create a `config.pro` file if you are using `Java 1.1`. Add the following line to this file:
   ```
   jlink_java2   off
   ```

> **Note**
>
> For more information on the supported JDK versions for synchronous J-Link refer to `http://www.ptc.com/partners/hardware/current/jlink.htm`.

5. Run PTC Creo Parametric.

   The PTC Creo Parametric **File** menu has a new button, added by the J-Link application, called `J-Link Install Test`. When you choose this button, the J-Link application displays a custom dialog indicating whether the installation test has succeeded:



> **Note**
>
> On Windows the results dialog may appear behind the PTC Creo Parametric window. Use **Atl-Tab** to switch to the `Java` dialog.

InstallTest

Location	Main Class
<creo_jlink_loadpoint>/jlink_appls/install_test	`AsyncInstallTest`

The application `AsyncInstallTest` is used to check the J-Link asynchronous installation. It verifies the following:

- Asynchronous J-Link setup
- PTC Creo Parametric start and stop methods
- Menubar functions
- Custom UI functions
- Sequences, arrays, exceptions, and action listener functions

Testing the J-Link Asynchronous Installation

To test the asynchronous J-Link application:

1. Set the `path` and `CLASSPATH` variables to include the `Java Development Kit` as described in Java Options and Debugging on page 421.
2. Set the `CLASSPATH` to include the J-Link asynchronous archive and the current directory.

   On `Windows` set the `CLASSPATH` as:
   ```
   set CLASSPATH=<creo_loadpoint>\<datecode>\Common Files
   \text\java\pfcasync.jar;%CLASSPATH%
   ```
3. Set the library path to include the asynchronous library and make sure that `PRO_COMM_MSG_EXE` is set.

   On `Windows` set the library path as:
   ```
   set path=<creo_loadpoint>\<datecode>\Common Files\<machine type>\lib;%PATH%
   set PRO_COMM_MSG_EXE=<creo_loadpoint>\<datecode>\Common Files
   \<machine type>\obj\pro_comm_msg.exe
   ```
4. Compile the `java` files in the directory using the command `javac *.java`.

Note

- The `java` file `StartInstallTest.java` does not get compiled as it is used in the synchronous mode only. Before compiling, rename this file to a non `java` file, that is, `StartInstallTest.bak`.
- Remove any `.class` files compiled previously using synchronous J-Link.
- Rename or remove the registry file (`creotk.dat`, `protk.dat`, or `prodev.dat`) from the location from where you are running the J-Link asynchronous test.

5. Run the application `java [asynchronous flags] AsyncInstallTest <command to run Creo Parametric>`.

Note

For more information on the supported JDK versions for asynchronous J-Link and the value of the asynchronous flags refer to `http://www.ptc.com/partners/hardware/current/jlink.htm`

jlinkexamples

Location	Main Class
`<creo_jlink_loadpoint>/jlink_appls/jlinkexamples`	`pfcExamplesMenu.java`, however note that not all examples may be tied to this class.

The application `jlinkexamples` is a collection of the *J-Link User's Guide* example source files . It covers most of the areas of J-Link.

jlinkasyncexamples

Location	Main Class
`<creo_jlink_loadpoint>/jlink_appls/jlinkasyncexamples`	Many independent examples

The application `jlinkasyncexamples` is a collection of the asynchronous *J-Link User's Guide* example source files.

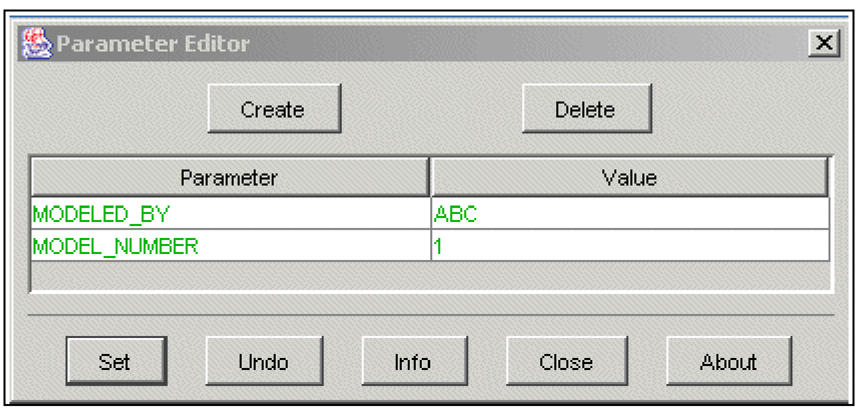
Parameter Editor

Location	Main Class
<creo_jlink_loadpoint>/jlink_appls/jlink_param	com.ptc.jlinkdemo.parameditor.ParamEditor

The parameter editor example demonstrates a synchronous J-Link user interface that governs parameters and parameter values in the model. Setup and run the J-Link Parameter Editor example using the following:

1. Set the path and `CLASSPATH` variables to include the `Java Development Kit` as described in (link) as described in Java Options and Debugging on page 421.
2. Set the `CLASSPATH` to include the `jlink_param` directory and the J-Link synchronous Jar file (`pfc.jar`). Refer to the section Testing the J-Link Synchronous Installation on page 414 for more information on setting the `CLASSPATH`.
3. Compile the code.

   On `Windows`, execute the batch file `compile.bat`.
4. Start PTC Creo Parametric from a directory containing the `protk.dat` file. Create or retrieve any model that contains parameters.
5. Select **J-Link Parameter Editor** from the **Applications** Menu. The system will display a graphical interface that contains a list of parameters for the selected model as shown in the following figure.



The parameter editor also supports the following customized types of parameters:

- Using the editor to create parameters with descriptive names (user interface names) of up to 80 character. The value of the assigned user interface name will be displayed as the parameter name in the J-Link user interface.
- Creating parameters that obey specific rules:
  - Enumerated lists

- Specified ranges
- Specified ranges, with values limited to a certain increments (for example, any multiple of 5 between 0 and 100).

When you open the J-Link user interface, the parameter value is governed by the rules assigned to it. If the parameter value is changed to fall outside the permitted values it will be highlighted in red.

Round Checker Utility

Location	Main Class
`<creo_jlink_loadpoint>/jlink_appls/jlink_elev`	`com.ptc.jlinkdemo.round.RoundChecker`

The round checker example demonstrates a synchronous J-Link utility that monitors the values assigned to round dimensions. If the value of any modified or newly created round is reduced below a programmed limit, a J-Link user interface will appear with information about the violation.

Use the following steps to setup and run the example:

1. Set the path and `CLASSPATH` variables to include the `Java Development Kit` as described in (link).
2. Set the `CLASSPATH` to include the `jlink_elev` directory and the J-Link synchronous Jar file (`pfc.jar`).
3. Compile the code.

   On `Windows`, execute the batch file `compile.bat`.
4. Load any PTC Creo Parametric model with rounds. Modify the round to less than 0.5. A J-Link dialog that identifies the problem will be displayed. The same dialog will appear if a new round that does not adhere to the specified dimensions is created.

Save Check Utility

Location	Main Class
`<creo_jlink_loadpoint>/jlink_appls/jlink_elev`	`com.ptc.jlinkdemo.savecheck.SaveChecker`

The save check example demonstrates a synchronous J-Link utility that presents a user interface that identifies if any problems exist in the model you are about to save. If any problems exist in the assigned parameter values or if a material has not been assigned to a part, the user interface will appear with information about the problems.

The instructions to setup and run the save check example are similar to the instructions for the round checker utility. To access the interface, choose **Tools**, **Perform Release Checks**.

C

Java Options and Debugging

This appendix describes how to control the procedure used by PTC Creo Parametric to invoke synchronous J-Link applications to enable you to use a non-default JVM or to debug your applications.

Supported Java Virtual Machine Versions

The machine information for the JVM versions supported by J-Link is available at:

http://www.ptc.com/partners/hardware/current/jlink.htm

The PTC Creo Parametric installation includes a default JVM shipped as a part of its CD image. For synchronous J-Link applications, PTC Creo Parametric uses the PTC Creo Parametric-supplied JVM by default.

PTC Creo Parametric includes the ability to override the default JVM command used to invoke J-Link applications. This allows you to:

- Use a non-standard JVM in your deployment, if that JVM has a feature or a fix that is necessary for your application to work correctly.
- Apply command line flags to the `Java` invocation, thus allowing it to be used for debugging or other customized purposes.

Overriding the Java command used by Synchronous J-Link

The JVM that is used can be overridden using one of the following mechanisms:

- The configuration option `jlink_java_command`, if set to the path to the `java` executable, will determine the JVM be used to start synchronous J-Link applications.
- The environment variable `PRO_JAVA_COMMAND` serves the same purpose as the configuration option. The environment variable takes precedence over the configuration option.

Note

The appropriate flags for synchronous J-Link as well as the flags for the user-supplied JRE must be used. The synchronous J-Link flags are listed on the J-Link platform page. It is recommended that you update the version of the JVM on your machine to the minimum supported version for the platform.

Debugging a Synchronous Mode Application

As PTC Creo Parametric has control over the start and stop of `Java` processes used by J-Link, you must use special controls to be able to debug an application. The most typical deployment should do the following:

1. Use the appropriate `javac` compiler flags to build the application debuggable.
2. Use the technique described in the section Overriding the Java command used by Synchronous J-Link on page 422 to set the `Java` command to the appropriate debug command line, for example, `[JDK_HOME]/bin/java.exe -Xdebug`
3. Start PTC Creo Parametric and let it invoke the `Java` application.
4. Attach your `Java` debugger to the process that was started by PTC Creo Parametric.

If you need to debug within the application start method, you can make the first invocation within that method a UI popup dialog box (`javax.swing.JOptionPane`) which will allow time to attach the debugger to the process.

CLASSPATH Variables

Synchronous Mode

If you are using the default JVM and are running J-Link applications on your machine, you need to add only your application classes to the classpath. The mechanisms to accomplish this are:

- Setting the environment variable `CLASSPATH`.
- Using the `java_app_classpath` field in the registry file. This field has a character limit of 2047 wide characters (`wchar_t`).
- Loading a user-specified Jar file through the user interface (only available for a model program).

PTC Creo Parametric will automatically add the J-Link archive `pfc.jar` to the `CLASSPATH`.

To compile J-Link applications, the environment variable `CLASSPATH` must include the path to the locations of classes and archives that you intend to use. Also, you must add J-Link archive `pfc.jar` to the `CLASSPATH`. This archive is located at `<creo_loadpoint>\<datecode>\Common Files\text\java\pfc.jar`.

JAVA Options for Asynchronous Mode

Asynchronous mode applications are started by an external `Java` process. Thus PTC Creo Parametric does not have any control over them, and you may use any JVM and command line to invoke them.

Note

Regardless of how the `Java` process is invoked, it must use the `Java` command line flags specified for asynchronous mode under

`http://www.ptc.com/partners/hardware/current/jlink.htm`

For both running and compiling, the environment variable `CLASSPATH` must point to the locations of the application classes and archives.

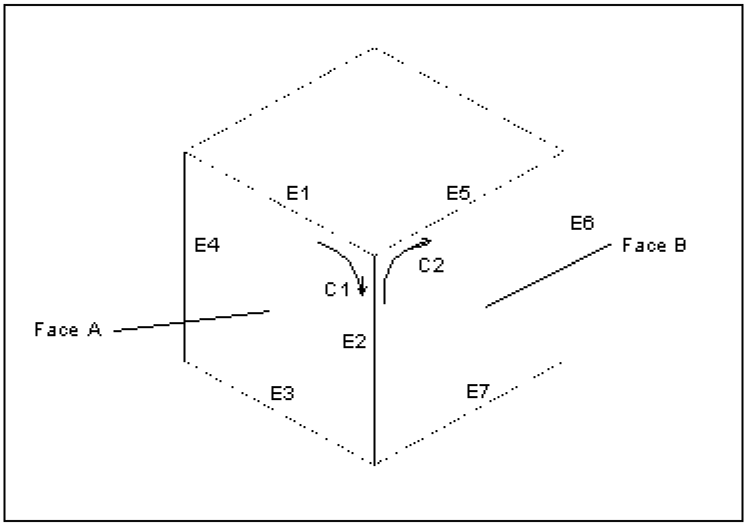
The `CLASSPATH` should also include the path to the J-Link asynchronous mode archive file `pfcasync.jar`. This archive is located at `<creo_loadpoint>\<datecode>\Common Files\text\java\pfcasync.jar`.

D

Geometry Traversal

This appendix illustrates the relationships between faces, contours, and edges. Examples E-1 through E-5 show some sample parts and list the information about their surfaces, faces, contours, and edges.

