PTC®

Creo® Parametric 2.0
Web.Link™ User's Guide

Contents

The Basics of Web.Link

The Web.Link Online Browser

Session Objects

Selection

Models

Drawings

Dimensions and Parameters

Relations

Simplified Representations

Task Based Application Libraries

Graphics

External Data

Windchill Connectivity APIs

Digital Rights Management

Geometry Traversal

Geometry Representations

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 Web.Link, a tool that links the World Wide Web (WWW, or Web) to Creo Parametric, enabling you to use the Web as a tool to automate and streamline parts of your engineering process.

Note

Web.Link is supported only with Creo Parametric. It is not supported with the other Creo applications.

Audience

This manual is intended for experienced Creo Parametric users who know HyperText Markup Language (HTML) and JavaScript.

Prerequisites

This manual assumes you have the following knowledge:

* Creo Parametric
* HTML
* JavaScript

Documentation

The documentation for Web.Link includes the following:

* *Web.Link User's Guide*
* An online browser that describes the syntax of the Web.Link functions and provides a link to the online version of this manual. This includes the Embedded Browser-based *Web.Link User's Guide*.

Conventions

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

Convention	Description
UPPERCASE	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, Web.Link methods, names of dialog box buttons, and Creo Parametric commands.
Monospace (Courier)	Code samples appear in courier font like this. Web.Link 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 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 Creo applications are installed, for example, `C:\Program Files\PTC\Creo 1.0.`
`<creo_weblink_loadpoint>`	The location where the Web.Link application is installed, that is, `<creo_loadpoint>\Common Files\<datecode>\weblink.`

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.

Introduction

This section describes the fundamentals of Web.Link. For information on how to set up your environment, see the section <u>Setting Up Web.Link</u>.

Overview

Web.Link links the World Wide Web to Creo Parametric, enabling you to use the Web as a tool to automate and streamline parts of your engineering process.

Pro/Web.Link in Pro/ENGINEER Wildfire has been simplified and enhanced with new capabilities by the introduction of an embedded web browser in Pro/ENGINEER. Web.Link pages can be loaded directly into the embedded browser of Creo Parametric.

Creo Parametric is always connected to the contents of the embedded browser and there is no need to start or connect to Creo Parametric from Web.Link compared to the old version of Pro/Web.Link, where web pages had to try to start or connect to Creo Parametric.

Creo Parametric supports the embedded browser versions of Web.Link on Windows platforms using Microsoft Internet Explorer. PTC has retired the Netscape-based version Pro/Web.Link.

Note

From Creo Parametric 1.0 M010 onwards, the default value of the configuration option `windows_browser_type` *is* `ie_in_sep_process`*. If you specify the default value, Creo Parametric will use the Internet Explorer browser engine in a separate child process initiated from the Creo Parametric process.*

Although Web.Link still supports the old 'PWL' style methods, PTC does not recommend the use of PWL. Instead you should use JavaScript version of 'PFC' (Parametric Foundation Classes), which is the basis for the J-Link interface as well. This guide provides instructions on how to switch from 'PWL' to 'PFC'.

The embedded browser version of Web.Link is as shown in the following figure.

File ▾ | Model | Analysis | Annotate | Render | Manikin | Tools | View | Applications

Operations ▾ | Get Data ▾ | Component ▾ | Datum ▾ | Cut & Surface ▾ | Modifiers ▾ | Model Display ▾ | Model Intent ▾ | Investigate ▾

file:///D:/WebLinkExample/html/pfcComponentFeatExamples.html

Web.Link examples | 3DModelSpace

Web.Link Component Feature Examples

Replacing components

This example function ("replaceBoltsInAssembly") replaces one instance of a component with another in an assembly. The current model must be an assembly. It should contain a subcomponent "bolt.prt" with instances "slot7_8" and "phillips7_8".

Replace Bolts

Accessing component assembly constraints

This example function ("highlightConstraints") highlights each of the pairs of constraints for a selected component visually, and lists the type and offset values for the component. The current model must be an assembly. Select a component feature from the window to show its constraints.

Highlight Constraints

Assembling components

This example function ("assembleByDatums") auto-assembles the component entered below by aligning

Done

Smart

Loading Application Web Pages

To load and run a Web.Link application web page:

1. Ensure that Web.Link is set up to run properly. See the section <u>Setting Up Web.Link</u> for more details.

2. Type the URL for the page directly into the embedded browser address bar, follow a link in the embedded browser to a Web.Link enabled page, or load the web page into the embedded browser via the navigation tools in the Creo Parametric navigator. The Creo Parametric navigator contains the following navigation tools:

   • Folders—(Default) Provides navigation of the local file system, the local network, and Internet data.

   • Favorites—Contains user-selected Web locations (bookmarks) and paths to Creo Parametric objects, database locations, or other points of interest.

   • Search—Provides search capability for objects in the data management system.

   > ### *Note*
   >
   > *The **Search** option appears when you declare a Windchill system as your primary data management system.*

   • History—Provides a record of Creo Parametric objects you have opened and Web locations you have visited. Click the **History** icon on the browser toolbar to add the option to the Creo Parametric navigator.

   • Connections—Provides access to connections and built-in PTC solutions, such as Pro/COLLABORATE, PartsLink, and the PTC User area.

3. Depending upon how the application web page is constructed, the Web.Link code may run upon loading of the page, or may be invoked by changes in the forms and components embedded in the web page.

4. Navigate to a new Web.Link enabled page using the same techniques defined above.

> ### *Note*
>
> *The Web.Link pages do not stay resident in the Creo Parametric session; the application code is only accessible while the page is loaded in the embedded browser.*

Object Types

Web.Link is made up of a number classes in many packages. The following are the seven main class types:

• `Creo Parametric-Related Classes`—Contain unique methods and properties that are directly related to the functions in Creo Parametric. See the section <u>Creo Parametric-Related Classes</u> for more information.

• `Compact Data Classes`—Classes containing data needed as arguments to some Web.Link methods. See the section, <u>Compact Data Classes</u>, for additional information.

• `Union Classes`—A class with a potential for multiple types of values. See the section <u>Unions</u> for additional information.

• `Sequence Classes`—Expandable arrays of objects or primitive data types. See the section <u>Sequences</u> for more information.

• `Array Classes`—Arrays that are limited to a certain size. See the section <u>Arrays</u> for more information.

• `Enumeration Classes`—Defines enumerated types. See the section <u>Enumeration Classes</u> for more information.

• `Module-Level Classes`—Contains static methods used to initialize certain Web.Link objects. See the <u>Module-Level Classes</u> section for more information.

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

Creo Parametric-Related Classes

The `Creo Parametric-Related Classes` contain methods that directly manipulate objects in Creo Parametric. Examples of these objects include models, features, and parameters.

Initialization

You cannot construct one of these objects explicitly using JavaScript syntax. Objects that represent Creo

Parametric objects cannot be created directly but are returned by a `Get` or `Create` method.

For example, `pfcBaseSession.CurrentModel` returns a `pfcModel` object set to the current model and `pfcParameterOwner.CreateParam` returns a newly created `Parameter` object for manipulation.

Properties

Properties within Creo Parametric-related objects are directly accessible. Some attributes that have been designated as read can only be accessed, but not modified by Web.Link.

Methods

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

```
var window;
window.SetBrowserSize (0.0);// The window has not yet
                            //been initialized.

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

The following calls are legal:

```
var session = pfcCreat ("MpfcCOMGlobal").GetProESession();
var window = session.CurrentWindow;
   // You have initialized the window object.
window.SetBrowserSize (0.0);
window.Repaint();
```

Inheritance

Many Creo Parametric related objects inherit methods from other interfaces. JavaScript allows you to invoke any method or property assigned to the object or its parent. You can directly invoke any property or method of a subclass, provided you know that the object belongs to that subclass.

For example, a component feature could use the methods and properties as follows:

- `pfcObject`
- `pfcChild`
- `pfcActionSource`
- `pfcModelItem`
- `pfcFeature`
- `pfcComponentFeat`

Compact Data Classes

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

Initialization

You can create instances of these classes using a static create method. In order to call a static method on the class, you must first instantiate the appropriate class object:

```
var instrs = pfcCreate ("pfcBOMExportInstructions").Create();
```

Properties

Properties within compact data related classes are directly accessible. Some attributes that have been designated as read can only be accessed, but not modified by Web.Link.

Methods

You must invoke non-static methods from the object in question and you must first initialize that object.

Inheritance

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

Unions

Unions are classes containing potentially several different value types. Every union has a discriminator property with the pre-defined name `discr`. This method returns a value identifying the type of data that the union object holds. For each union member, a separate property is used to access the different data types. It is illegal to attempt to read any property except the one that matches the value returned from the discriminator. However, any property that switches the discriminator to the new value type can be modified.

The following is an example of a Web.Link union:

```
class ParamValue
{
    pfcParamValueType           discr;
    string                       StringValue;
    integer                      IntValue;
    boolean                      BoolValue;
    number                       DoubleValue;
    integer                      NoteId;
};
```

Sequences

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

Initialization

You can create instances of these classes directly by instantiating the appropriate class object:

```
var models = pfcCreate ("pfcModels");
```

Properties

The readonly `Count` attribute identifies how many members are currently in the sequence.

Methods

Sequence objects always contain the same methods. Use the following methods to access the contents of the sequence:

* `Item()`
* `Set()`
* `Append()`
* `Insert()`
* `InsertSeq()`
* `Remove()`
* `Clear()`

Inheritance

Sequence classes do not inherit from any other Web.Link classes. Therefore, you cannot use sequence objects as arguments where any other type of Web.Link object is expected, including other types of sequences. For example, if you have a list of `pfcModelItems` that happen to be features, you cannot use the sequence as if it were a sequence of `pfcFeatures`.

To construct the array of features, you must insert each member of the `pfcModelItems` list into the new `pfcFeatures` list.

Exceptions

If you try to get or remove an object beyond the last object in the sequence, an exception will be thrown.

Arrays

Arrays are groups of primitive types or objects of a specified size. An array can be one or two dimensional. The online reference documentation indicates the exact size of each array class.

Initialization

You can create instances of these classes directly by instantiating the appropriate class object:

```
var point = pfcCreate ("pfcPoint3D");
```

Methods

Array objects always contain the same methods: Item and Set, used to access the contents of the array.

Inheritance

Array classes do not inherit from any other Web.Link classes.

Exceptions

If you try to access an object that is not within the size of the array, an exception will be thrown.

Enumeration Classes

In Web.Link, an enumeration class defines a limited number of values which correspond to the members of the enumeration. Each value represents an appropriate type and may be accessed by name. In the `pfcFeatureType`enumeration class the value `FEATTYPE_HOLE` represents a Hole feature in Creo Parametric. Enumeration classes in Web.Link generally have names of the form `pfcXYZType` or `pfcXYZStatus`.

Initialization

You can create instances of these classes directly by instantiating the appropriate class object:

```
var modelType = pfcCreate ("pfcModelType");
```

Attributes

An enumeration class is made up of constant integer properties. The names of these properties are all uppercase and describe what the attribute represents. For example:

- `PARAM_INTEGER`—A value in the `pfcParamValueType` enumeration class that is used to indicate that a parameter stores an integer value.

- `ITEM_FEATURE`—An value in the `pfcModelItemType` enumeration class that is used to indicate that a model item is a feature.

An enumeration class always has an integer vale named `<type>_nil`, which is one more than the highest acceptable numerical value for that enumeration class.

Module-Level Classes

Some modules in Web.Link have one class that contains special static functions used to create and access some of the other classes in the package. These module classes have the naming convention: "M"+ the name of the module, as in `MpfcSelect`.

Initialization

You can create instances of these classes directly by instantiating the appropriate class object:

```
var session = pfcCreate ("MpfcCOMGlobal").GetProESession();
```

Properties

Module-level classes do not have any accessible attributes.

Methods

Module-level classes contain only static methods used for initializing certain Web.Link objects.

Inheritance

Module-level classes do not inherit from any other Web.Link classes.

Programming Considerations

The items in this section introduce programming tips and techniques used for programming Web.Link in the embedded browser.

Creating Platform Independent Code

PTC recommends constructing web pages in a way that will work for Windows.

Non-Web.Link JavaScript code should be designed for use in both architectures.

The mechanism for Web.Link library on Windows is `ActiveX` on Internet Explorer. References to `ActiveX` should be encapsulated in a platform-independent manner.

To do this, add or import a header to the start of the script based on embedded JavaScript as follows:

```
function pfcIsWindows ()
{
  if (navigator.appName.indexOf ("Microsoft") != -1)
    return true;
  else
    return false;
}

function pfcCreate (className)
{
  if (!pfcIsWindows())
  netscape.security.PrivilegeManager.enablePrivilege("UniversalXPConnect");

  if (pfcIsWindows())
    return new ActiveXObject ("pfc."+className);
  else
    {
      ret = Components.classes ["@ptc.com/pfc/" +
            className + ";1"].createInstance();

      return ret;
    }
}
```

Use the function `pfcIsWindows()` to determine the platform on which the browser is running.

Use the function `pfcCreate()` in any situation where a Web.Link object or class must be initialized using its string name. For convenience, these and other useful Web.Link utilities are provided in a file in the example set located at:

```
<creo_weblink_loadpoint>/weblinkexamples /jscript/pfcUtils.js
```

Variable Typing

Although JavaScript is not strongly typed, the interfaces in Web.Link do expect variables and arguments of certain types. The following primitive types are used by Web.Link and its methods:

- `boolean`—a JavaScript Boolean, with valid values true and false.
- `integer`—a JavaScript Number of integral type.
- `number`—a JavaScript Number; it need not be integral.
- `string`—a JavaScript String object or string literal.

These variable types, as well as all explicit object types, are listed in the Web.Link documentation for each property and method argument.

PTC recommends that Web.Link applications ensure that values passed to Web.Link classes are of the correct type.

Optional arguments and tags

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

For example, the `pfcModelItemOwner.ListItems()` method takes an optional *Type* argument.

```
pfcModelItems ListItems (/*optional*/ pfcModelItemType  Type);
```

You can pass the JavaScript value keyword `void null` in place of any such optional argument. The Web.Link

methods that take optional arguments provide default handling for `void null` parameters which is described in the online documentation.

Note

You can only pass `void null` in place of arguments that are shown in the documentation to be "optional".

Optional Returns for Web.Link Methods

Some methods in Web.Link have an optional return. Usually these correspond to lookup methods that may or may not find an object to return. For example, the `pfcBaseSession.GetModel()` method returns an optional model:

```
/*optional*/ pfcModel GetModel(string Name,
                               pfcModelType Type);
```

Web.Link might return void null in certain cases where these methods are called. You must use appropriate value checks in your application code to handle these situations.

Parent-Child Relationships Between Web.Link Objects

Some Web.Link objects inherit from either the module `pfcObject.Parent` or `pfcObject.Child`. These interfaces are used to maintain a relationship between the two objects. This has nothing to do with Java or JavaScript inheritance. In Web.Link, the Child is owned by the Parent.

Property Introduced:

- **pfcChild.DBParent**

The `pfcChild.DBParent` property returns the owner of the child object. The application developer must know the expected type of the parent in order to use it in later calls. The following table lists parent/child relationships in Web.Link.

Parent	Child
pfcSession	pfcModel
pfcSession	pfcWindow
pfcModel	pfcModelItem
pfcSolid	pfcFeature
pfcModel	pfcParameter
pfcModel	pfcExternalDataAccess
pfcPart	pfcMaterial
pfcModel	pfcView
pfcModel2D	pfcView2D
pfcSolid	pfcXSection
pfcSession	pfcDll (Creo Parametric TOOLKIT)
pfcSession	pfcApplication (J-Link)

Run-Time Type Identification in Web.Link

Web.Link and the JavaScript language provides several methods to identify the type of an object.

Many Web.Link classes provide read access to a type enumerated class. For example, the `pfcFeature` class has a `pfcFeature.FeatType` property, returning a `pfcFeatureType` enumeration value representing the type of the feature. Based upon the type, a user can recognize that the `pfcFeature` object is actually a particular subtype, such as `pfcComponentFeat`, which is an assembly component.

Exceptions

Web.Link signals error conditions via exceptions. Exceptions may be caught and handled via a try/catch block surrounding Web.Link code. If exceptions are not caught, they may be ignored by the web browser altogether, or may present a debug dialog box to the user.

Descriptions for Web.Link exceptions may be accessed in a platform-independent way using the JavaScript utility function `pfcGetExceptionDescription()`, included in the example files in `pfcUtils.js`. This function returns the full exception description as `[Exception type]; [additional details]`. The exception type will be the module and exception name, for example, `pfcExceptions::pfcXToolkitCheckoutConflict`.

The additional details will include details which were contained in the exception when it was thrown by the PFC layer, like conflict descriptions for exceptions caused by server operations and error details for exceptions generated during drawing creation.

PFC Exceptions

The methods that make up Web.Link's public interface may throw the PFC exceptions. The following table describes some of these exceptions.

Exception	Purpose
`pfcExceptions::pfcXBadExternalData`	An attempt to read contents of an external data object which has been terminated.
`pfcExceptions::pfcXBadGetArgValue`	Indicates attempt to read the wrong type of data from the `pfcArgValue` union.
`pfcExceptions::pfcXBadGetExternalData`	Indicates attempt to read the wrong type of data from the `pfcExternalData` union.
`pfcExceptions::pfcXBadGetParamValue`	Indicates attempt to read the wrong type of data from the `pfcParamValue` union.
`pfcExceptions::pfcXBadOutlineExcludeType`	Indicates an invalid type of item was passed to the outline calculation method.
`pfcExceptions::pfcXCannotAccess`	The contents of a Web.Link object cannot be accessed in this situation.
`pfcExceptions::pfcXEmptyString`	An empty string was passed to a method that does not accept this type of input.
`pfcExceptions::pfcXInvalidEnumValue`	Indicates an invalid value for a specified enumeration class.
`pfcExceptions::pfcXInvalidFileName`	Indicates a file name passed to a method was incorrectly structured.
`pfcExceptions::pfcXInvalidFileType`	Indicates a model descriptor contained an invalid file type for a requested operation.
`pfcExceptions::pfcXInvalidModelItem`	Indicates that the item requested to be used is no longer usable (for example, it may have been deleted).
`pfcExceptions::pfcXInvalidSelection`	Indicates that the `pfcSelection` passed is invalid or is missing a needed piece of information. For example, its component path, drawing view, or parameters.
`pfcExceptions::pfcXJLinkApplicationException`	Contains the details when an attempt to call code in an external J-Link application failed due to an exception.
`pfcExceptions::pfcXJLinkApplicationInactive`	Unable to operate on the requested `pfcJLinkApplication` object because it has been shut down.
`pfcExceptions::pfcXJLinkTaskNotFound`	Indicates that the J-Link task with the given name could not be found and run.
`pfcExceptions::pfcXModelNotInSession`	Indicates that the model is no longer in session; it may have been erased or deleted.

pfcExceptions::pfcXNegativeNumber	Numeric argument was negative.
pfcExceptions::pfcXNumberTooLarge	Numeric argument was too large.
pfcExceptions::pfcXProEWasNotConnected	The Creo Parametric session is not available so the operation failed.
pfcExceptions::pfcXSequenceTooLong	Sequence argument was too long.
pfcExceptions::pfcXStringTooLong	String argument was too long.
pfcExceptions::pfcXUnimplemented	Indicates unimplemented method.
pfcExceptions::pfcXUnknownModelExtension	Indicates that a file extension does not match a known Creo Parametric model type.

Creo Parametric TOOLKIT Errors

The pfcExceptions::pfcXToolkitError exception provides access to error codes from Creo Parametric TOOLKIT functions that Web.Link uses internally and to the names of the functions returning such errors. pfcExceptions::pfcXToolkitError is the exception you are most likely to encounter because Web.Link is built on top of Creo Parametric TOOLKIT. The following table lists the integer values that can be returned by the pfcXToolkitError.GetErrorCode() method and shows the corresponding Creo Parametric TOOLKIT constant that indicates the cause of the error. Each specific pfcExceptions::pfcXToolkitError exception is represented by an appropriately named child class. The child class name (for example pfcExceptions::pfcXToolkitGeneralError, will be returned by the error's description property.

pfcXToolkitError Child Class	Creo Parametric TOOLKIT Error	#
pfcExceptions::pfcXToolkitGeneralError	PRO_TK_GENERAL_ERROR	-1
pfcExceptions::pfcXToolkitBadInputs	PRO_TK_BAD_INPUTS	-2
pfcExceptions::pfcXToolkitUserAbort	PRO_TK_USER_ABORT	-3
pfcExceptions::pfcXToolkitNotFound	PRO_TK_E_NOT_FOUND	-4
pfcExceptions::pfcXToolkitFound	PRO_TK_E_FOUND	-5
pfcExceptions::pfcXToolkitLineTooLong	PRO_TK_LINE_TOO_LONG	-6
pfcExceptions::pfcXToolkitContinue	PRO_TK_CONTINUE	-7
pfcExceptions::pfcXToolkitBadContext	PRO_TK_BAD_CONTEXT	-8
pfcExceptions::pfcXToolkitNotImplemented	PRO_TK_NOT_IMPLEMENTED	-9
pfcExceptions::pfcXToolkitOutOfMemory	PRO_TK_OUT_OF_MEMORY	-10
pfcExceptions::pfcXToolkitCommError	PRO_TK_COMM_ERROR	-11