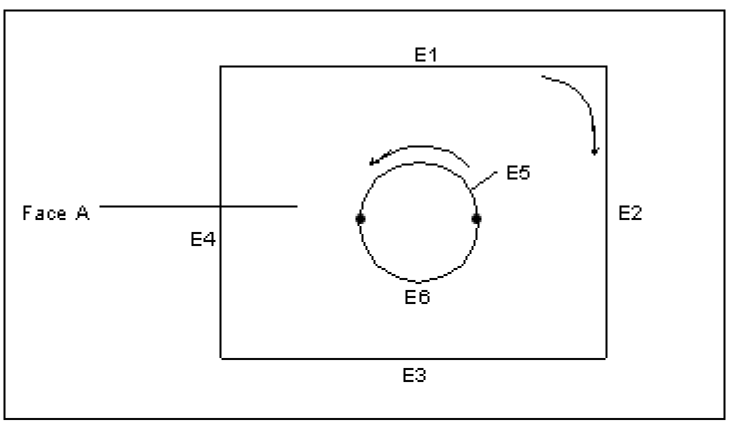
Example 1



This part has 6 faces.

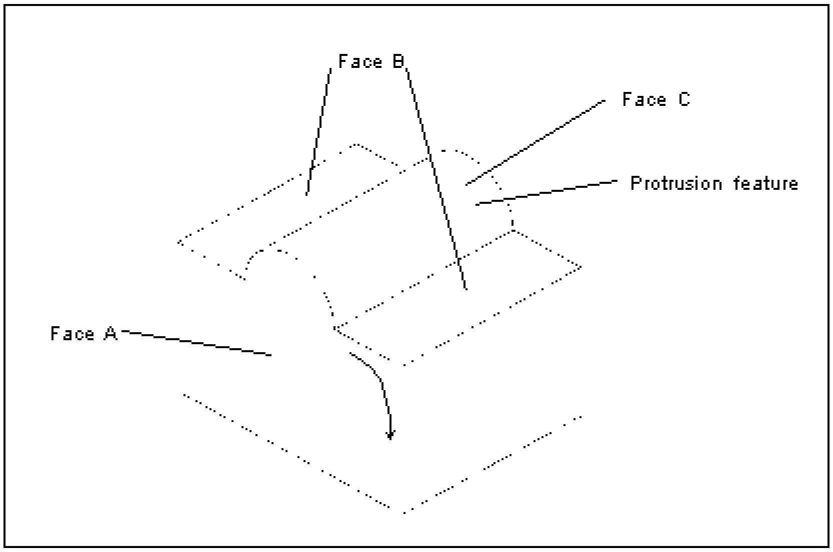
- Face A has 1 contour and 4 edges.
- Edge E2 is the intersection of faces A and B.
- Edge E2 is a component of contours C1 and C2.

Example 2



Face A has 2 contours and 6 edges.

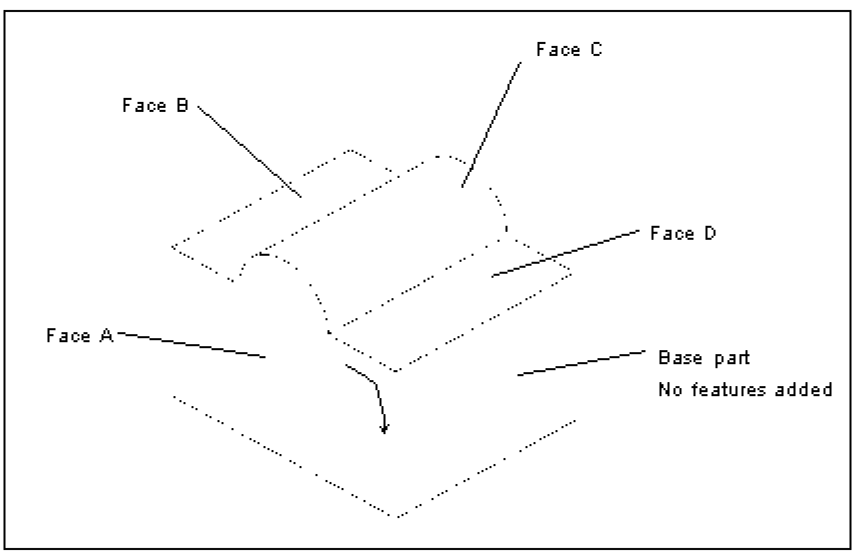
Example 3



This part was extruded from a rectangular cross section. The feature on the top was added later as an extruded protrusion in the shape of a semicircle.

- Face A has 1 contour and 6 edges.
- Face B has 2 contours and 8 edges.
- Face C has 1 contour and 4 edges.

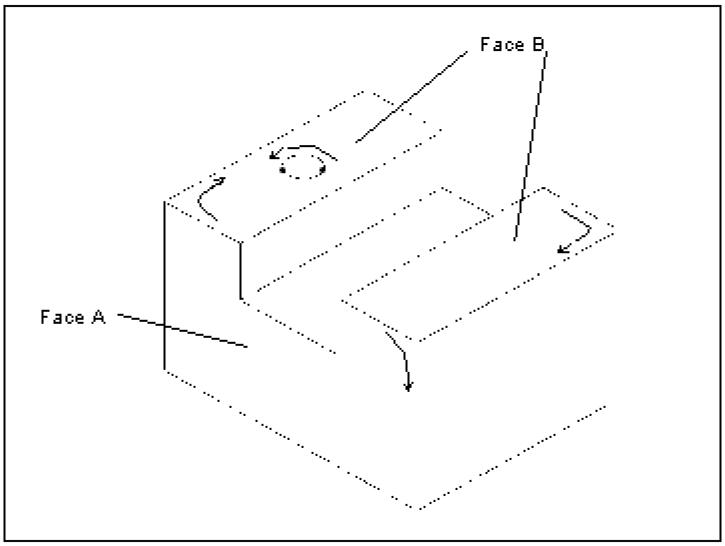
Example 4



This part was extruded from a cross section identical to Face A. In the Sketcher, the top boundary was sketched with two lines and an arc. The sketch was then extruded to form the base part, as shown.

- Face A has 1 contour and 6 edges.
- Face B has 1 contour and 4 edges.
- Face C has 1 contour and 4 edges.
- Face D has 1 contour and 4 edges.

Example 5



This part was extruded from a rectangular cross section. The slot and hole features were added later.

- Face A has 1 contour and 8 edges.
- Face B has 3 contours and 10 edges.

E

Geometry Representations

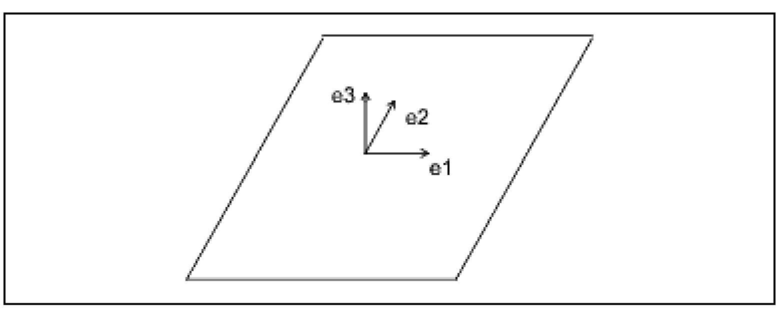
This appendix describes the geometry representations of the data used by J-Link.

Surface Parameterization

A surface in PTC Creo Parametric contains data that describes the boundary of the surface, and a pointer to the primitive surface on which it lies. The primitive surface is a three-dimensional geometric surface parameterized by two variables (*u* and *v*). The surface boundary consists of closed loops (contours) of edges. Each edge is attached to two surfaces, and each edge contains the *u* and *v* values of the portion of the boundary that it forms for both surfaces. Surface boundaries are traversed clockwise around the outside of a surface, so an edge has a direction in each surface with respect to the direction of traversal.

This section describes the surface parameterization. The surfaces are listed in order of complexity. For ease of use, the alphabetical listing of the data structures is as follows:

Plane



The plane entity consists of two perpendicular unit vectors (`e1` and `e2`), the normal to the plane (`e3`), and the origin of the plane.

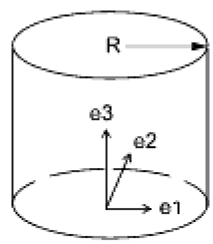
Data Format:

```
e1[3]        Unit vector, in the u direction
e2[3]        Unit vector, in the v direction
e3[3]        Normal to the plane
origin[3]    Origin of the plane
```

Parameterization:

```
(x, y, z) = u * e1 + v * e2 + origin
```

Cylinder



The generating curve of a cylinder is a line, parallel to the axis, at a distance `R` from the axis. The radial distance of a point is constant, and the height of the point is `v`.

Data Format:

```
e1[3]        Unit vector, in the u direction
e2[3]        Unit vector, in the v direction
e3[3]        Normal to the plane
origin[3]    Origin of the cylinder
radius       Radius of the cylinder
```

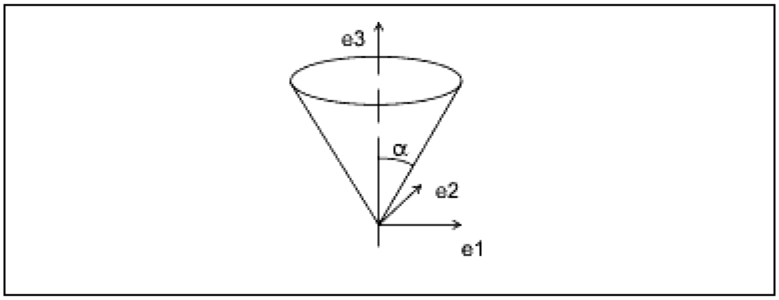
Parameterization:

```
(x, y, z) = radius * [cos(u) * e1 + sin(u) * e2] +
           v * e3 + origin
```

Engineering Notes:

For the cylinder, cone, torus, and general surface of revolution, a local coordinate system is used that consists of three orthogonal unit vectors (`e1`, `e2`, and `e3`) and an origin. The curve lies in the plane of `e1` and `e3`, and is rotated in the direction from `e1` to `e2`. The `u` surface parameter determines the angle of rotation, and the `v` parameter determines the position of the point on the generating curve.

Cone



The generating curve of a cone is a line at an angle alpha to the axis of revolution that intersects the axis at the origin. The `v` parameter is the height of the point along the axis, and the radial distance of the point is `v * tan(alpha)`.

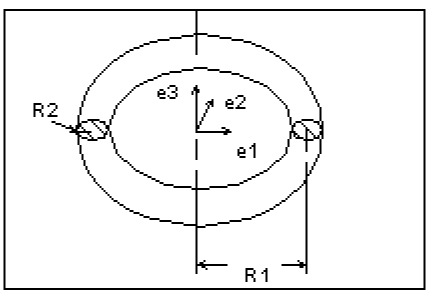
Data Format:

```
e1[3]       Unit vector, in the u direction
e2[3]       Unit vector, in the v direction
e3[3]       Normal to the plane
origin[3]   Origin of the cone
alpha       Angle between the axis of the cone
            and the generating line
```

Parameterization:

```
(x, y, z) = v * tan(alpha) * [cos(u) * e1 +
            sin(u) * e2] + v * e3 + origin
```

Torus



The generating curve of a torus is an arc of radius `R2` with its center at a distance `R1` from the origin. The starting point of the generating arc is located at a distance `R1 + R2` from the origin, in the direction of the first vector of the local coordinate system. The radial distance of a point on the torus is `R1 + R2 * cos(v)`, and the height of the point along the axis of revolution is `R2 * sin(v)`.

Data Format:

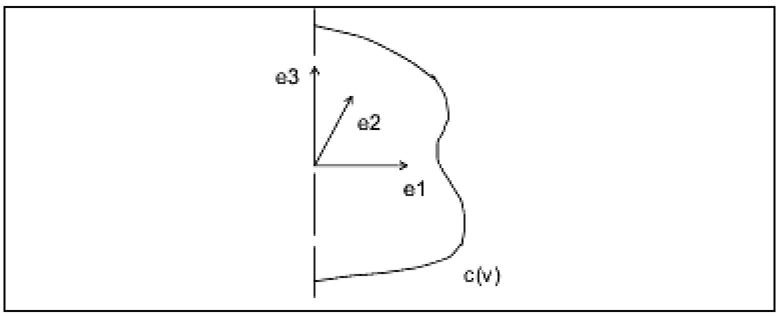
```
e1[3]       Unit vector, in the u direction
e2[3]       Unit vector, in the v direction
e3[3]       Normal to the plane
origin[3]   Origin of the torus
radius1     Distance from the center of the
```

```
           generating arc to the axis of
           revolution
radius2    Radius of the generating arc
```

Parameterization:

```
(x, y, z) = (R1 + R2 * cos(v)) * [cos(u) * e1 +
            sin(u) * e2] + R2 * sin(v) * e3 +
            origin
```

General Surface of Revolution



A general surface of revolution is created by rotating a curve entity, usually a spline, around an axis. The curve is evaluated at the normalized parameter `v`, and the resulting point is rotated around the axis through an angle `u`. The surface of revolution data structure consists of a local coordinate system and a curve structure.

Data Format:

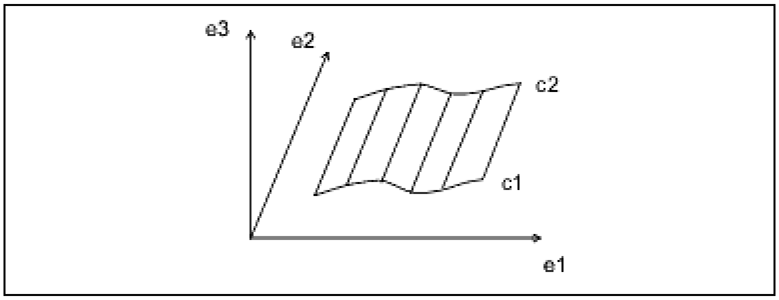
```
e1[3]       Unit vector, in the u direction
e2[3]       Unit vector, in the v direction
e3[3]       Normal to the plane
origin[3]   Origin of the surface of revolution
curve       Generating curve
```

Parameterization:

```
curve(v) = (c1, c2, c3) is a point on the curve.

(x, y, z) = [c1 * cos(u) - c2 * sin(u)] * e1 +
            [c1 * sin(u) + c2 * cos(u)] * e2 +
            c3 * e3 + origin
```

Ruled Surface



A ruled surface is the surface generated by interpolating linearly between corresponding points of two curve entities. The `u` coordinate is the normalized parameter at which both curves are evaluated, and the `v` coordinate is the linear parameter between the two points. The curves are not defined in the local coordinate system of the part, so the resulting point must be transformed by the local coordinate system of the surface.

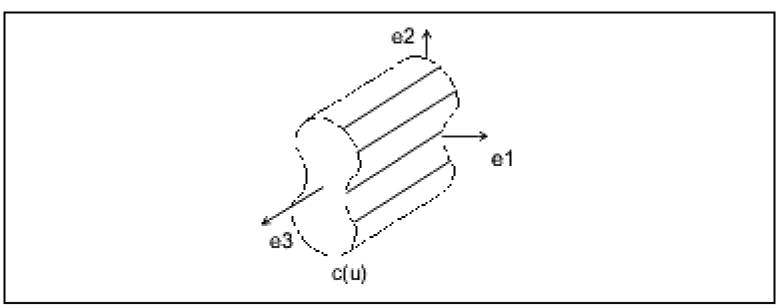
Data Format:

```
e1[3]       Unit vector, in the u direction
e2[3]       Unit vector, in the v direction
e3[3]       Normal to the plane
origin[3]  Origin of the ruled surface
curve_1     First generating curve
curve_2     Second generating curve
```

Parameterization:

```
(x', y', z') is the point in local coordinates.
(x', y', z') = (1 - v) * C1(u) + v * C2(u)
(x, y, z) = x' * e1 + y' * e2 + z' * e3 + origin
```

Tabulated Cylinder



A tabulated cylinder is calculated by projecting a curve linearly through space. The curve is evaluated at the `u` parameter, and the `z` coordinate is offset by the `v` parameter. The resulting point is expressed in local coordinates and must be transformed by the local coordinate system to be expressed in part coordinates.

Data Format:

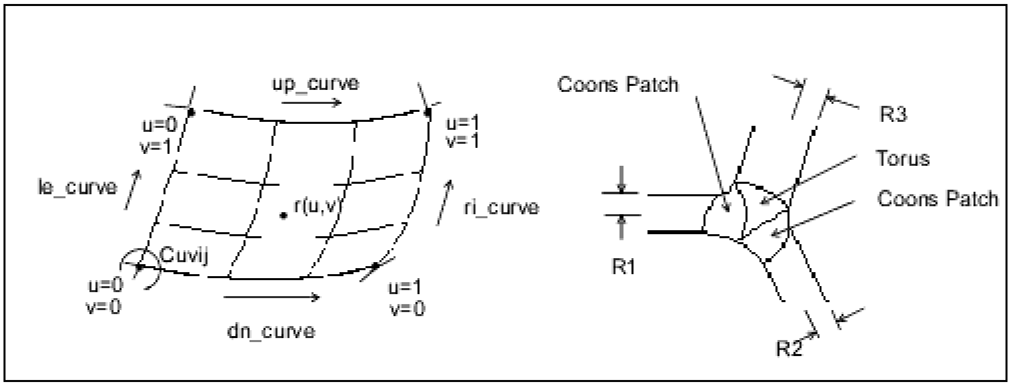
```
e1[3]       Unit vector, in the u direction
```

```
e2[3]       Unit vector, in the v direction
e3[3]       Normal to the plane
origin[3]   Origin of the tabulated cylinder
curve       Generating curve
```

Parameterization:

```
(x', y', z') is the point in local coordinates.
(x', y', z') = C(u) + (0, 0, v)
(x, y, z) = x' * e1 + y' * e2 + z' * e3 + origin
```

Coons Patch

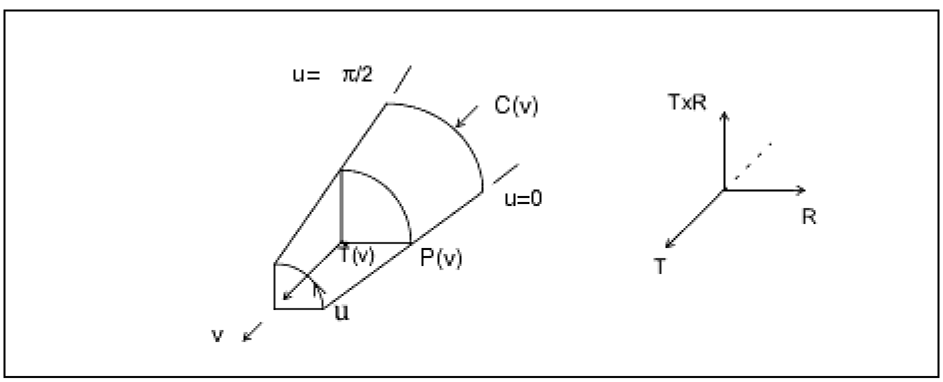


A Coons patch is used to blend surfaces together. For example, you would use a Coons patch at a corner where three fillets (each of a different radius) meet.

Data Format:

```
le_curve             u = 0 boundary
ri_curve             u = 1 boundary
dn_curve             v = 0 boundary
up_curve             v = 1 boundary
point_matrix[2][2]   Corner points
uvder_matrix[2][2]   Corner mixed derivatives
```

Fillet Surface



A fillet surface is found where a round or a fillet is placed on a curved edge, or on an edge with non-constant arc radii. On a straight edge, a cylinder would be used to represent the fillet.

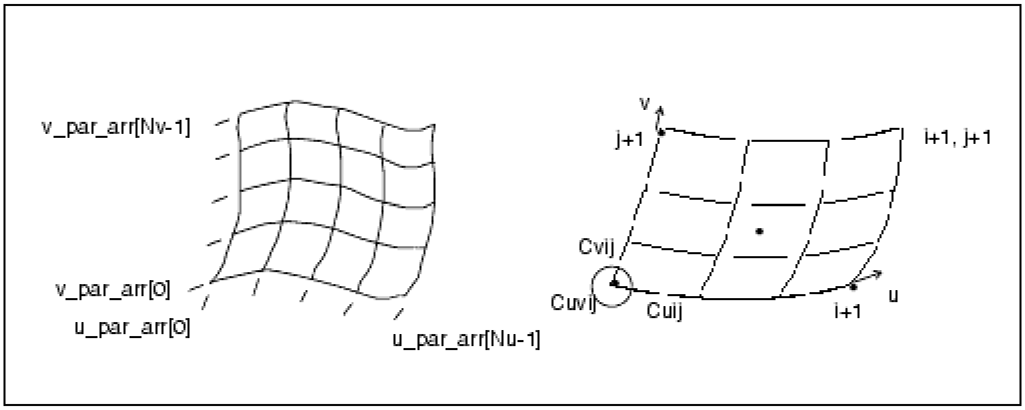
Data Format:

```
pnt_spline   P(v) spline running along the u = 0 boundary
ctr_spline   C(v) spline along the centers of the
             fillet arcs
tan_spline   T(v) spline of unit tangents to the
             axis of the fillet arcs
```

Parameterization:

```
R(v) = P(v) - C(v)
(x,y,z) = C(v) + R(v) * cos(u) + T(v) X R(v) *
          sin(u)
```

Spline Surface



The parametric spline surface is a nonuniform bicubic spline surface that passes through a grid with tangent vectors given at each point. The grid is curvilinear in `uv` space. Use this for bicubic blending between corner points.

Data Format:

```
u_par_arr[]      Point parameters, in the u
                 direction, of size Nu
v_par_arr[]      Point parameters, in the v
                 direction, of size Nv
point_arr[][3]   Array of interpolant points, of
                 size Nu x Nv
u_tan_arr[][3]   Array of u tangent vectors
                 at interpolant points, of size
                 Nu x Nv
v_tan_arr[][3]   Array of v tangent vectors at
                 interpolant points, of size
                 Nu x Nv
uvder_arr[][3]   Array of mixed derivatives at
                 interpolant points, of size
```

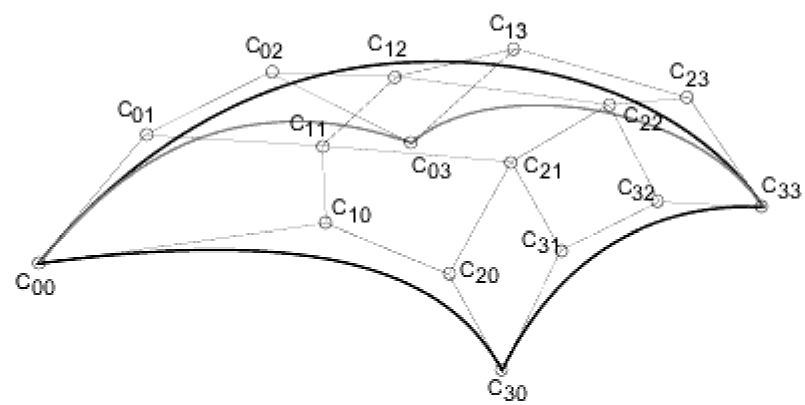
`Nu x Nv`

Engineering Notes:

- Allows for a unique 3x3 polynomial around every patch.
- There is second order continuity across patch boundaries.
- The point and tangent vectors represent the ordering of an array of `[i][j]`, where u varies with , and `v` varies with `j`. In walking through the `point_arr[][3]`, you will find that the innermost variable representing `v(j)` varies first.

NURBS Surface

The NURBS surface is defined by basis functions (in `u` and `v`), expandable arrays of knots, weights, and control points.



Cubic NURBS Surface

Data Format:

`deg[2]`	`Degree of the basis functions (in u and v)`
`u_par_arr[]`	`Array of knots on the parameter line u`
`v_par_arr[]`	`Array of knots on the parameter line v`
`wghts[]`	`Array of weights for rational NURBS, otherwise NULL`
`c_point_arr[][3]`	`Array of control points`

Definition:

$$R(u, v) = \frac{\sum_{i=0}^{N1} \sum_{j=0}^{N2} C_{i,j} \times B_{i,k}(u) \times B_{j,l}(v)}{\sum_{i=0}^{N1} \sum_{j=0}^{N2} w_{i,j} \times B_{i,k}(u) \times B_{j,l}(v)}$$

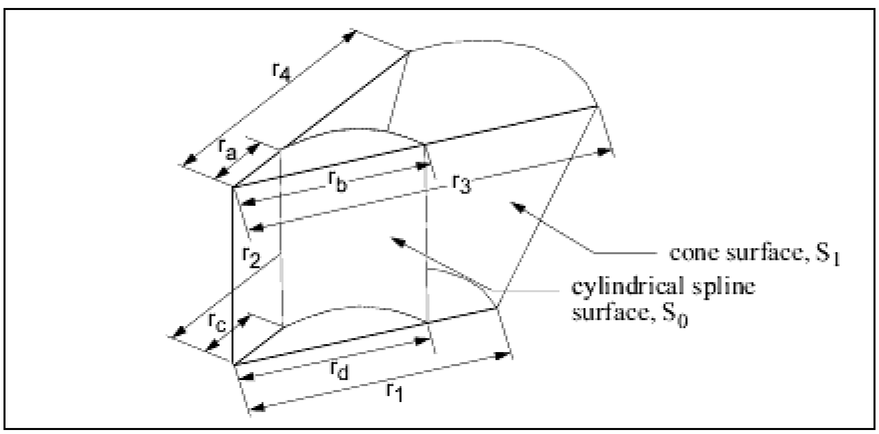
```
k    = degree in u
l    = degree in v
N1   = (number of knots in u) - (degree in u) - 2
N2   = (number of knots in v) - (degree in v) - 2
Bi,k = basis function in u
Bj, l  = basis function in v
wij   = weights
Ci, j   = control points (x,y,z) * wi,j
```

Engineering Notes:

The `weights` and `c_points_arr` arrays represent matrices of size `wghts[N1+1] [N2+1]` and `c_points_arr [N1+1] [N2+1]`. Elements of the matrices are packed into arrays in row-major order.