pfcExceptions::pfcXToolkitNoChange	PRO_TK_NO_CHANGE	-12
pfcExceptions::pfcXToolkitSuppressedParents	PRO_TK_SUPP_PARENTS	-13
pfcExceptions::pfcXToolkitPickAbove	PRO_TK_PICK_ABOVE	-14
pfcExceptions::pfcXToolkitInvalidDir	PRO_TK_INVALID_DIR	-15
pfcExceptions::pfcXToolkitInvalidFile	PRO_TK_INVALID_FILE	-16
pfcExceptions::pfcXToolkitCantWrite	PRO_TK_CANT_WRITE	-17
pfcExceptions::pfcXToolkitInvalidType	PRO_TK_INVALID_TYPE	-18
pfcExceptions::pfcXToolkitInvalidPtr	PRO_TK_INVALID_PTR	-19
pfcExceptions::pfcXToolkitUnavailableSection	PRO_TK_UNAV_SEC	-20

pfcExceptions::pfcXToolkitInvalidMatrix	PRO_TK_INVALID_MATRIX	-21
pfcExceptions::pfcXToolkitInvalidName	PRO_TK_INVALID_NAME	-22
pfcExceptions::pfcXToolkitNotExist	PRO_TK_NOT_EXIST	-23
pfcExceptions::pfcXToolkitCantOpen	PRO_TK_CANT_OPEN	-24
pfcExceptions::pfcXToolkitAbort	PRO_TK_ABORT	-25
pfcExceptions::pfcXToolkitNotValid	PRO_TK_NOT_VALID	-26
pfcExceptions::pfcXToolkitInvalidItem	PRO_TK_INVALID_ITEM	-27
pfcExceptions::pfcXToolkitMsgNotFound	PRO_TK_MSG_NOT_FOUND	-28
pfcExceptions::pfcXToolkitMsgNoTrans	PRO_TK_MSG_NO_TRANS	-29
pfcExceptions::pfcXToolkitMsgFmtError	PRO_TK_MSG_FMT_ERROR	-30
pfcExceptions::pfcXToolkitMsgUserQuit	PRO_TK_MSG_USER_QUIT	-31
pfcExceptions::pfcXToolkitMsgTooLong	PRO_TK_MSG_TOO_LONG	-32
pfcExceptions::pfcXToolkitCantAccess	PRO_TK_CANT_ACCESS	-33
pfcExceptions::pfcXToolkitObsoleteFunc	PRO_TK_OBSOLETE_FUNC	-34
pfcExceptions::pfcXToolkitNoCoordSystem	PRO_TK_NO_COORD_SYSTEM	-35
pfcExceptions::pfcXToolkitAmbiguous	PRO_TK_E_AMBIGUOUS	-36
pfcExceptions::pfcXToolkitDeadLock	PRO_TK_E_DEADLOCK	-37
pfcExceptions::pfcXToolkitBusy	PRO_TK_E_BUSY	-38
pfcExceptions::pfcXToolkitInUse	PRO_TK_E_IN_USE	-39
pfcExceptions::pfcXToolkitNoLicense	PRO_TK_NO_LICENSE	-40
pfcExceptions::pfcXToolkitBsplUnsuitableDegree	PRO_TK_BSPL_UNSUITABLE_DEGREE	-41
pfcExceptions::pfcXToolkitBsplNonStdEndKnots	PRO_TK_BSPL_NON_STD_END_KNOTS	-42
pfcExceptions::pfcXToolkitBsplMultiInnerKnots	PRO_TK_BSPL_MULTI_INNER_KNOTS	-43
pfcExceptions::pfcXToolkitBadSrfCrv	PRO_TK_BAD_SRF_CRV	-44
pfcExceptions::pfcXToolkitEmpty	PRO_TK_EMPTY	-45
pfcExceptions::pfcXToolkitBadDimAttach	PRO_TK_BAD_DIM_ATTACH	-46
pfcExceptions::pfcXToolkitNotDisplayed	PRO_TK_NOT_DISPLAYED	-47
pfcExceptions::pfcXToolkitCantModify	PRO_TK_CANT_MODIFY	-48
pfcExceptions::pfcXToolkitCheckoutConflict	PRO_TK_CHECKOUT_CONFLICT	-49
pfcExceptions::pfcXToolkitCreateViewBadSheet	PRO_TK_CRE_VIEW_BAD_SHEET	-50
pfcExceptions::pfcXToolkitCreateViewBadModel	PRO_TK_CRE_VIEW_BAD_MODEL	-51
pfcExceptions::pfcXToolkitCreateViewBadParent	PRO_TK_CRE_VIEW_BAD_PARENT	-52
pfcExceptions::pfcXToolkitCreateViewBadType	PRO_TK_CRE_VIEW_BAD_TYPE	-53
pfcExceptions::pfcXToolkitCreateViewBadExplode	PRO_TK_CRE_VIEW_BAD_EXPLODE	-54
pfcExceptions::pfcXToolkitUnattachedFeats	PRO_TK_UNATTACHED_FEATS	-55
pfcExceptions::pfcXToolkitRegenerateAgain	PRO_TK_REGEN_AGAIN	-56

pfcExceptions::pfcXToolkitDrawingCreateErrors	PRO_TK_DWGCREATE_ERRORS	-57
pfcExceptions::pfcXToolkitUnsupported	PRO_TK_UNSUPPORTED	-58
pfcExceptions::pfcXToolkitNoPermission	PRO_TK_NO_PERMISSION	-59
pfcExceptions::pfcXToolkitAuthenticationFailure	PRO_TK_AUTHENTICATION_FAILURE	-60
pfcExceptions::pfcXToolkitAppNoLicense	PRO_TK_APP_NO_LICENSE	-92
pfcExceptions::pfcXToolkitAppExcessCallbacks	PRO_TK_APP_XS_CALLBACKS	-93
pfcExceptions::pfcXToolkitAppStartupFailed	PRO_TK_APP_STARTUP_FAIL	-94
pfcExceptions::pfcXToolkitAppInitializationFailed	PRO_TK_APP_INIT_FAIL	-95
pfcExceptions::pfcXToolkitAppVersionMismatch	PRO_TK_APP_VERSION_MISMATCH	-96
pfcExceptions::pfcXToolkitAppCommunicationFailure	PRO_TK_APP_COMM_FAILURE	-97
pfcExceptions::pfcXToolkitAppNewVersion	PRO_TK_APP_NEW_VERSION	-98

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

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

Setting Up Web.Link

This section describes instructions to setup Web.Link.

See the *Creo Parametric Installation and Administration Guide* for information on how to install Web.Link.

Supported Hardware

On Windows you can use Web.Link in the embedded browser.

Supported Software

Web.Link in the embedded browser supports the browsers supported by Creo Parametric, specified at http://www.ptc.com/support/creo.htm.

Security on Windows

Operations performed using Web.Link in the embedded browser can read and write information in the Creo Parametric session and from the local disk. Because of this, Web.Link in Creo Parametric uses three levels of security:

- Web.Link code only functions in web pages loaded into the Creo Parametric embedded browser. Pages containing Web.Link code will not work if the user browses to them using external web browsers.
- Web.Link is disabled by default using a Creo Parametric configuration option.
- The Web.Link `ActiveX` control has been created as not safe for scripting. This requires that security settings be enabled in Internet Explorer, allowing only certain sites access to the Web.Link methods and objects.

Enabling Web.Link

The configuration option `web_enable_javascript` controls whether the Creo Parametric session is able to load the `ActiveX` control. Set `web_enable_javascript` to `ON` to enable Web.Link, and set it to `OFF` to disable it. The default value for the Creo Parametric session is `OFF`. If Web.Link applications are loaded into the embedded browser with the configuration option turned off, the applications will throw a `pfcXNotConnectedToProE` exception.

Setting Up Browser Security

Follow the procedure below to change the security settings:

1. In Internet Explorer, select **Tools ▸ Internet Options**. Click the **Security** tab as shown in the following figure.
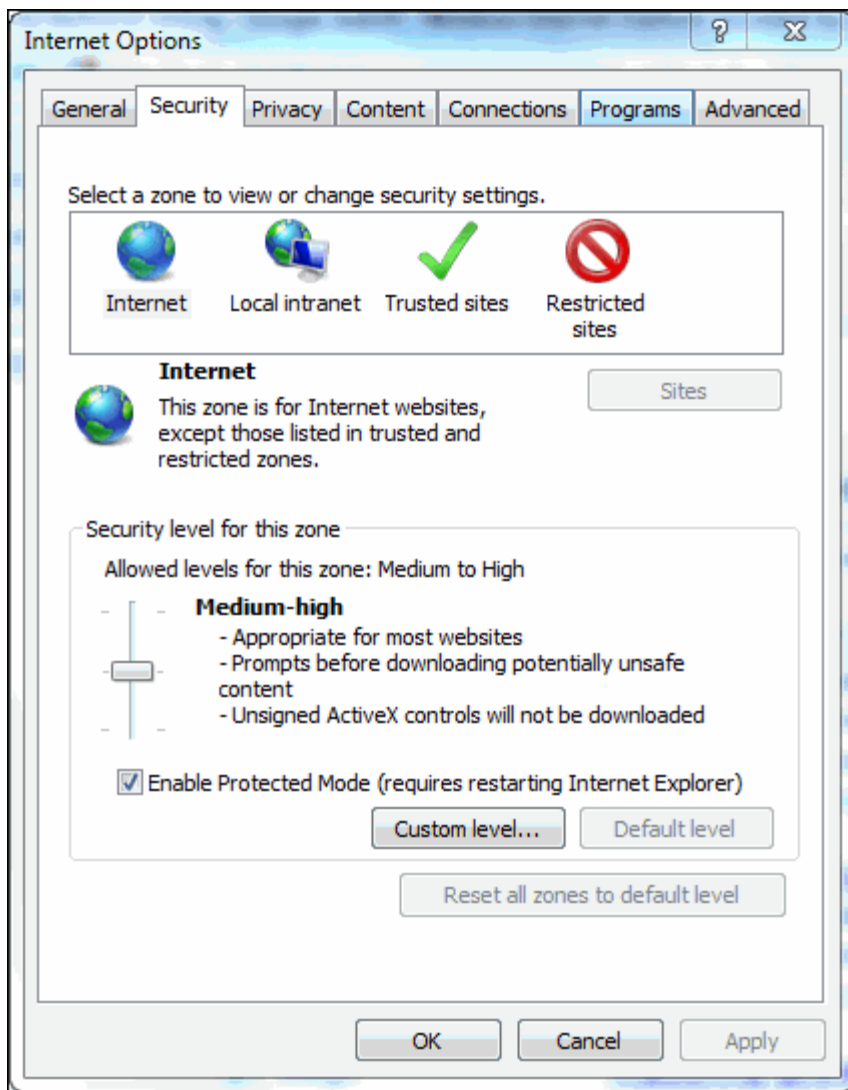
2. Select a zone for which you want to change security settings.
3. Click **Custom Level…**.
4. Change the setting for **Initialize and Script ActiveX controls not marked as safe** under **ActiveX controls and plugins** to `Enable`, as shown in the following figure.
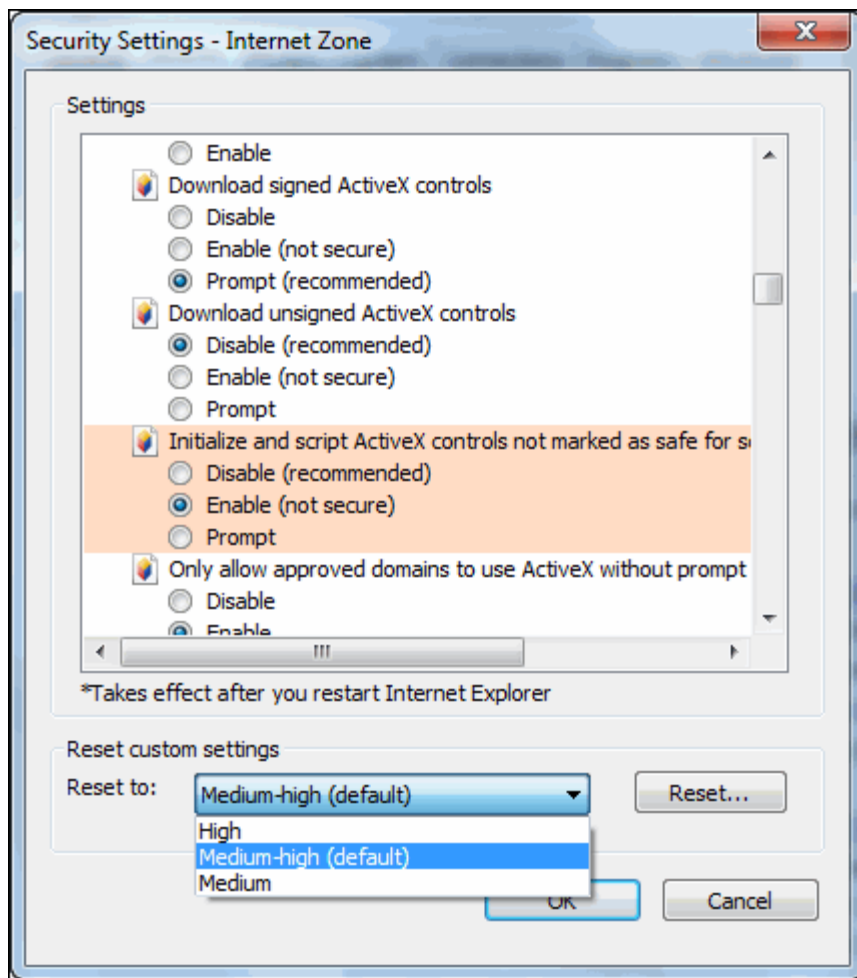
5. Change the setting for **Active Scripting** under **Scripting** to `Enable` as shown in the following figure.
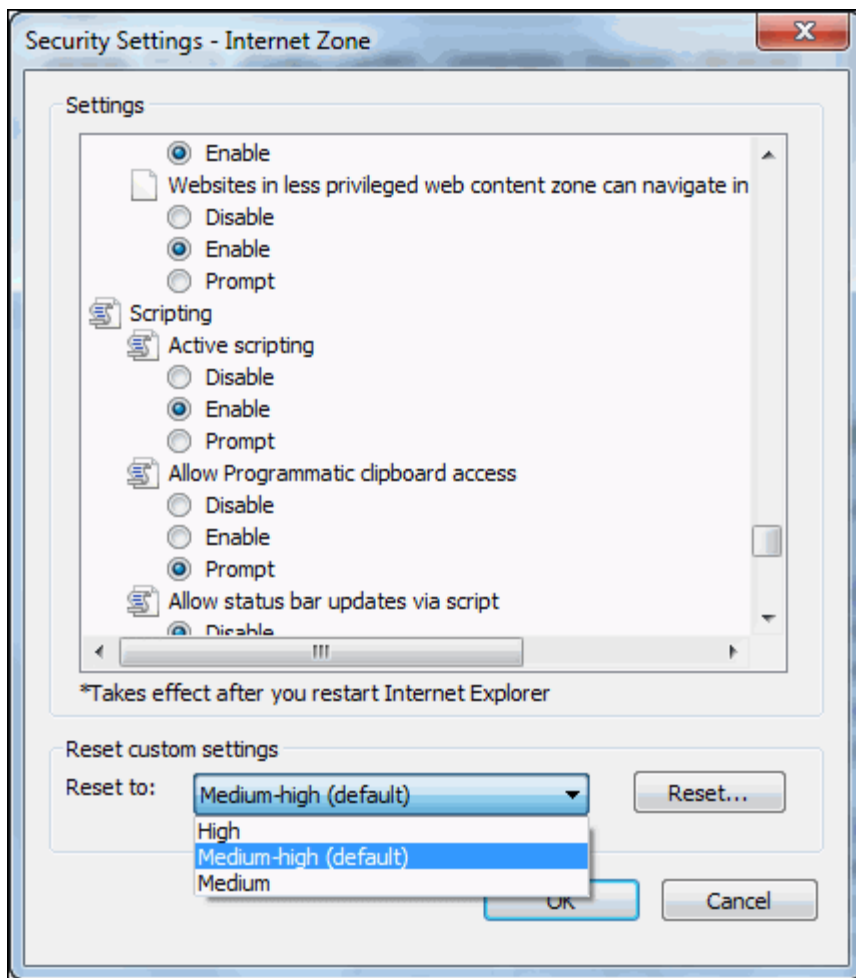
Add and Remove Sites to Security Zones

Follow the procedure below to add sites to the security zones:

1. In Internet Explorer, select **Tools   Internet Options**.
2. Click the **Security** tab.
3. Select the security zone to which you want to add sites.
4. Click **Sites...**.
5. Click **Advanced...**, this option is available only for the local intranet.
6. Enter the name of site.
7. Click **Add**.

The site is added to the security zone as shown in the figure below:
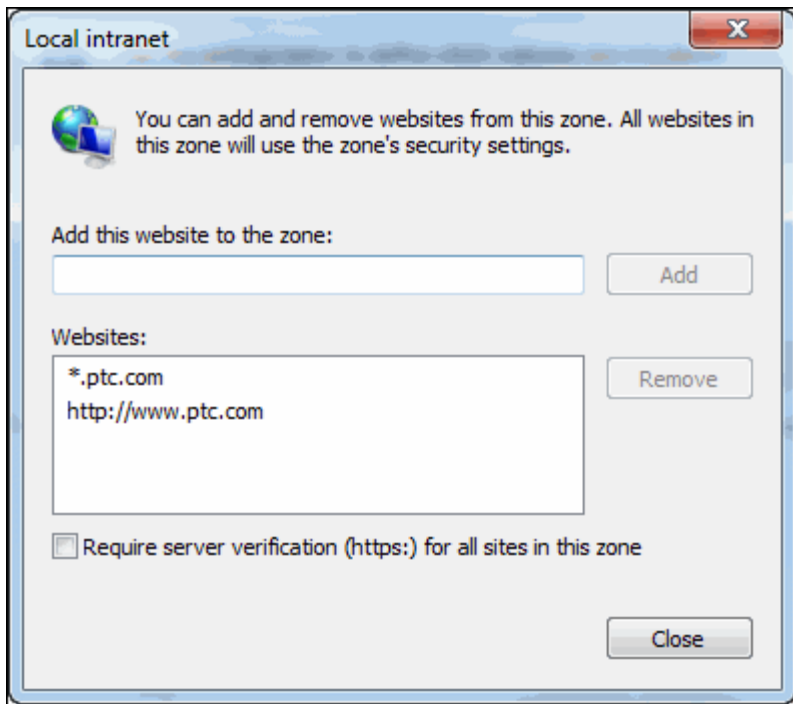
Enabling Security Settings

To run Web.Link in the embedded browser security set the following in Microsoft Internet Explorer:

- Allow scripting of `ActiveX` controls not marked as safe
- Allow active scripting

These security features can be set to the following values:

- `Disable`—The activity is not permitted. Attempting to load a Web.Link page will result in the following exception:

  ```
  "Automation server can't create object"
  ```

- `Prompt`—Each time the browser loads a web page that tries to access Web.Link methods and objects, you are prompted to allow the interaction activity as shown in the following figure.
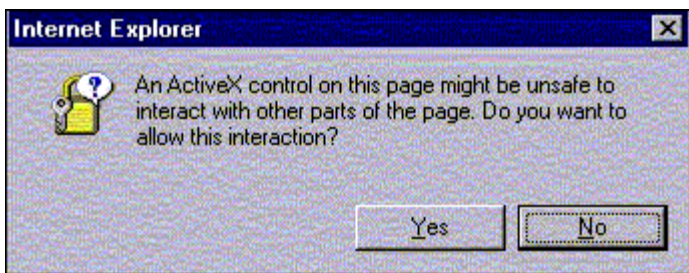
- `Enable`—The interaction activity is always permitted

Script security can be independently assigned to four domains:

- `Intranet`—The organization's local intranet, including all access via `file://` `URLs` and selected internal web servers.
- `Trusted sites`—Web sites designated as trusted.
- `Restricted sites`—Web sites designated as untrusted.
- `Internet`—All other sites accessed via the Internet.

Advanced Setup

```
ActiveX
```

The control must be registered with Windows in order to be available to Creo Parametric.

Note

To register the `ActiveX` control for the first time you must have the permission to create new keys under the Windows registry key `HKEY_CLASSES_ROOT`.

If the configuration option `web_enable_javascript` is enabled, `ActiveX` control is registered automatically when Creo Parametric starts.

Creo Parametric does not unregister the `ActiveX` control automatically. If the control is already registered, Creo Parametric will not register the DLL again, unless it is a different version of the application.

For multiple installations of Creo Parametric on a particular machine, you need to manually unregister the `ActiveX` control, to ensure that Creo Parametric locates the correct version of the installation. To unregister the `ActiveX` control manually use the following command:

```
C:\winnt\system32\regsvr32 /u [/s] <creo_loadpoint>\Common Files\<datecode>\<machine type>\obj\pfcscom.dll
```

Use the `/s` flag to unregister without displaying a confirmation dialog box.

Running Web.Link On Your Machine

To run Web.Link on your machine do the following:

- Edit your `config.pro` file to enable Web.Link on the local machine.
- Optionally setup browser security for your local intranet settings.
- Run Creo Parametric.

Load web pages containing Web.Link functions and application code into the embedded browser of Creo Parametric.

Troubleshooting

The following table describes some common errors and how to resolve them.

Error	Explanation
`pfcXNotConnectedToProE` exception	The web page was loaded into a web browser that is not the Creo Parametric embedded web browser. OR The web page was loaded into the embedded web browser but the configuration option `web_enable_javascript` is not `on`.
Nothing happens when JavaScript is invoked; or "Automation server can't create object."	The Internet Explorer or Mozilla security is not configured to allow the web page to run Web.Link, or the Web.Link license is not configured.

The Basics of Web.Link

This section explains the basics of Web.Link.

Examples using Web.Link

Most of the examples include a standard header that includes some standard options on every page. The header has buttons to start, connect, and stop Creo Parametric. Some file operations are also provided. The header is a JavaScript file loaded using the following lines of HTML:

```
<script src = "wl_header.js">
document.writeln ("Error loading Web.Link header<p>");
</script>
```

The first line includes the header file in your source file. If an error occurs (for example, the header file is not in the current directory), the second line causes an error message to be displayed.

Note

To avoid redundancy, the header is included but not explicitly listed in the examples themselves.

The following sections describe the header files used in the example programs:

*   `wl_header.js`—Contains only the JavaScript functions. This file is included in the head of the HTML page.

JavaScript Header

The contents of the header file `wl_header.js` are as follows:

```
try
{
  if (!pfcIsWindows())
    netscape.security.PrivilegeManager.enablePrivilege("UniversalXPConnect");

  wpwl = pfcGetScript ();
  document.pwl = wpwl;
  wpwlc = wpwl.GetPWLConstants ();
  document.pwlc = wpwlc;
  wpwlf = wpwl.GetPWLFeatureConstants ();
  document.pwlf = wpwlf;
}
catch (err)
{
  alert ("Exception caught: "+pfcGetExceptionType (err));
}

function WlProEStart()
{
  if (document.pwl == void null)
    {
      alert("Connect failed.");
      return ;
    }
}

function WlProEConnect()
  //Connect to a running Creo Parametric session.
{
  WlProEStart();
}

function WlModelOpen()
  //Open a Creo Parametric model.
{
  if (!pfcIsWindows())
    netscape.security.PrivilegeManager.enablePrivilege("UniversalXPConnect");
  if (document.main.ModelName.value == "")
    return ;
  var ret = document.pwl.pwlMdlOpen(document.main.ModelName.value,
      document.main.ModelPath.value, true);
  if (!ret.Status)
    {
```

```
        if (ret.ErrorCode == -4 && document.main.ModelPath.value == "")
          return ;
        else
          {
            alert("pwlMdlOpen failed (" + ret.ErrorCode + ")");
            return ;
          }
      }
}

function WlModelRegenerate()
  //Regenerate the Creo Parametric model.
{
  if (!pfcIsWindows())
    netscape.security.PrivilegeManager.enablePrivilege("UniversalXPConnect");

  var ret =
          document.pwl.pwlMdlRegenerate(document.main.ModelNameExt.value);
  if (!ret.Status)
    {
      alert("pwlMdlRegenerate failed (" + ret.ErrorCode + ")");
      return ;
    }
}

function WlModelSave()
  //Save a Creo Parametric model.
{
  if (!pfcIsWindows())
    netscape.security.PrivilegeManager.enablePrivilege("UniversalXPConnect");
  var ret = document.pwl.pwlMdlSaveAs(document.main.ModelNameExt.value,
            void null, void null);
  if (!ret.Status)
    {
      alert("pwlMdlSaveAs failed (" + ret.ErrorCode + ")");
      return ;
    }
}

function WlModelSaveAs()
  //Save a Creo Parametric model under a new name.
{
  if (!pfcIsWindows())
    netscape.security.PrivilegeManager.enablePrivilege("UniversalXPConnect");
  var NewPath = document.main.NewPath.value;
  var NewName = document.main.NewName.value;
  if (NewPath == "")
    {
      NewPath = void null;
    }
  if (NewName == "")
    {
      NewName = void null;
    }
  var ret = document.pwl.pwlMdlSaveAs(document.main.ModelNameExt.value,
       NewPath, NewName);
  if (!ret.Status)
    {
      alert("pwlMdlSaveAs failed (" + ret.ErrorCode + ")");
      return ;
    }
}

function WlWindowRepaint()
  //Repaint the active window.
{
  if (!pfcIsWindows())
    netscape.security.PrivilegeManager.enablePrivilege("UniversalXPConnect");
  var get_ret = document.pwl.pwlWindowActiveGet();
  if (!get_ret.Status)
    {
      alert("pwlWindowActiveGet failed (" + get_ret.ErrorCode + ")");
      return ;
    }
```