Cylindrical Spline Surface

The cylindrical spline surface is a nonuniform bicubic spline surface that passes through a grid with tangent vectors given at each point. The grid is curvilinear in modeling space.



Data Format:

```
e1[3]      x' vector of the local coordinate
           system
e2[3]      y' vector of the local coordinate
           system
e3[3]      z' vector of the local coordinate
           system, which corresponds to the
           axis of revolution of the surface
```

```
origin[3]  Origin of the local coordinate
           system
splsrf     Spline surface data structure
```

The spline surface data structure contains the following fields:

```
u_par_arr[]         Point parameters, in the
                    u direction, of size Nu
v_par_arr[]         Point parameters, in the
                    v direction, of size Nv
point_arr[][3]      Array of points, in
                    cylindrical coordinates,
                    of size Nu x Nv. The array
                    components are as follows:
                        point_arr[i][0] - Radius
                        point_arr[i][1] - Theta
                        point_arr[i][2] - Z
u_tan_arr[][3]      Array of u tangent vectors.
                    in cylindrical coordinates,
                    of size Nu x Nv
v_tan_arr[][3]      Array of v tangent vectors,
                    in cylindrical coordinates,
                    of size Nu x Nv
uvder_arr[][3]      Array of mixed derivatives,
                    in cylindrical coordinates,
                    of size Nu x Nv
```

Engineering Notes:

If the surface is represented in cylindrical coordinates (r, theta, z), the local coordinate system values (x', y', z') are interpreted as follows:

```
x' = r cos (theta)
y' = r sin (theta)
z' = z
```

A cylindrical spline surface can be obtained, for example, by creating a smooth rotational blend (shown in the figure on the previous page).

In some cases, you can replace a cylindrical spline surface with a surface such as a plane, cylinder, or cone. For example, in the figure, the cylindrical spline surface S1 was replaced with a cone (r1 = r2, r3 = r4, and r1 ¼ r3).

If a replacement cannot be done (such as for the surface S0 in the figure (ra ¼ rb or rc ¼ rd)), leave it as a cylindrical spline surface representation.

Edge and Curve Parameterization

This parameterization represents edges (line, arc, and spline) as well as the curves (line, arc, spline, and NURBS) within the surfaces.

This section describes edges and curves, arranged in order of complexity. For ease of use, the alphabetical listing is as follows:

- Arc on page 440
- Line on page 440
- NURBS on page 441
- Spline on page 440

Line

Data Format:

```
end1[3]   Starting point of the line
end2[3]   Ending point of the line
```

Parameterization:

```
(x, y, z) = (1 - t) * end1 + t * end2
```

Arc

The arc entity is defined by a plane in which the arc lies. The arc is centered at the origin, and is parameterized by the angle of rotation from the first plane unit vector in the direction of the second plane vector. The start and end angle parameters of the arc and the radius are also given. The direction of the arc is counterclockwise if the start angle is less than the end angle, otherwise it is clockwise.

Data Format:

```
vector1[3]     First vector that defines the
               plane of the arc
vector2[3]     Second vector that defines the
               plane of the arc
origin[3]      Origin that defines the plane
               of the arc
start_angle    Angular parameter of the starting
               point
end_angle      Angular parameter of the ending
               point
radius         Radius of the arc.
```

Parameterization:

```
t' (the unnormalized parameter) is
   (1 - t) * start_angle + t * end_angle
(x, y, z) = radius * [cos(t') * vector1 +
   sin(t') * vector2] + origin
```

Spline

The spline curve entity is a nonuniform cubic spline, defined by a series of three-dimensional points, tangent vectors at each point, and an array of unnormalized spline parameters at each point.

Data Format:

```
par_arr[]       Array of spline parameters
                (t) at each point.
pnt_arr[][3]    Array of spline interpolant points
tan_arr[][3]    Array of tangent vectors at
                each point
```

Parameterization:

`x`, `y`, and `z` are a series of unique cubic functions, one per segment, fully determined by the starting and ending points, and tangents of each segment.

Let `p_max` be the parameter of the last spline point. Then, `t`, the unnormalized parameter, is `t * p_max`.

Locate the th spline segment such that:

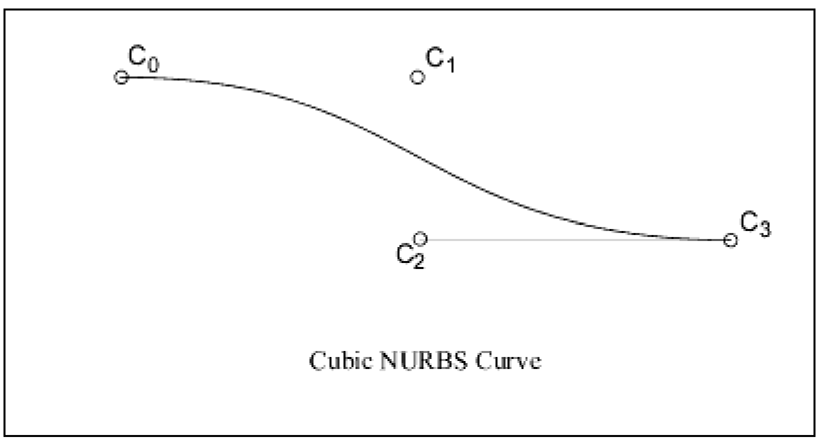
```
par_arr[i] < t' < par_arr[i+1]
```

(If `t` < 0 or `t` > +1, use the first or last segment.)

```
t0 = (t' - par_arr[i])   / (par_arr[i+1] - par_arr[i])
t1 = (par_arr[i+1] - t') / (par_arr[i+1] - par_arr[i])
```

NURBS

The NURBS (nonuniform rational B-spline) curve is defined by expandable arrays of knots, weights, and control points.



Data Format:

```
degree          Degree of the basis function
params[]        Array of knots
weights[]       Array of weights for rational
                NURBS, otherwise NULL.
c_pnts[][3]     Array of control points
```

Definition:

$$R(t) = \frac{\sum_{i=0}^{N} C_i \times B_{i,k}(t)}{\sum_{i=0}^{N} w_i \times B_{i,k}(t)}$$

```
k      = degree of basis function
N     = (number of knots) - (degree) - 2
wi    = weights
Ci     = control points (x, y, z) * wi
Bi,k = basis functions
```

By this equation, the number of control points equals `N+1`.

References:

Faux, I.D., M.J. Pratt. Computational Geometry for Design and Manufacture. Ellis Harwood Publishers, 1983.

Mortenson, M.E. Geometric Modeling. John Wiley & Sons, 1985.

F

J-Link Classes

This appendix lists and briefly describes the classes that make up the J-Link interface.

List of J-Link Classes

The following table briefly describes the classes in the J-Link interface.

Class	Package	Returned by
ActionListener	pfcBase	Base class; not returned.
This class defines an action listener.		
ActionListeners	pfcBase	ActionListeners.create()
This data type is used to specify a list of action listeners.		
ActionSource	pfcBase	Base class; not returned.
This class specifies an action source.		
ActionSources	pfcBase	ActionSources.create()
This type describes an array of action sources.		
AnalysisFeat	pfcAnalysisFeat	Downcast of pfcFeature.Feature.
This feature type specifies an analysis feature.		
Arc	pfcGeometry	Downcast of pfcGeometry.Curve.
This class defines an arc.		
AreaNibbleFeat	pfcAreaNibbleFeat	Downcast of pfcFeature.Feature.
This feature type specifies a nibble area. This feature type is used in sheetmetal applications.		
Arrow	pfcGeometry	Downcast of pfcGeometry.Curve.
This class defines an arrow.		
Assembly	pfcAssembly	Session.CreateAssem bly() , ComponentPath.GetRoot()
This class describes an assembly.		
AssemblyCutCopyFeat	pfcAssemblyCutCopyFeat	Downcast of pfcFeature.Feature.
This feature type specifies a cut and copied feature, which is used in the Assembly Design module.		
AssemblyCutFeat	pfcAssemblyCutFeat	Downcast of pfcFeature.Feature.
This feature type specifies a cutout feature, which is used in the Assembly Design module.		
AttachFeat	pfcAttachFeat	Downcast of pfcFeature.Feature.
This feature type specifies an attached feature.		
AttachVolumeFeat	pfcAttachVolumeFeat	Downcast of pfcFeature.Feature.
This feature type specifies an attached volume.		
AuxiliaryFeat	pfcAuxiliaryFeat	Downcast of pfcFeature.Feature.
This feature type specifies an auxiliary feature.		
Axis	pfcGeometry	Downcast of pfcModelItem.ModelItem.
This class defines an axis.		

Class	Package	Returned by
`BaseDimension`	`pfcDimension`	Base class; not returned.
This class defines the base dimension.		
`BaseParameter`	`pfcModelItem`	Base class; not returned.
Describes the base parameter.		
`BeamSectionFeat`	`pfcBeamSectionFeat`	Downcast of `pfcFeature.Feature`.
This feature type specifies a beam section.		
`BendBackFeat`	`pfcBendBackFeat`	Downcast of `pfcFeature.Feature`.
This feature type specifies a bend back feature, which is used in the PTC Creo NC Sheetmetal module.		
`BendFeat`	`pfcBendFeat`	Downcast of `pfcFeature.Feature`.
This feature type specifies a bend feature.		
`BldOperationFeat`	`pfcBldOperationFeat`	Downcast of `pfcFeature.Feature`.
This feature type specifies a build operation.		
`BOMExportInstructions`	`pfcModel`	`pfcModel.BOMExport Instructions_Create()`
Used to export a BOM for an assembly.		
`BSpline`	`pfcGeometry`	Downcast of `pfcGeometry.Curve`.
This class defines a B-spline curve.		
`BSplinePoint`	`pfcGeometry`	`BSplinePoints.get()`
This class defines a B-spline point.		
`BSplinePoints`	`pfcGeometry`	`BSplinePoints.create()`, `BSpline.GetPoints()`
This data type is used to specify an array of B-spline points.		
`BulkObjectFeat`	`pfcBulkObjectFeat`	Downcast of `pfcFeature.Feature`.
This feature type specifies a bulk object.		
`CableCosmeticFeat`	`pfcCableCosmeticFeat`	Downcast of `pfcFeature.Feature`.
This feature type specifies a cosmetic feature used with the cabling.		
`CableFeat`	`pfcCableFeat`	Downcast of `pfcFeature.Feature`.
This feature type specifies a cabling feature.		
`CableLocationFeat`	`pfcCableLocationFeat`	Downcast of `pfcFeature.Feature`.
This feature type specifies a cable location.		
`CableParamsFileInstruc tions`	`pfcModel`	`pfcModel.CableParams FileInstructions_ Create()`
Used to export cable parameters from an assembly.		
`CableSegmentFeat`	`pfcCableSegmentFeat`	Downcast of

Class	Package	Returned by
		pfcFeature.Feature.
This feature type specifies a cable segment.		
CATIAExportInstructions	pfcModel	pfcModel.CATIAExport Instructions_Create()
Used to export a part or assembly in CATIA format (as precise geometry)		
CATIAFacetsExport Instructions	pfcModel	pfcModel.CATIAFacets ExportInstructions _Create()
Used to export a part or assembly in CATIA format (as a faceted model).		
CavDeviationFeat	pfcCavDeviationFeat	Downcast of pfcFeature.Feature.
This feature type specifies a deviation feature, which is used in the Verify module.		
CavFitFeat	pfcCavFitFeat	Downcast of pfcFeature.Feature.
This feature type specifies a fit feature, which is used in the Verify module.		
CavScanSetFeat	pfcCavScanSetFeat	Downcast of pfcFeature.Feature.
This feature type specifies a scanset feature, which is used in the Verify module.		
CGMExportType	pfcModel	CGMExportType.FromInt() or by using one of the static instances (e.g., EXPORT_CGM_ CLEAR_TEXT)
Indicates whether a CGM export file should be ASCII (clear text) or binary (mil spec)		
CGMFILEExportInstruc tions	pfcModel	pfcModel.CGMFILEExport Instructions_Create()
Used to export a drawing in CGM format.		
CGMScaleType	pfcModel	CGMScaleType.FromInt() or by using any of the static instances (e.g., EXPORT_CGM_ ABSTRACT)
Indicates whether a CGM export file should include abstract or metric units		
ChamferFeat	pfcChamferFeat	Downcast of pfcFeature.Feature.
This feature type specifies a chamfer.		
ChannelFeat	pfcChannelFeat	Downcast of pfcFeature.Feature.
This feature type specifies a channel.		
Child	pfcObject	Parent.GetChild()
Describes a child feature.		
Circle	pfcGeometry	Downcast of pfcGeometry.Curve.
This class defines a circle.		
CMMConstrFeat	pfcCMMConstrFeat	Downcast of pfcFeature.Feature.
This feature type specifies a construction feature used in the CMM module.		
CMMMeasureStepFeat	pfcCMMMeasureStepFeat	Downcast of pfcFeature.Feature.

Class	Package	Returned by
This feature type specifies a measured step feature, which is used in the CMM module.		
CMMVerifyFeat	pfcCMMVerifyFeat	Downcast of pfcFeature.Feature.
This feature type specifies a verify feature, which is used in the CMM module.		
ColorRGB	pfcBase	pfcBase.ColorRGB _Create(), Session.GetRGBFrom StdColor()
Specifies the red, green, and blue (RGB) values of a color.		
CompModelReplace	pfcComponentFeat	ComponentFeat.Create ReplaceOp()
Used to replace one model in a component with another.		
ComponentFeat	pfcComponentFeat	Downcast of pfcFeature.Feature.
Specifies a component feature.		
ComponentPath	pfcAssembly	Selection.GetPath()
This class describes a component path.		
ComponentType	pfcComponentFeat	ComponentType.FromInt() or by using any of the static instances (e.g., COMP_ WORKPIECE)
This enumerated type lists the possible component types.		
CompositeCurve	pfcGeometry	Downcast of pfcGeometry.Curve.
This class defines a composite curve.		
Cone	pfcGeometry	Downcast of pfcGeometry.Surface.
This class defines a cone.		
ConnectorParamExportIn structions	pfcModel	pfcModel.ConnectorParam ExportInstructions_ Create()
Used to write the parameters of a connector to a file.		
ContMapFeat	pfcContMapFeat	Downcast of pfcFeature.Feature.
This feature type specifies a contour map.		
Contour	pfcGeometry	Contours.get(), Contour.FindContaining Contour()
This class describes a contour.		
Contours	pfcGeometry	Contours.create(), Surface.ListContours()
This data type is used to describe an array of contours.		
ContourTraversal	pfcGeometry	ContourTraversal. FromInt() or by using any of the static instances (e.g., CONTOUR_TRAV_INTERNAL)
This enumerated type lists the possible values for traversing the contour.		
CoordAxis	pfcBase	CoordAxis.

Class	Package	Returned by
		FromInt() or by using any of the static instances (e.g., COORD_AXIS_X)
This enumerated type specifies the axes of a coordinate system.		
CoordSysExportInstructions	pfcModel	Base class; not returned.
Base class of classes that export files with information that describes faceted, solid models		
CoordSysFeat	pfcCoordSysFeat	Downcast of pfcFeature.Feature.
Describes a coordinate system feature, including constraint and translation information.		
CoordSystem	pfcGeometry	Downcast of pfcModelItem.ModelItem.
This class describes a coordinate system.		
CoreFeat	pfcCoreFeat	Downcast of pfcFeature.Feature.
This feature type specifies a core feature.		
CornerChamferFeat	pfcCornerChamferFeat	Downcast of pfcFeature.Feature.
This feature type specifies a corner chamfer.		
CosmeticFeat	pfcCosmeticFeat	Downcast of pfcFeature.Feature.
This feature type specifies a cosmetic feature.		
CrossSectionFeat	pfcCrossSectionFeat	Downcast of pfcFeature.Feature.
This feature type specifies a cross section.		
CurvatureData	pfcGeometry	Surface.EvalPrincipalCurv()
This class specifies the curvature data.		
Curve	pfcGeometry	Downcast of pfcGeometry.Curve.
This class defines a curve.		
CurveFeat	pfcCurveFeat	Downcast of pfcFeature.Feature.
Specifies a curve feature.		
Curves	pfcGeometry	Curves.create(), CompositeCurve.ListElements()
This data type is used to specify an array of curves.		
CurveStartPoint	pfcCurveFeat	CurveStartPoint.FromInt() or by using any of the static instances (e.g., CURVE_START)
This enumerated type lists the possible starting points of the datum curve offset.		
CurveXYZData	pfcGeometry	GeomCurve.Eval3DData(), GeomCurve.EvalFromLength()

Class	Package	Returned by
Stores the results of an edge evaluation.		
CustomizeFeat	pfcCustomizeFeat	Downcast of pfcFeature.Feature.
This feature type specifies a customized feature.		
CutFeat	pfcCutFeat	Downcast of pfcFeature.Feature.
This feature type specifies a cut feature.		
CutMotionFeat	pfcCutMotionFeat	Downcast of pfcFeature.Feature.
This feature type specifies a cut motion feature, which is used in the PTC Creo NC module.		
Cylinder	pfcGeometry	Downcast of pfcGeometry.Surface.
This class defines a cylinder.		
DatumAxisFeat	pfcDatumAxisFeat	Downcast of pfcFeature.Feature.
This feature type specifies a datum axis feature.		
DatumPlaneFeat	pfcDatumPlaneFeat	Downcast of pfcFeature.Feature.
This feature type specifies a datum plane.		
DatumPointFeat	pfcDatumPointFeat	Downcast of pfcFeature.Feature.
This feature type specifies a datum point.		
DatumQuiltFeat	pfcDatumQuiltFeat	Downcast of pfcFeature.Feature.
This feature type specifies a datum quilt.		
DatumSurfaceFeat	pfcDatumSurfaceFeat	Downcast of pfcFeature.Feature.
This feature type specifies a datum surface.		
DeclareFeat	pfcDeclareFeat	Downcast of pfcFeature.Feature.
This feature type specifies a declared feature.		
DeformAreaFeat	pfcDeformAreaFeat	Downcast of pfcFeature.Feature.
This feature type specifies a deformed area.		
DeleteOperation	pfcFeature	Feature.CreateDeleteOp()
This class defines a delete operation.		
Dependencies	pfcModel	Dependencies.create(), Model.ListDependencies()
This data type is used to specify the first-level dependencies for an object.		
Dependency	pfcModel	Dependencies.get()
Describes the first-level dependency for an object.		
Dimension	pfcDimension	Downcast of pfcModelItem.ModelItem.
This class describes a dimension.		

Class	Package	Returned by
DimensionType	pfcDimension	DimensionType.FromInt() or by using any of the static instances (e.g., DIM_LINEAR)
This enumerated type lists the possible dimension types.		
DimTolerance	pfcDimension	Dimension.GetToler ance()
This class defines the dimension tolerance.		
DimTolLimits	pfcDimension	DimTolLimits.Create()
This class displays the limits for the dimension tolerance.		
DimTolPlusMinus	pfcDimension	DimTolPlusMinus.Create
This class displays the dimension tolerance in the form +/-x, where x is the plus tolerance. The value of the minus tolerance is unused.		
DimTolSymmetric	pfcDimension	DimTolSymmetric. Create()
This class displays the dimension tolerance in symmetrical form.		
DisplayStatus	pfcLayer	DisplayStatus.FromInt() or by using any of the static instances (e.g., LAYER_NORMAL).
This enumerated type lists the possible values for the display status of a layer.		
Dome2Feat	pfcDome2Feat	Downcast of pfcFeature.Feature.
This feature type specifies a section dome.		
DomeFeat	pfcDomeFeat	Downcast of pfcFeature.Feature.
This feature type specifies a radius dome.		
DraftFeat	pfcDraftFeat	Downcast of pfcFeature.Feature.
This feature type specifies a draft.		
DraftLineFeat	pfcDraftLineFeat	Downcast of pfcFeature.Feature.
This feature type specifies a line draft, which is used with the Legacy module.		
Drawing	pfcDrawing	Session.CreateDrawing
This class describes a drawing.		
DrillFeat	pfcDrillFeat	Downcast of pfcFeature.Feature.
This feature type specifies a drill, which is used with the PTC Creo NC module.		
DrillGroupFeat	pfcDrillGroupFeat	Downcast of pfcFeature.Feature.
This feature type specifies a drill group, which is used in the PTC Creo NC module.		
DrvToolCurveFeat	pfcDrvToolCurveFeat	Downcast of pfcFeature.Feature.
This feature type specifies a driven-tool curve, which is used in the PTC Creo NC module.		
DrvToolEdgeFeat	pfcDrvToolEdgeFeat	Downcast of pfcFeature.Feature.
This feature type specifies a driven-tool edge, which is used in the PTC Creo NC module.		
DrvToolProfileFeat	pfcDrvToolProfileFeat	Downcast of pfcFeature.Feature.

Class	Package	Returned by
This feature type specifies a driven-tool profile.		
DrvToolSketchFeat	pfcDrvToolSketchFeat	Downcast of pfcFeature.Feature.
This feature type specifies a driven-tool sketch.		
DrvToolSurfFeat	pfcDrvToolSurfFeat	Downcast of pfcFeature.Feature.
This feature type specifies a driven-tool surface.		
DrvToolTwoCntrFeat	pfcDrvToolTwoCntrFeat	Downcast of pfcFeature.Feature.
This feature type specifies a tool with two centers.		
DWGSetupExportInstructions	pfcModel	pfcModel.DWGSetupExportInstructions_Create()
Used to export a drawing setup file.		
DXFExportInstructions	pfcModel	pfcModel.DXFExportInstructions_Create()
Used to export a drawing in DXF format.		
EarFeat	pfcEarFeat	Downcast of pfcFeature.Feature.
This feature type specifies an ear feature.		
Edge	pfcGeometry	Edges.get(), Edge.GetEdge1(), Edge.GetEdge2()
Describes the edge, including the next and previous edge, and the two surfaces.		
EdgeBendFeat	pfcEdgeBendFeat	Downcast of pfcFeature.Feature.
This feature type specifies an edge bend.		
EdgeEvalData	pfcGeometry	Edge.EvalUV()
This class provides edge evaluation data.		
Edges	pfcGeometry	Edges.create(), Contour.ListElements()
This data type is used to specify an array of edges.		
Ellipse	pfcGeometry	Downcast of pfcGeometry.Curve.
This class defines an ellipse.		
EtchFeat	pfcEtchFeat	Downcast of pfcFeature.Feature.
This feature type specifies an etched feature.		
ExplodeLineFeat	pfcExplodeLineFeat	Downcast of pfcFeature.Feature.
This feature type specifies an explode line.		
ExportInstructions	pfcModel	Base class; not returned.
Base class to all the export-instructions classes.		
ExportType	pfcModel	ExportType.FromInt() or by using any of the static instances (e.g., EXPORT_IGES_SECTION)
Enumeration of the available export options.		