```
    /* You can also repaint the active window using -1 as the window
       identifier. */
    var ret = document.pwl.pwlWindowRepaint(parseInt(get_ret.WindowID));
    if (!ret.Status)
      {
        alert("pwlWindowRepaint failed (" + ret.ErrorCode + ")");
        return ;
      }
}

// Define the form with all the buttons to perform the above actions.
document.writeln("<form name='main'>");

document.writeln("<hr>");
document.writeln("<h4>Main Controls</h4>");
document.writeln("<p>");
document.writeln("<center>");
document.writeln("<input type='button' value='Start Pro/ENGINEER' onclick='WlProEStart()'>");
document.writeln("<input type='button' value='Connect to Pro/ENGINEER' onclick='WlProEConnect()'>");
document.writeln("<p>");
document.writeln("Path: <input type='text' name='ModelPath' onchange='WlModelOpen()'>");
document.writeln("<spacer size=20>");
document.writeln("Model: <input type='text' name='ModelName'
onchange='WlModelOpen()'>");
document.writeln("<spacer size=20>");
document.writeln("<input type='button' value='Open Model'
onclick='WlModelOpen()'>");
document.writeln("<p>");
document.writeln("<table>");
document.writeln("<tr>");
document.writeln("<td><center>Model:</center></td>");
document.writeln("<td><center>New Path:</center></td>");
document.writeln("<td><center>New Name:</center></td></tr>");
document.writeln("<tr>");
document.writeln("<td><input type='text' name='ModelNameExt'></td>");
document.writeln("<td><input type='text' name='NewPath'></td>");
document.writeln("<td><input type='text' name='NewName'></td></tr>");
document.writeln("</table>");
document.writeln("<input type='button' value='Regenerate Model'
onclick='WlModelRegenerate()'>");
document.writeln("<spacer size=10>");document.writeln("<input type='button' value='Save Model'
onclick='WlModelSave()'>");
document.writeln("<spacer size=10>");
document.writeln("<input type='button' value='Save Model As'
onclick='WlModelSaveAs()'>");
document.writeln("<p>");
document.writeln("<input type='button' value='REPAINT  SCREEN' onclick='WlWindowRepaint()'>");
document.writeln("</center>");
document.writeln("<hr>");
document.writeln("</form>");
```

The following figure shows the header as it appears in the browser.

Error Codes

Error codes are used to test the conditions in your code. You can use the Web.Link error codes as constants. This enables you to use symbolic constants in the form `pwl.ErrorCode`, which is a good coding practice. For example:

```
if (!ret.Status && ret.ErrorCode != pwl.PWL_E_NOT_FOUND)
```

The class `pfcPWLConstants` contains the following error codes:

- PWL_NO_ERROR
- PWL_EXEC_NOT_FOUND
- PWL_NO_ACCESS
- PWL_GENERAL_ERROR
- PWL_BAD_INPUTS
- PWL_USER_ABORT
- PWL_E_NOT_FOUND
- PWL_E_FOUND
- PWL_LINE_TOO_LONG
- PWL_CONTINUE
- PWL_BAD_CONTEXT
- PWL_NOT_IMPLEMENTED
- PWL_OUT_OF_MEMORY
- PWL_COMM_ERROR
- PWL_NO_CHANGE
- PWL_SUPP_PARENTS
- PWL_PICK_ABOVE
- PWL_INVALID_DIR
- PWL_INVALID_FILE
- PWL_CANT_WRITE
- PWL_INVALID_TYPE
- PWL_INVALID_PTR
- PWL_UNAV_SEC
- PWL_INVALID_MATRIX

- PWL_INVALID_NAME

- PWL_NOT_EXIST

- PWL_CANT_OPEN

- PWL_ABORT

- PWL_NOT_VALID

- PWL_INVALID_ITEM

- PWL_MSG_NOT_FOUND

- PWL_MSG_NO_TRANS

- PWL_MSG_FMT_ERROR

- PWL_MSG_USER_QUIT

- PWL_MSG_TOO_LONG

- PWL_CANT_ACCESS

- PWL_OBSOLETE_FUNC

- PWL_NO_COORD_SYSTEM

- PWL_E_AMBIGUOUS

- PWL_E_DEADLOCK

- PWL_E_BUSY

- PWL_NOT_IN_SESSION

- PWL_INVALID_SELSTRING

Model and File Management

This section describes the Web.Link functions that enable you to access and manipulate models.

Model Management

Functions Introduced:

- **pfcScript.pwlMdlCurrentGet()**
- **pfcScript.pwlSessionMdlsGet()**
- **pfcScript.pwlMdlDependenciesGet()**
- **pfcScript.pwlMdlInfoGet()**
- **pfcScript.pwlMdlIntralinkInfoGet()**
- **pfcScript.pwlMdlRegenerate()**

To retrieve the current model in session, call the function `pfcScript.pwlMdlCurrentGet()`. The syntax is as follows:

```
pwlMdlCurrentGet();
Additional return fields:
    string MdlNameExt;   // The full name of the
                         // current model
```

The function `pfcScript.pwlSessionMdlsGet()` provides a list of all the models with the specified type that are in session. The syntax is as follows:

```
pwlSessionMdlsGet (
    integer    MdlType   // The type of model to list.
                         // Use parseInt with this
                         // argument.
);
Additional return fields:
    integer    NumMdls;  // The number of models in
                         // the list.
    string MdlNameExt[]; // The full names of the
                         // models of the specified
                         // type.
```

The valid model types are as follows:

- PWL_ASSEMBLY

- PWL_PART

- PWL_DRAWING

- PWL_2DSECTION

- PWL_LAYOUT

- PWL_DWGFORM

- PWL_MFG

- PWL_REPORT

- PWL_MARKUP

- PWL_DIAGRAM

The function `pfcScript.pwlMdlDependenciesGet()` lists all the top-level dependencies for the specified model (that is, all the models upon which the given model depends). If any of these models is an assembly, the function does not list the dependencies for that assembly. The syntax is as follows:

```
pwlMdlDependenciesGet (
    string MdlNameExt    // The full name of the model
                         // whose dependencies you want
);
Additional return fields:
    integer NumMdls;        // The number of models in the
                            // returned array
    string  MdlNameExt[]; // The array of models upon
                            // which the specified model
                            // depends
```

For the specified model, the function `pfcScript.pwlMdlInfoGet()` provides the generic name and model type. The syntax is as follows:

```
pwlMdlInfoGet (
    string NameExt        // The full name of the model
);
Additional return fields:
    string  ImmediateGeneralName; // The immediate
                                  // general name
    string  TopGenericName;       // The top-level
                                  // generic name of
                                  // the model
    integer MdlType;              // The model type
```

The function `pfcScript.pwlMdlIntralinkInfoGet()` provides Pro/INTRALINK^{TM} information about the specified model. Given the name of the model, this function provides the version, revision, branch, and release level. The syntax is as follows:

```
pwlMdlIntralinkInfoGet (
    string NameExt         // The full name of the model
);
Additional return fields:
    string Version;        // The version
    string Revision;       // The model revision
    string Branch;         // The branch
    string ReleaseLevel;   // The release level
```

Note

Markups and sections are not supported.

The `pfcScript.Script.pwlMdlRegenerate` function regenerates the specified model. The syntax is as follows:

```
pwlMdlRegenerate (
    string MdlNameExt    // The full name of the model
);
```

File Management Operations

Functions Introduced:

- **pfcScript.pwlMdlOpen()**

pfcScript.pwlMdlSaveAs()

- **pfcScript.pwlMdlErase()**
- **pfcScript.pwlMdlRename()**

The function `pfcScript.pwlMdlOpen()` retrieves the specified model. The syntax is as follows:

```
pwlMdlOpen (
    string  MdlNameExt,       // The full name of the
                              // model.
    string  Path,             // The full directory
                              // path to the model.
    boolean DisplayInWindow   // If this is true, display
                              // the retrieved model in
                              // a Creo Parametric window.
);
Additional return field:
    integer    WindowID;      // The identifier of the
                              // window in which the
                              // model is displayed.
```

The *Path* argument is the full directory path to the model. If the model is already in memory, the function ignores this argument. If you try to open a model that is already in memory and supply an invalid *Path*, `pfcScript.Script.pwlMdlOpen` successfully opens the model anyway.

If *Path* is an empty string, the function uses the default Creo Parametric search path, including the current Workspace if Pro/INTRALINK is being used.

You can use the function `pfcScript.Script.pwlMdlOpen` to open a family table instance by specifying the name of the generic instance as the *MdlNameExt* argument.

The function `pfcScript.pwlMdlSaveAs()` saves the model in memory to disk, under a new name. The syntax is as follows:

```
pwlMdlSaveAs (
    string OrigNameExt,   // The original name of the
                          // model, including the
                          // extension
    string NewPath,       // The new path to the model
    string NewNameExt     // The new name of the model,
                          // including the extension
);
```

Note

Web.Link does not currently support the Creo Parametric methods of saving subcomponents under new names, so `NewPath` *and* `NewNameExt` *are optional.*

The function `pfcScript.pwlMdlErase()` removes the specified model from memory. The syntax is as follows:

```
pwlMdlErase (
    string MdlNameExt   // The full name of the model
                        // to erase from memory
);
```

To rename a model in memory and on disk, use the function `pfcScript.Script.pwlMdlRename`. Note that the model must be in the current directory for the model to be renamed on disk. The syntax is as follows:

```
pwlMdlRename (
    string OrigNameExt,   // The original name of the
                          // model, including the
                          // extension
    string NewNameExt     // The new name of the model,
                          // including the extension
);
```

Checking Out Pro/INTRALINK Objects

Function introduced:

- **pfcScript.pwlObjMdlsCheckOut()**

To check out Pro/INTRALINK objects from the Commonspace to a Workspace, call the function `pfcScript.pwlObjMdlsCheckOut`. The syntax is as follows:

```
    pwlObjMdlsCheckOut (
        boolean LinkOrCopy,      // If this is true, check
                                 // out the object as a
                                 // link.
        string  WorkspaceName,   // The name of the
                                 // Workspace to which the
                                 // objects are checked out.
        integer NumObj,          // The number of objects
                                 // to check out. Use
                                 // parseInt with this
                                 // argument.
        string  ObjNames[],      // An array of names (and
                                 // extensions) of the
                                 // objects to check out.
        integer ObjVersions[],   // The object versions.
                                 // If this is null, the
                                 // default action is to
                                 // use the latest versions.
                                 // Use parseInt with this
                                 // argument.
        integer RelCriteria      // The relationship
                                 // criteria. Use parseInt
                                 // with this argument.
);
```

The *RelCriteria* argument identifies which dependents should be checked out along with the *ObjNames[]*. The possible values are as follows:

- 1—Include all the dependents.
- 2—Include the required dependents only.
- 3—Do not include any dependents.

Model Example

The following example shows how to use the Web.Link model functions.

```
<html>
<head>
<title>Web.Link Models Test</title>
<script src="../jscript/pfcUtils.js">
</script>
<script src="../jscript/wl_header.js">
document.writeln ("Error loading Web.Link header!");
</script>

<script language="JavaScript">

function WlModelOpenWindowless()
//      Open a Creo Parametric model without a window.
{
    var ret = document.pwl.pwlMdlOpen(document.open.ModelName.value,
                                      document.open.ModelPath.value, false);
    if (!ret.Status)
    {
        alert("pwlMdlOpen failed (" + ret.ErrorCode + ")");
        return ;
    }
}

function WlModelErase()
//      Erase a model from memory.
{
    var ret = document.pwl.pwlMdlErase(document.erase.ModelNameExt.value);
    if (!ret.Status)
    {
        alert("pwlMdlErase failed (" + ret.ErrorCode + ")");
        return ;
    }
}

function WlModelRename()
//      Rename a model.
{
```

```
    var ret = document.pwl.pwlMdlRename(document.rename.ModelNameExt.value,
                    document.rename.NewNameExt.value);
    if (!ret.Status)
    {
        alert("pwlMdlRename failed (" + ret.ErrorCode + ")");
        return ;
    }
}

function WlModelsList()
//      List all the models in the current session.
{
 //    var ModelType = document.pwlc.eval(
   //                        document.list_mdls.ModelType.options[
    //                            document.list_mdls.ModelType.selectedIndex].value);
     var ModelType = parseInt (document.list_mdls.ModelType.options[
                            document.list_mdls.ModelType.selectedIndex].value);
    var ret = document.pwl.pwlSessionMdlsGet(parseInt(ModelType));
    if (!ret.Status)
    {
        alert("pwlSessionMdlsGet failed (" + ret.ErrorCode + ")");
        return ;
    }
    document.list_mdls.ModelNameExts.value = "";
    for (var i = 0; i < ret.NumMdls; i++)
    {
        document.list_mdls.ModelNameExts.value += ret.MdlNameExt.Item(i) + "\n";
    }
}

function WlModelGetCurrent()
//      Get the current model.
{
    var ret = document.pwl.pwlMdlCurrentGet();
    if (!ret.Status)
    {
        alert("pwlMdlCurrentGet failed (" + ret.ErrorCode + ")");
        return ;
    }
    document.get_cur.ModelNameExt.value = ret.MdlNameExt;
}

function WlModelGetDependencies()
//      Get the dependencies of a model.
{
    var ret = document.pwl.pwlMdlDependenciesGet(
                    document.get_deps.ModelNameExt.value);
    if (!ret.Status)
    {
        alert("pwlMdlDependenciesGet failed (" + ret.ErrorCode + ")");
        return ;
    }
    document.get_deps.ModelNameExts.value = "";
    for (var i = 0; i < ret.NumMdls; i++)
    {
        document.get_deps.ModelNameExts.value += ret.MdlNameExt.Item(i) + "\n";
    }
}

function WlModelGetInfo()
//      Get some information about a model from Pro/INTRALINK.
{
    var info_ret = document.pwl.pwlMdlInfoGet(
                        document.get_info.ModelNameExt.value);
    if (!info_ret.Status)
    {
        alert("pwlMdlInfoGet failed (" + info_ret.ErrorCode + ")");
        return ;
    }
    document.get_info.Generic.value = info_ret.GenericName;

    // Use the list models select list to convert the type to a string.
    var ModelType = "Undefined";
    for (var i = 0; i < document.list_mdls.ModelType.length; i++)
    {
```

```
        if (parseInt(document.pwlc.eval(
                document.list_mdls.ModelType.options[i].value)) ==
            info_ret.MdlType)
        {
            ModelType = document.list_mdls.ModelType.options[i].text;
        }
    }
    document.get_info.ModelType.value = ModelType;

    document.get_info.Version.value = "";
    document.get_info.Revision.value = "";
    document.get_info.Branch.value = "";
    document.get_info.ReleaseLevel.value = "";
    var ret = document.pwl.pwlMdlIntralinkInfoGet(
                document.get_info.ModelNameExt.value);
    if (!ret.Status)
    {
        if (ret.ErrorCode != parseInt(document.pwlc.PWL_NOT_IN_SESSION))
        {
            alert("pwlMdlIntralinkInfoGet failed (" + ret.ErrorCode + ")");
        }
        return ;
    }
    document.get_info.Version.value = ret.Version;
    document.get_info.Revision.value = ret.Revision;
    document.get_info.Branch.value = ret.Branch;
    document.get_info.ReleaseLevel.value = ret.ReleaseLevel;
}

function WlModelGetDirectory ()
{
    var ret = document.pwl.pwlDirectoryCurrentGet ();

    if (!ret.Status)
        {
            alert("pwlDirectoryCurrentGet failed (" + ret.ErrorCode + ")");
        return ;
        }

    document.get_dir.CurrDirectory.value = ret.DirectoryPath;
}

function WlModelSetDirectory ()
{
    var ret = document.pwl.pwlDirectoryCurrentSet (document.get_dir.CurrDirectory.value);

    if (!ret.Status)
        {
            alert("pwlDirectoryCurrentSet failed (" + ret.ErrorCode + ")");
        }
        return ;
}

function WlModelRunMacro ()
{
    var session = (pfcCreate ("MpfcCOMGlobal")).GetProESession();

    session.RunMacro (document.macro_form.MacroVal.value);
}

</script>
</head>

<body>

<form name="open">
  <h4>Open a Model Without a Window</h4>
  <div align="center"><center><p><!-- Input arguments --> Path:
 <input type="text" name="ModelPath" size="20"> <spacer size="20">
 Model: <input type="text" name="ModelName" size="20"> <spacer size="20">
 <!-- Buttons --> <input type="button"
  value="Open Model" onclick="WlModelOpenWindowless()"></p>
  </center></div><hr>
</form>
```

```
<form name="erase">
  <h4>Erase Model</h4>
  <div align="center"><center><p><!-- Input arguments --> Model: <input
type="text" name="ModelNameExt" size="20"> <!-- Buttons --> <input
  type="button" value="Erase Model" onclick="WlModelErase()"></p>
  </center></div><hr>
</form>

<form name="rename">
  <h4>Rename Model</h4>
  <div align="center"><center><p><!-- Input arguments --> Model: <input
type="text" name="ModelNameExt" size="20"> <spacer size="20">
  New Name: <input type="text" name="NewNameExt" size="20"></p>
  </center></div><div align="center"><center><p><!-- Buttons --> <input
type="button" value="Rename Model"
  onclick="WlModelRename()"></p>
  </center></div><hr>
</form>