Class	Package	Returned by
ExpRatioFeat	pfcExpRatioFeat	Downcast of pfcFeature.Feature.
This feature type specifies an exponential-ratio feature.		
ExtendFeat	pfcExtendFeat	Downcast of pfcFeature.Feature.
This feature type specifies an extend feature.		
ExtractFeat	pfcExtractFeat	Downcast of pfcFeature.Feature.
This feature type specifies an extraction.		
FamColComp	pfcFamily	FamilyMember.CreateComponentColumn()
This class describes a component column in a family table.		
FamColCompModel	pfcFamily	FamilyMember.CreateCompModelColumn()
This class describes a component model column in a family table.		
FamColDimension	pfcFamily	FamilyMember.CreateDimensionColumn()
This class specifies a dimension column in a family table.		
FamColExternalRef	pfcFamily	Not returned.
This class describes an external reference column in a family table.		
FamColFeature	pfcFamily	FamilyMember.CreateFeatureColumn()
This class specifies a family column feature.		
FamColGroup	pfcFamily	FamilyMember.CreateGroupColumn()
This class describes a group column in a family table.		
FamColGTol	pfcFamily	Not returned.
This class describes a geometric tolerance (gtol) column in a family table.		
FamColMergePart	pfcFamily	FamilyMember.CreateMergePartColumn()
This class describes a merged part column in a family table.		
FamColModelItem	pfcFamily	Base class; not returned.
This class specifies a column in the family table.		
FamColParam	pfcFamily	FamilyMember.CreateParamColumn()
This class specifies a parameter column in a family table.		
FamColSystemParam	pfcFamily	Not returned.
This class describes a system parameter column in a family table.		
FamColUDF	pfcFamily	Not returned.
This class describes a UDF column in a family table.		
FamilyMember	pfcFamily	FamilyMember.GetParent()
This class describes a member in a family table.		
FamilyTableColumn	pfcFamily	FamilyMember.AddColumn() ,

Class	Package	Returned by
		FamilyTableColumns.get()
This class specifies a column in a family table.		
FamilyTableColumns	pfcFamily	FamilyTableColumns.create(), FamilyMember.ListColumns()
This data type is used to specify a list of columns in a family table.		
FamilyTableRow	pfcFamily	FamilyMember.AddRow(), FamilyMember.GetRow(), FamilyTableRows.get()
This class specifies a row in a family table.		
FamilyTableRows	pfcFamily	FamilyTableRows.create() , FamilyMember.ListRows()
This data type is used to specify a list of rows in a family table.		
FeatIdExportInstructions	pfcModel	Base class; not returned.
Base class of instructions classes that export data for a single feature.		
FeatInfoExportInstructions	pfcModel	pfcModel.FeatInfoExportInstructions_Create()
Used to export information about one feature in a part or assembly.		
Feature	pfcFeature	Solid.GetFeatureByName() , FeatureOperation.GetOpFeature(). Also, by downcasting pfcModelItem.ModelItem.
This class defines the feature information.		
FeatureActionListener_u	pfcFeature	Base class; not returned.
Abstract base class that all user-defined FeatureActionListener classes must extend.		
FeatureActionListener	pfcFeature	Base class; not returned.
Interface that must be implemented by user-defined classes that respond to Feature events.		
FeatureGroup	pfcFeature	Feature.GetGroup()
This class describes a feature group.		
FeatureOperation	pfcFeature	FeatureOperations.get()
This class defines a feature operation.		
FeatureOperations	pfcFeature	FeatureOperations.create()
This class specifies a list of feature operations.		
FeaturePattern	pfcFeature	Feature.GetPattern(), FeatureGroup.GetPattern()
This class specifies a feature pattern.		
FeaturePlacement	pfcFeature	Feature.GetPlacement()
Specifies the placement of a feature.		
Features	pfcFeature	Features.create(),

Class	Package	Returned by
		`Feature.ListChildren()`, `Feature.ListParents()`, `FeaturePattern.ListMembers()`
This data type specifies an array of features.		
`FeatureStatus`	`pfcFeature`	`FeatureStatus.FromInt()` or by using any of the static instances (e.g., `FEAT_ACTIVE`)
This enumerated type specifies the feature status.		
`FeatureType`	`pfcFeature`	`FeatureType.FromInt()` or by using any of the static instances (e.g., `FEATTYPE_PROTRUSION`)
This enumerated type lists the possible feature types.		
`FIATExportInstructions`	`pfcModel`	`pfcModel.FIATExportInstructions_Create()`
Used to export a part or assembly in `FIAT` format.		
`FirstFeat`	`pfcFirstFeat`	Downcast of `pfcFeature.Feature`.
This feature type specifies the first feature in a model.		
`FixtureSetupFeat`	`pfcFixtureSetupFeat`	Downcast of `pfcFeature.Feature`.
This feature type specifies a fixture setup.		
`FlangeFeat`	`pfcFlangeFeat`	Downcast of `pfcFeature.Feature`.
This feature type specifies a flange.		
`FlatPatFeat`	`pfcFlatPatFeat`	Downcast of `pfcFeature.Feature`.
This feature type specifies a flat pattern.		
`FlatRibbonSegmentFeat`	`pfcFlatRibbonSegmentFeat`	Downcast of `pfcFeature.Feature`.
This feature type is for flat ribbon segments.		
`FlattenFeat`	`pfcFlattenFeat`	Downcast of `pfcFeature.Feature`.
This feature type specifies a flattened-harness feature.		
`FormFeat`	`pfcFormFeat`	Downcast of `pfcFeature.Feature`.
This feature type specifies a form feature.		
`FreeFormFeat`	`pfcFreeFormFeat`	Downcast of `pfcFeature.Feature`.
This feature type specifies a free-form feature.		
`GeomCopyFeat`	`pfcGeomCopyFeat`	Downcast of `pfcFeature.Feature`.
This feature type specifies a geometric copy feature.		
`GeomCurve`	`pfcGeometry`	`RevolvedSurface.GetProfile()`, `RuledSurface.GetProfile1()`,

Class	Package	Returned by
		RuledSurface.Get Profile2(), TabulatedCylinder.Get Profile()
This class provides information for a geometry curve.		
GeomExportFlags	pfcModel	pfcModel.GeomExport Flags _Create()
Stores extend-surface and Bezier options for use when exporting geometric information from a model.		
GeomExportInstructions	pfcModel	Base class; not returned.
Base class to classes used to export precise geometric information from a model.		
GraphFeat	pfcGraphFeat	Downcast of pfcFeature.Feature.
This feature type specifies a graph.		
GrooveFeat	pfcGrooveFeat	Downcast of pfcFeature.Feature.
This feature type specifies a groove.		
HoleFeat	pfcHoleFeat	Downcast of pfcFeature.Feature.
This feature type specifies a hole feature.		
IGES3DExport Instructions	pfcModel	pfcModel.IGES3DExport Instructions_Create()
Used to export a part or assembly in IGES format.		
IGESFileExport Instructions	pfcModel	pfcModel.IGESFileExport Instructions_Create()
Used to export a drawing in IGES format		
ImportFeat	pfcImportFeat	Downcast of pfcFeature.Feature.
This feature type specifies an import feature.		
IntegerOId	pfcObject	Base class; not returned.
This class specifies an integer identifier. For internal use only.		
InternalUDFFeat	pfcInternalUDFFeat	Downcast of pfcFeature.Feature.
This feature type is for internal use only.		
IntersectFeat	pfcIntersectFeat	Downcast of pfcFeature.Feature.
This feature type specifies an intersection.		
InventorExportInstruc tions	pfcModel	pfcModel.Inventor ExportInstructions _Create()
Used to export a part or assembly in Inventor format.		
IPMQuiltFeat	pfcIPMQuiltFeat	Downcast of pfcFeature.Feature.
Specifies an IPM Quilt feature.		
ISegmentFeat	pfcISegmentFeat	Downcast of pfcFeature.Feature.

Class	Package	Returned by
This feature type specifies an ideal segment.		
Layer	pfcLayer	Model.CreateLayer(). Also, by downcasting pfcModelItem.ModelItem.
This class describes a layer.		
Line	pfcGeometry	Downcast of pfcGeometry.Curve.
This class defines a line.		
LineStockFeat	pfcLineStockFeat	Downcast of pfcFeature.Feature.
This feature type specifies a line stock, which is used in the piping.		
LipFeat	pfcLipFeat	Downcast of pfcFeature.Feature.
This feature type specifies a lip feature.		
LocPushFeat	pfcLocPushFeat	Downcast of pfcFeature.Feature.
This feature type specifies a local push feature.		
ManualMillFeat	pfcManualMillFeat	Downcast of pfcFeature.Feature.
This feature type specifies a manual-mill feature.		
Material	pfcPart	Materials.get(), Part.GetCurrent Material(), Part.CreateMaterial(), Part.RetrieveMaterial()
This class provides information about a material.		
MaterialExport Instructions	pfcModel	pfcModel.MaterialExport Instructions_Create()
Used to export a material from a part.		
MaterialOId	pfcPart	
This class specifies the identifier of a Material. For internal use only.		
MaterialRemovalFeat	pfcMaterialRemovalFeat	Downcast of pfcFeature.Feature.
This feature type specifies a material removal feature.		
Materials	pfcPart	Materials.create(), PartListMaterials()
This data type is used to specify a list of materials.		
Matrix3D	pfcBase	Matrix3D.create(), Transform3D.GetMatrix()
This data type is used to describe a three-dimensional matrix.		
MeasureFeat	pfcMeasureFeat	Downcast of pfcFeature.Feature.
This feature type specifies a measure feature.		
MergeFeat	pfcMergeFeat	Downcast of pfcFeature.Feature.
This feature type specifies a merge feature.		
MFG	pfcMFG	Session.CreateMFG(). Also,

Class	Package	Returned by
		by downcasting `pfcModel.Model`.
This class describes a manufacturing object.		
`MFGCLExportInstructions`	`pfcModel`	Base class; not returned.
Base class to classes that export cutter-location files.		
`MFGFeatCLExport Instructions`	`pfcModel`	`pfcModel.MFGFeatCL ExportInstructions _Create()`
Used to export a cutter location (CL) file for one NC sequence in a manufacturing assembly.		
`MFGGatherFeat`	`pfcMFGGatherFeat`	Downcast of `pfcFeature.Feature`.
This feature type specifies a gather feature.		
`MFGMergeFeat`	`pfcMFGMergeFeat`	Downcast of `pfcFeature.Feature`.
This feature type specifies a manufacturing merge feature.		
`MFGOperCLExportInstruc tions`	`pfcModel`	`pfcModel.MFGOperCL ExportInstructions _Create ()`
Used to export a cutter location (CL) file for all the NC sequences in an operation.		
`MFGRefineFeat`	`pfcMFGRefineFeat`	Downcast of `pfcFeature.Feature`.
This feature type specifies a manufacturing refine feature.		
`MFGTrimFeat`	`pfcMFGTrimFeat`	Downcast of `pfcFeature.Feature`.
This feature type specifies a manufacturing trim feature.		
`MFGUseVolumeFeat`	`pfcMFGUseVolumeFeat`	Downcast of `pfcFeature.Feature`.
This feature type specifies a manufacturing use volumes feature.		
`MillFeat`	`pfcMillFeat`	Downcast of `pfcFeature.Feature`.
This feature type specifies a milling feature.		
`Model`	`pfcModel`	`Selection.GetSelModel()`, `Window.GetModel()`, `CompModelReplace. GetNewModel()`,`Cable Params FileInstructions .GetMdl()`
This class specifies the information about a model.		
`ModelDescriptor`	`pfcModel`	`pfcModelDescriptor. ModelDescriptor _Create()`, `Model.GetDescr()`
This class describes the descriptor for a model.		
`ModelDescriptors`	`pfcModel`	`ModelDescriptors .create()`, `Model.ListDeclared`

Class	Package	Returned by
		Models()
This data type is used to specify an array of model descriptors.		
ModelInfoExport Instructions	pfcModel	pfcModel.ModelInfo ExportInstructions _Create()
Used to export information about a model, including units information, features, and children.		
ModelItem	pfcModelItem	ModelItemOwner.GetItem ById(), ModelItemOwner.Get ItemByName(), Selection.GetSelItem(), FamColModelItem.Get RefItem()
This class defines a model item.		
ModelItemOId	pfcModelItem	pfcModel.ModelItemOId _Create()
This class specifies the owner of a model item. For internal use only.		
ModelItemOwner	pfcModelItem	Base class; not returned.
This class specifies the owner of a model item.		
ModelItems	pfcModelItem	ModelItems.create(), Feature.ListSubItems(), ModelItemOwner.List Items(), Layer.ListItems()
Specifies a list of model items.		
ModelItemType	pfcModelItem	ModelItemType.FromInt() or by using any of the static instances (e.g., ITEM_SURFACE)
This enumerated type lists the different kinds of model item.		
ModelItemTypes	pfcModelItem	ModelItemTypes.create(), Solid.ExcludeTypes()
Specifies a list of model item types.		
ModelOId	pfcModel	pfcModelOId.ModelOId _Create(), Model.GetOId()
This class describes a model owner. For internal use only.		
Models	pfcModel	Models.create(), Session.ListModels()
This data type is used to specify a list of models.		
ModelType	pfcModel	ModelType.FromInt() or by using any of the static instances (e.g., MDL_ASSEMBLY)
This enumerated type lists the supported model types.		
MoldFeat	pfcMoldFeat	Downcast of pfcFeature.Feature.
This feature type specifies a mold feature.		
NamedModelItem	pfcModelItem	Base class; not returned.
This class specifies the name of a model item.		

Class	Package	Returned by
NeckFeat	pfcNeckFeat	Downcast of pfcFeature.Feature.
This feature type specifies a neck feature.		
Note	pfcNote	pfcSolid.CreateNote()
Specifies the information for a note.		
Object	pfcObject	Base class; not returned.
Base classes to classes that represent PTC Creo Parametric objects.		
OffsetCurveDirection	pfcCurveFeat	OffsetCurveDirection .FromInt() or by using any of the static instances (e.g., OFFSET_SIDE_ONE)
This enumerated type specifies the direction of an offset.		
OffsetFeat	pfcOffsetFeat	Downcast of pfcFeature.Feature.
This feature type specifies an offset feature.		
OId	pfcObject	Child.GetOId()
This class defines the owner identifier object. For internal use only.		
OperationComponentFeat	pfcOperationComponent Feat	Downcast of pfcFeature.Feature.
This feature type specifies an operation component feature.		
OperationFeat	pfcOperationFeat	Downcast of pfcFeature.Feature.
This feature type specifies an operation feature.		
OptegraVisExport Instructions	pfcModel	pfcModel.OptegraVis ExportInstructions _Create()
Used to export a part or assembly in Optegra Vis format.		
Outline2D	pfcBase	Outline2D.create(), Contour.EvalOutline()
This data type is used to specify a two-dimensional outline.		
Outline3D	pfcBase	Outline3D.create(), Solid.GetGeomOutline(), Solid.EvalOutline()
This data type is used to specify a three-dimensional outline.		
Parameter	pfcModelItem	ParameterOwner.Create Param(), ParameterOwner.Get Param(), ParameterOwner.Select Param(), FamColParam.GetRef Param(), Parameters.get(), pfcModelItem.Create *ParamValue()
This class defines a parameter object.		
ParameterOwner	pfcModelItem	Base class; not returned.
This class defines a parameter owner object.		

Class	Package	Returned by
Parameters	pfcModelItem	Parameters.create()
Specifies a list of parameters.		
ParamOId	pfcModelItem	pfcModel.ParamOId _Create(), ParameterOwner .ListParams()
This class specifies the owner of a parameter. For internal use only.		
ParamType	pfcSession	ParamType.FromInt() or by using any of the static instances (e. g., DIMTOL_PARAM)
Enumeration of parameters types that is used to specify which parameters to display.		
ParamValue	pfcModelItem	BaseParameter.GetValue() , FamilyMember.GetCell(), ParamValues.get()
This class describes the value of the parameter.		
ParamValues	pfcModelItem	ParamValues.create()
This data type is used to specify an array of parameters.		
ParamValueType	pfcModelItem	ParamValueType.FromInt() or by using any of the static instances (e.g., PARAM_STRING)
This enumerated type lists the possible kinds of parameter value.		
Parent	pfcObject	Child.GetDBParent()
This class specifies a parent object.		
Part	pfcPart	Session.CreatePart()
This class defines the material data for a part.		
PatchFeat	pfcPatchFeat	Downcast of pfcFeature.Feature.
This feature type specifies a patch.		
PipeBranchFeat	pfcPipeBranchFeat	Downcast of pfcFeature.Feature.
This feature type specifies a pipe branch.		
PipeExtendFeat	pfcPipeExtendFeat	Downcast of pfcFeature.Feature.
This feature type specifies a pipe extension.		
PipeFeat	pfcPipeFeat	Downcast of pfcFeature.Feature.
This feature type specifies a pipe feature.		
PipeFollowFeat	pfcPipeFollowFeat	Downcast of pfcFeature.Feature.
This feature type specifies a follow feature, which is used in pipe routing.		
PipeJoinFeat	pfcPipeJoinFeat	Downcast of pfcFeature.Feature.
This feature type specifies a pipe join feature.		
PipeJointFeat	pfcPipeJointFeat	Downcast of pfcFeature.Feature.

Class	Package	Returned by
This feature type specifies a pipe joint.		
PipeLineFeat	pfcPipeLineFeat	Downcast of pfcFeature.Feature.
This feature type specifies a pipeline feature.		
PipePointToPointFeat	pfcPipePointToPointFeat	Downcast of pfcFeature.Feature.
This feature type specifies a point-to-point pipe feature.		
PipeSetStartFeat	pfcPipeSetStartFeat	Downcast of pfcFeature.Feature.
This feature type specifies a start feature, which is used in piping.		
PipeTrimFeat	pfcPipeTrimFeat	Downcast of pfcFeature.Feature.
This feature type specifies a trim feature, which is used in piping.		
Placement	pfcBase	Placement.FromInt() or by using any of the static instances (e.g., PLACE_INSIDE)
This enumerated type lists the possible placement types for points on contours.		
Plane	pfcGeometry	Downcast of pfcGeometry.Surface.
This class defines a plane.		
PlotInstructions	pfcModel	pfcModel.Plot Instructions_Create()
Used with to plot a part, drawing, or assembly.		
PlotPageRange	pfcModel	PlotPageRange.FromInt() or by using any of the static instances (e.g., PLOT_RANGE_ CURRENT)
This enumerated type specifies which pages to plot.		
PlotPaperSize	pfcModel	PlotPaperSize.FromInt() or by using any of the static instances (e.g., BSIZEPLOT)
This enumerated type specifies the size of the paper used for the plot.		
PlyFeat	pfcPlyFeat	Downcast of pfcFeature.Feature.
This feature type specifies a ply feature.		
Point	pfcGeometry	Downcast of pfcModelItem.ModelItem.
This class defines a point.		
Point2D	pfcBase	Point2D.create(), Outline2D.get()
This data type is used to specify a two-dimensional point.		
Point3D	pfcBase	Point3D.create(), Outline3D.get(), and additional methods that return multiple points
This data type is used to specify a three-dimensional point.		
Point3Ds	pfcBase	Point3Ds.create(),

Class	Package	Returned by
		`Polygon.GetVertices()`
Defines a list of three-dimensional points.		
`Polygon`	`pfcGeometry`	Downcast of `pfcGeometry.Curve`.
This class defines a polygon.		
`PositionFoldFeat`	`pfcPositionFoldFeat`	Downcast of `pfcFeature.Feature`.
This feature type specifies a position fold feature.		
`ProcessStepFeat`	`pfcProcessStepFeat`	Downcast of `pfcFeature.Feature`.
This feature type specifies a process step feature, which is used in the Manufacturing Process Planning for ASSEMBLIES module.		
`ProgramExport Instructions`	`pfcModel`	`pfcModel.Program ExportInstructions _Create()`
Used to export a program file for a part or assembly, which can be edited to change the model.		
`ProtrusionFeat`	`pfcProtrusionFeat`	Downcast of `pfcFeature.Feature`.
This feature type specifies a protrusion.		
`Quilt`	`pfcGeometry`	`Surface.GetOwnerQuilt()`. Also, by downcasting `pfcModelItem.ModelItem`.
This class defines a quilt.		
`RefDimension`	`pfcDimension`	Downcast of `pfcModelItem.ModelItem`.
This class describes a reference dimension.		
`RegenInstructions`	`pfcSolid`	`pfcSolid.Regen Instructions_Create()`
This class describes the regeneration instructions of a feature.		
`RelationExport Instructions`	`pfcModel`	`pfcModel.Relation ExportInstructions _Create()`
Used to export a list of the relations and parameters in a part or assembly.		
`RemoveSurfacesFeat`	`pfcRemoveSurfacesFeat`	Downcast of `pfcFeature.Feature`.
This feature type specifies a removed-surface feature.		
`RenderExport Instructions`	`pfcModel`	`pfcModel.Render ExportInstructions _Create()`
Used to export a part or assembly in `RENDER` format.		
`ReorderAfterOperation`	`pfcFeature`	`Feature.CreateReorder AfterOp()`
This class defines how to reorder the features in the regeneration order list.		
`ReorderBeforeOperation`	`pfcFeature`	`Feature.CreateReorder BeforeOp()`
This class determines how to move the features backward in the regeneration order list.		
`ReplaceSurfaceFeat`	`pfcReplaceSurfaceFeat`	Downcast of

Class	Package	Returned by
		pfcFeature.Feature.
This feature type specifies a replaced surface feature.		
ResumeOperation	pfcFeature	Feature.CreateResumeOp()
Specifies a resume operation for a feature.		
RetrieveModelOptions	pfcSession	pfcSession.RetrieveModelOptions_Create()
This class determines what information about the simplified representations in a model to retrieve.		
RevolvedSurface	pfcGeometry	Downcast of pfcGeometry.Surface.
This class defines a revolved surface.		
RibbonCableFeat	pfcRibbonCableFeat	Downcast of pfcFeature.Feature.
This feature type specifies a ribbon cable.		
RibbonExtendFeat	pfcRibbonExtendFeat	Downcast of pfcFeature.Feature.
This feature type specifies a ribbon extension.		
RibbonPathFeat	pfcRibbonPathFeat	Downcast of pfcFeature.Feature.
This feature type specifies a ribbon path feature.		
RibbonSolidFeat	pfcRibbonSolidFeat	Downcast of pfcFeature.Feature.
This feature type specifies a solid ribbon.		
RibFeat	pfcRibFeat	Downcast of pfcFeature.Feature.
This feature type specifies a rib feature.		
RipFeat	pfcRipFeat	Downcast of pfcFeature.Feature.
This feature type specifies a rip feature, which is used in the PTC Creo NC Sheetmetal module.		
RoundFeat	pfcRoundFeat	Downcast of pfcFeature.Feature.
This feature type specifies a round feature.		
RuledSurface	pfcGeometry	Downcast of pfcGeometry.Surface.
This class defines a ruled surface.		
SawFeat	pfcSawFeat	Downcast of pfcFeature.Feature.
This feature type specifies a saw feature.		
Selection	pfcSelect	Selections.get(),
This class contains the selection information.		
SelectionOptions	pfcSelect	pfcSelect.SelectionOptions_Create()
This class describes the selection options.		
Selections	pfcSelect	Selections.create(), Session.Select()
This data type is used to specify an array of selections.		

Class	Package	Returned by
Session	pfcSession	pfcGloba.GetProE Session()
This class defines the information about a session object.		
SessionActionListener_u	pfcSession	Base class; not returned.
Abstract base class that all user-defined SessionActionListener classes must extend.		
SessionActionListener	pfcSession	Base class; not returned.
Interface to be implemented by user-defined classes that respond to session events.		
SETExportInstructions	pfcModel	pfcModel.SETExport Instructions_Create()
This class defines a ruled surface.		
SETFeat	pfcSETFeat	Downcast of pfcFeature.Feature.
This feature type specifies a SET file.		
ShaftFeat	pfcShaftFeat	Downcast of pfcFeature.Feature.
This feature type specifies a shaft.		
SheetmetalClampFeat	pfcSheetmetalClampFeat	Downcast of pfcFeature.Feature.
This feature type specifies a sheetmetal clamp.		
SheetmetalConversion Feat	pfcSheetmetalConversion Feat	Downcast of pfcFeature.Feature.
This feature type specifies a conversion feature, which is used in the PTC Creo NC Sheetmetal module.		
SheetmetalCutFeat	pfcSheetmetal CutFeat	Downcast of pfcFeature.Feature.
This feature type specifies a sheetmetal cut feature.		
SheetmetalOptimizeFeat	pfcSheetmetalOptimize Feat	Downcast of pfcFeature.Feature.
This feature type specifies an optimize feature, used in the PTC Creo NC Sheetmetal module.		
SheetmetalPopulateFeat	pfcSheetmetalPopulate Feat	Downcast of pfcFeature.Feature.
This feature type specifies a populate feature, which is used in the PTC Creo NC Sheetmetal module.		
SheetmetalPunchPoint Feat	pfcSheetmetalPunch PointFeat	Downcast of pfcFeature.Feature.
This feature type specifies a punch point, which is used in the PTC Creo NC Sheetmetal module.		
SheetmetalShearFeat	pfcSheetmetalShearFeat	Downcast of pfcFeature.Feature.
This feature type specifies a sheetmetal shear feature.		
SheetmetalZoneFeat	pfcSheetmetalZone Feat	Downcast of pfcFeature.Feature.
This feature type specifies a sheetmetal zone.		
ShellFeat	pfcShellFeat	Downcast of pfcFeature.Feature.
This feature type specifies a shell.		
ShrinkageFeat	pfcShrinkageFeat	Downcast of

Class	Package	Returned by
		pfcFeature.Feature.
This feature type specifies a shrinkage feature, which is used in the Mold Design and Casting modules.		
ShrinkDimFeat	pfcShrinkDimFeat	Downcast of pfcFeature.Feature.
This feature type specifies a shrink dimension feature, which is used in the Mold Design and Casting modules.		
SilhouetteTrimFeat	pfcSilhouetteTrimFeat	Downcast of pfcFeature.Feature.
This feature type specifies a silhouette trim feature.		
STLASCIIExport Instructions	pfcModel	pfcModel.SLAASCIIExport Instructions_Create()
Used to export a part or assembly to an ASCII STL file.		
STLBinaryExport Instructions	pfcModel	pfcModel.SLABinaryEx port Instructions_Create()
Used to export a part or assembly in a binary STL file.		
SlotFeat	pfcSlotFeat	Downcast of pfcFeature.Feature.
This feature type specifies a slot.		
SMMFGApproachFeat	pfcSMMFGApproachFeat	Downcast of pfcFeature.Feature.
This feature type specifies an approach feature, which is used in sheetmetal manufacturing.		
SMMFGCosmeticFeat	pfcSMMFGCosmeticFeat	Downcast of pfcFeature.Feature.
This feature type specifies a cosmetic feature used in sheetmetal manufacturing.		
SMMFGCutFeat	pfcSMMFGCutFeat	Downcast of pfcFeature.Feature.
This feature type specifies a cut feature for sheetmetal manufacturing.		
SMMFGFormFeat	pfcSMMFGFormFeat	Downcast of pfcFeature.Feature.
This feature type specifies a form feature used in sheetmetal manufacturing.		
SMMFGMaterial RemoveFeat	pfcSMMFGMaterial RemoveFeat	Downcast of pfcFeature.Feature.
This feature type specifies a material removal feature, which is used in sheetmetal manufacturing.		
SMMFGOffsetFeat	pfcSMMFGOffsetFeat	Downcast of pfcFeature.Feature.
This feature type specifies a sheetmetal offset feature.		
SMMFGPunchFeat	pfcSMMFGPunchFeat	Downcast of pfcFeature.Feature.
This feature type specifies a sheetmetal punch feature.		
SMMFGShapeFeat	pfcSMMFGShapeFeat	Downcast of pfcFeature.Feature.

Class	Package	Returned by
This feature type specifies a sheetmetal shape feature.		
SMMFGSlotFeat	pfcSMMFGSlotFeat	Downcast of pfcFeature.Feature.
This feature type specifies a sheetmetal slot.		
Solid	pfcSolid	ComponentPath.GetLeaf()
This class defines a solid.		
SolidActionListener_u	pfcSolid	Base class; not returned.
Abstract base class that all user-defined SolidActionListener classes must extend.		
SolidActionListener	pfcSolid	Base class; not returned.
Interface to be implemented by user-defined classes that respond to solid events.		
SolidifyFeat	pfcSolidifyFeat	Downcast of pfcFeature.Feature.
This feature type specifies a solidify feature.		
SolidPipeFeat	pfcSolidPipeFeat	Downcast of pfcFeature.Feature.
This feature type specifies a solid pipe feature.		
SpinalBendFeat	pfcSpinalBendFeat	Downcast of pfcFeature.Feature.
This feature type specifies a spinal bend.		
Spline	pfcGeometry	Downcast of pfcGeometry.Curve.
This class defines a spline.		
SplinePoint	pfcGeometry	SplinePoints.get()
This class defines a spline point.		
SplinePoints	pfcGeometry	SplinePoints.create(), Spline.GetPoints()
This data type is used to specify an array of spline points.		
SplitFeat	pfcSplitFeat	Downcast of pfcFeature.Feature.
This feature type specifies a split feature.		
SplitSurfaceFeat	pfcSplitSurfaceFeat	Downcast of pfcFeature.Feature.
This feature type specifies a split-surface feature.		
SpoolFeat	pfcSpoolFeat	Downcast of pfcFeature.Feature.
This feature type specifies a spool.		
SpringBackFeat	pfcSpringBackFeat	Downcast of pfcFeature.Feature.
This feature type specifies a spring-back feature.		
StdColor	pfcBase	Session.SetTextColor(), GetRGBFromStdColor(), StdColor.FromInt(), or by using any of the static instances (e.g., COLOR_SHEETMETAL)
This enumerated type lists the supported color types.		
StdLineStyle	pfcBase	Session.SetLineStyle(),

Class	Package	Returned by
		StdLineStyle.FromInt(), or by using any of the static instances (e.g., LINE_SOLID)
This enumerated type lists the possible line styles.		
STEPExportInstructions	pfcModel	pfcModel.STEPExportInstructions_Create()
Used to export a part or assembly in STEP format.		
StringOId	pfcObject	Base class; not returned.
This class specifies a string identifier. For internal use only.		
SubHarnessFeat	pfcSubHarnessFeat	Downcast of pfcFeature.Feature.
This feature type specifies a subharness.		
SuppressOperation	pfcFeature	Feature.CreateSuppressOp()
This class defines a suppress operation.		
Surface	pfcGeometry	Surfaces.get(), Edge.GetSurface1(), GetSurface2(). Also, by downcasting pfcModelItem.ModelItem.
This class defines a surface.		
SurfaceModelFeat	pfcSurfaceModelFeat	Downcast of pfcFeature.Feature.
This feature type specifies a surface model.		
Surfaces	pfcGeometry	Surfaces.create(), Quilt.ListElements(), Surface.ListSameSurfaces().
This data type is used to describe an array of surfaces.		
SurfXYZData	pfcGeometry	Surface.Eval3DData()
Stores the results of a surface evaluation.		
TabulatedCylinder	pfcGeometry	Downcast of pfcGeometry.Surface.
This class defines a tabulated cylinder.		
Text	pfcGeometry	Downcast of pfcGeometry.Curve.
This class defines the text information.		
TextStyle	pfcBase	pfcBase.TextStyle_Create(), Text.GetStyle(). Note.GetStyle()
This class specifies the text attributes.		
ThickenFeat	pfcThickenFeat	Downcast of pfcFeature.Feature.
This feature type specifies a thicken feature, which is used in the PTC Creo NC Sheetmetal module.		
ThreadFeat	pfcThreadFeat	Downcast of pfcFeature.Feature.

Class	Package	Returned by
This feature type specifies a thread.		
Torus	pfcGeometry	Downcast of pfcGeometry.Surface.
This class defines a torus.		
TorusFeat	pfcTorusFeat	Downcast of pfcFeature.Feature.
This feature type is used for a torus.		
Transform3D	pfcBase	pfcBase.Transform3D _Create(), ComponentPath.Get Transform(), CoordSystem.Get CoordSys(), TransformedSurface. GetCoordSys(), View.GetTransform(), Window.GetTransform()
This class provides information about the transformation.		
TransformedSurface	pfcGeometry	Downcast of pfcGeometry.Surface.
This class defines a transformed surface.		
TurnFeat	pfcTurnFeat	Downcast of pfcFeature.Feature.
This feature type specifies a turn feature, which is used in the manufacturing module.		
TwistFeat	pfcTwistFeat	Downcast of pfcFeature.Feature.
This feature type specifies a twist feature.		
UDFClampFeat	pfcUDFClampFeat	Downcast of pfcFeature.Feature.
This feature type specifies a UDF clamp.		
UDFFeat	pfcUDFFeat	Downcast of pfcFeature.Feature.
This feature type specifies a UDF feature.		
UDFNotchFeat	pfcUDFNotchFeat	Downcast of pfcFeature.Feature.
This feature type specifies a UDF notch feature.		
UDFPunchFeat	pfcUDFPunchFeat	Downcast of pfcFeature.Feature.
This feature type specifies a UDF punch feature.		
UDFRmdtFeat	pfcUDFRmdtFeat	Downcast of pfcFeature.Feature.
This feature type specifies a UDF for rapid mold design.		
UDFThreadFeat	pfcUDFThreadFeat	Downcast of pfcFeature.Feature.
This feature type specifies a UDF thread feature.		
UDFWorkRegionFeat	pfcUDFWorkRegionFeat	Downcast of pfcFeature.Feature.
This feature type specifies a UDF work region feature.		

Class	Package	Returned by
UDFZoneFeat	pfcUDFZoneFeat	Downcast of pfcFeature.Feature.
This feature type specifies a UDF zone feature.		
UICommand	pfcCommand	Session.UICreate Command()
An action-listener object for menu commands.		
UICommandAction Listener_u	pfcCommand	Base class; not returned.
Abstract base class that all user-defined UICommandActionListener classes must extend.		
UICommandActionListener	pfcCommand	Base class; not returned.
Interface to be implemented by user-defined classes that respond to UI command events.		
UnbendFeat	pfcUnbendFeat	Downcast of pfcFeature.Feature.
This feature type specifies an unbend feature, which is used in the PTC Creo NC Sheetmetal module.		
UserFeat	pfcUserFeat	Downcast of pfcFeature.Feature.
This feature type specifies a user feature.		
UVParams	pfcBase	UVParams.create(), EdgeEvalData.Get Point*() , Selection.GetParams(), SurfXYZData.GetParams()
This data type is used to specify UV parameters.		
UVVector	pfcBase	UVVector.create(), EdgeEvalData. GetDerivative*()
This data type is used to specify a UV-vector.		
VDAExportInstructions	pfcModel	pfcModel.CATIAExport Instructions_Create(), VDAExportInstructions _Create()
Used to export a part or assembly in VDA format.		
VDAFeat	pfcVDAFeat	Downcast of pfcFeature.Feature.
This feature type specifies a VDA file.		
Vector2D	pfcBase	Vector2D.create()
This data type is used to specify a two-dimensional vector.		
Vector3D	pfcBase	Vector3D.create(), Vectors3D.get() and additional methods that return information about geometric curves.
This data type is used to specify a three-dimensional vector.		
Vector3Ds	pfcBase	Vector3Ds.create()
This data type is used to specify a list of three-dimensional vectors.		
View	pfcView	ViewOwner.GetView() ViewOwner.SaveView() ViewOwner.Retrieve

Class	Package	Returned by
		View() Views.get()
This class specifies information about a view.		
ViewOId	pfcView	pfcView.ViewOId_ Create()
This class specifies the owner of a view. For internal use only.		
ViewOwner	pfcView	Base class; not returned.
This class describes the owner of the view.		
Views	pfcView	Views.create(), ViewOwner.ListViews()
This data type is used to specify an array of views.		
VolSplitFeat	pfcVolSplitFeat	Downcast of pfcFeature.Feature.
This feature type specifies a split-volume feature.		
WallFeat	pfcWallFeat	Downcast of pfcFeature.Feature.
This feature type specifies a wall, which is used in the PTC Creo NC Sheetmetal module.		
WaterLineFeat	pfcWaterLineFeat	Downcast of pfcFeature.Feature.
This feature type specifies a waterline feature.		
WeldEdgePrepFeat	pfcWeldEdgePrepFeat	Downcast of pfcFeature.Feature.
This feature type specifies a preparation edge, which is used in the Welding Design module.		
WeldFilletFeat	pfcWeldFilletFeat	Downcast of pfcFeature.Feature.
This feature type specifies a welding fillet.		
WeldGrooveFeat	pfcWeldGrooveFeat	Downcast of pfcFeature.Feature.
This feature type specifies a weld groove.		
WeldingRodFeat	pfcWeldingRodFeat	Downcast of pfcFeature.Feature.
This feature type specifies a welding rod.		
WeldPlugSlotFeat	pfcWeldPlugSlotFeat	Downcast of pfcFeature.Feature.
This feature type specifies a plug-slot feature, which is used in the Welding Design module.		
WeldSpotFeat	pfcWeldSpotFeat	Downcast of pfcFeature.Feature.
This feature type specifies a welding spot.		
Window	pfcWindow	Session.GetCurrent Window(), Session.CreateModel Window(), Session.OpenFile(), Session.GetWindow(), Windows.get()
This class describes the attributes of a window.		
WindowOId	pfcWindow	pfcWindow.WindowOId

Class	Package	Returned by
		_Create()
This class specifies a window identifier. For internal use only.		
Windows	pfcWindow	Session.ListWindows(), Windows.create()
This data type is used to specify an array of windows.		
WorkcellFeat	pfcWorkcellFeat	Downcast of pfcFeature.Feature.
This feature type specifies a workcell.		
XBadArgument	pfcExceptions	Created, thrown in J-Link code.
This exception is thrown when you specify an invalid argument.		
XBadGetParamValue	pfcExceptions	Created, thrown in J-Link code.
This exception is thrown when you specify an invalid parameter type.		
XBadOutlineExcludeType	pfcExceptions	Created, thrown in J-Link code.
This exception is thrown when you specify an invalid outline exclusion type.		
XInAMethod	pfcExceptions	Base class of most J-Link exceptions.
This exception is thrown when you specify an invalid method name.		
XInvalidEnumValue	pfcExceptions	Created, thrown in J-Link code.
This exception is thrown when you specified an invalid enumerated value.		
XPFC	pfcExceptions	Base class of most J-Link exceptions.
This exception is thrown when a general usage error occurs.		
XSequenceTooLong	pfcExceptions	Created, thrown in J-Link code.
This exception is thrown when the sequence length exceeds the maximum allowable size.		
XStringTooLong	pfcExceptions	Created, thrown in J-Link code.
This exception is thrown when the specified string exceeds the maximum allowable length.		
XToolkitError	pfcExceptions	Created, thrown in J-Link code.
This exception is thrown when there is a toolkit error.		
XUnimplemented	pfcExceptions	Created, thrown in J-Link code.
This exception is thrown when there is a call to a function that is unimplemented.		
XUnknownModelExtension	pfcExceptions	Created, thrown in J-Link code.
This exception is thrown when you specify an invalid model extension.		
ZoneFeat	pfcZoneFeature	Downcast of pfcFeature.Feature
This feature type specifies a zone feature.		

Index

A

B

C

D

E

J

K

L

M

N

O

P

R

S

T

U

V

W