<form name="list_mdls">
  <h4>List Models</h4>
  <div align="center"><center><p><!-- Input arguments --> Model
Type:<select name="ModelType" size="1">
    <option value="1">Assembly</option>
    <option value="2">Part</option>
    <option value="PWL_DRAWING">Drawing</option>
    <option value="PWL_2DSECTION">2D Section</option>
    <option value="PWL_LAYOUT">Layout</option>
    <option value="PWL_DWGFORM">Drawing Form</option>
    <option value="PWL_MFG">Manufaturing</option>
    <option value="PWL_REPORT">Report</option>
    <option value="PWL_MARKUP">Markup</option>
    <option value="PWL_DIAGRAM">Diagram</option>
  </select></p>
  </center></div><div align="center"><center><p><!-- Buttons --> <input type="button" value="List Models"
  onclick="WlModelsList()"></p>
  </center></div><div align="center"><center><p><!-- Output arguments --> Models:<br>
  <textarea name="ModelNameExts" rows="4" cols="50"></textarea></p>
  </center></div><hr>
</form>
<form name="get_cur">
  <h4>Get Current Model</h4>
  <div align="center"><center><p><!-- Buttons --> <input type="button" value="Get Current Model"
  onclick="WlModelGetCurrent()"></p>
  </center></div><div align="center"><center><p><!-- Output arguments --> Model: <input type="text"

  name="ModelNameExt" size="20"></p>
  </center></div><hr>
</form>
<form name="get_deps">
  <h4>Get Model Dependencies</h4>
  <div align="center"><center><p><!-- Input arguments --> Model: <input type="text" name="ModelNameExt"
size="20"></p>
  </center></div><div align="center"><center><p><!-- Buttons --> <input type="button"
  value="Get Dependencies" onclick="WlModelGetDependencies()"></p>
  </center></div><div align="center"><center><p><!-- Output arguments --> Models:<br>
  <textarea name="ModelNameExts" rows="4" cols="50"></textarea></p>
  </center></div><hr>
</form>
<form name="get_info">
  <h4>Get Information on a Model</h4>
  <div align="center"><center><p><!-- Input arguments --> Model: <input type="text" name="ModelNameExt"
size="20"></p>
  </center></div><div align="center"><center><p><!-- Buttons --> <input type="button" value="Get Info"
  onclick="WlModelGetInfo()"></p>
  </center></div><div align="center"><center><p><!-- Output arguments --> </p>
  </center></div><div align="center"><center><table>
    <tr>
      <td>Generic:</td>
      <td><input type="text" name="Generic" size="20"></td>
      <td>Model Type:</td>
      <td><input type="text" name="ModelType" size="20"></td>
    </tr>
    <tr>
```

```
      <td>Version:</td>
      <td><input type="text" name="Version" size="20"></td>
      <td>Revision:</td>
      <td><input type="text" name="Revision" size="20"></td>
    </tr>
    <tr>
      <td>Branch:</td>
      <td><input type="text" name="Branch" size="20"></td>
      <td>Release Level:</td>
      <td><input type="text" name="ReleaseLevel" size="20"></td>
    </tr>
  </table>
  </center></div><hr>
  <h4>Get Directory</h4>
</form>

<form name="get_dir">
  <div align="center"><center><p><!-- Input arguments --> Directory: <input type="text" name="CurrDirectory"
  size="20"></p>
  </center></div><div align="center"><center><p><!-- Buttons --> <input type="button" value="Get Directory"
  onclick="WlModelGetDirectory()" name="GetDirectory"> <input type="button"
  value="Set Directory" onclick="WlModelSetDirectory()" name="SetDirectory"></p>
  </center></div>
</form>

<h4>Run Macro</h4>
<form name="macro_form">
  <div align="center"><center><p><!-- Input arguments --> Macro: <input type="text" name="MacroVal"
size="20"></p>
  </center></div><div align="center"><center><p><!-- Buttons --> <input type="button" value="Run Macro"
  onclick="WlModelRunMacro()" name="RunMacro"> </p>
  </center></div>
</form>
</body>
</html>
```

The following figures show the results of this example, as seen in the browser. Note that the first figure does not include the standard header.

List Models

Model Type: Assembly ▼

[List Models]

Models:

```
[                                    ▲]
[                                     ]
[                                     ]
[                                    ▼]
```

Get Current Model

[Get Current Model]

Model: [   ]

Get Model Dependencies

Model: [   ]

[Get Dependencies]

Models:

```
[                                    ▲]
[                                     ]
[                                     ]
[                                    ▼]
```

Get Information on a Model

Model: [ ]

[Get Info]

Generic: [ ]   Model Type: [ ]
Version: [ ]   Revision: [ ]
Branch: [ ]   Release Level: [ ]

Get Directory

Directory: [ ]

[Get Directory]   [Set Direcory]

Run Macro

Macro: [ ]

[Run Macro]

Model Items

Functions Introduced:

- **pfcScript.pwlItemNameToID()**
- **pfcScript.pwlItemIDToName()**
- **pfcScript.pwlItemNameSetByID()**

The function `pfcScript.pwlItemNameToID()` returns the identifier of the specified model item, given its name. The syntax is as follows:

```
pwlItemNameToID (
    string  MdlNameExt,  // The full name of the model
    string  ItemName,    // The name of the model item
    integer ItemType     // The type of model item
);
Additional return field:
    integer ItemID;      // The identifier of the
                         // model item
```

Similarly, to get the name of a model item given its identifier, use the function `pfcScript.pwlItemIDToName()`. The syntax is as follows:

```
pwlItemIDToName (
    string  MdlNameExt,  // The full name of the model
    integer ItemID,      // The item identifier
    integer ItemType     // The type of model item
);
Additional return field:
    string  ItemName;    // The name of the model item
```

You can change the name of an item using the function `pfcScript.Script.pwlItemNameSetByID` function. The syntax is as follows:

```
pwlItemNameSetByID (
    string  MdlNameExt, // The full name of the model
    integer ItemID,     // The identifier of the
                        // model item
    integer ItemType,   // The type of model item
    string  ItemName    // The new name for the model
                        // item
);
```

The value of the *ItemType* argument should be `PWL_FEATURE`.

Windows and Views

This section describes the Web.Link functions that enable you to access and manipulate windows and views.

Windows

Functions Introduced:

- **pfcScript.pwlWindowRepaint()**
- **pfcScript.pwlSessionWindowsGet()**
- **pfcScript.pwlWindowMdlGet()**
- **pfcScript.pwlWindowActiveGet()**
- **pfcScript.pwlWindowActivate()**
- **pfcScript.pwlWindowClose()**

The function `pfcScript.pwlWindowRepaint()` repaints the window and removes highlights. Use the value −1 for the *WindowID* to repaint the current window. The syntax is as follows:

```
pwlWindowRepaint (
    integer WindowID  // The window identifier. Use -1
                      // to repaint the current window.
                      // Use parseInt with this argument.
);
```

The function `pfcScript.pwlSessionWindowsGet()` provides a count and the list of window identifiers for the current Creo Parametric session. The syntax is as follows:

```
pwlSessionWindowsGet();
Additional return fields:
    integer NumWindows;  // The number of windows
    integer WindowIDs[]; // The list of window identifiers
```

The function `pfcScript.pwlWindowMdlGet()` retrieves the model associated with the specified window. Use the value −1 for the current window. The syntax is as follows:

```
pwlWindowMdlGet (
    integer WindowID       // The window identifier.
                           // Use -1 for the current
                           // window. Use parseInt with
                           // this argument.
);
Additional return field:
    string  MdlNameExt;    // The full name of the
                           // model associated with
                           // the specified window.
```

The function `pfcScript.pwlWindowActiveGet()` provides the identifier of the currently active window. The syntax is as follows:

```
pwlWindowActiveGet();
Additional return field:
    integer WindowID; // The identifier of the currently
                      // active window. Use parseInt
                      // with this argument.
```

The function `pfcScript.pwlWindowActivate()` makes the specified window active. This is equivalent to selecting **Window**, **Activate** from the Creo Parametric menu bar. The syntax is as follows:

```
pwlWindowActivate (
    integer WindowID // The identifier of the window to
```

```
                          // make active. Use parseInt with
                          // this argument.
   );
```

To close a window, call `pfcScript.Script.pwlWindowClose`. Use the value –1 to close the current window. The syntax is as follows:

```
pwlWindowClose (
   integer WindowID // The identifier of the window
                    // to close. Use parseInt with this
                    // argument.
   );
```

Note

If you are in the middle of an operation, such as creating a feature, Creo Parametric might display a dialog box asking you to confirm the cancellation of that operation.

Use any of the following functions to get the window identifier:

- `pfcScript.pwlGeomSimprepOpen()`

- `pfcScript.pwlGraphicsSimprepOpen()`

- `pfcScript.pwlInstanceOpen()`

- `pfcScript.pwlMdlOpen()`

- `pfcScript.Script.pwlSessionWindowsGet`

- `pfcScript.pwlSimprepOpen()`

- `pfcScript.Script.pwlWindowActiveGet`

Views

Functions Introduced:

- **pfcScript.pwlViewSet()**
- **pfcScript.pwlViewDefaultSet()**
- **pfcScript.pwlMdlViewsGet()**

The function `pfcScript.pwlViewSet()` sets the view for the specified model. The syntax is as follows:

```
pwlViewSet (
   string  MdlNameExt,  // The full name of the model
   string  NamedView    // The name of the view
   );
```

The `pfcScript.pwlViewDefaultSet()` function sets the specified model to the default view for that model. The syntax is as follows:

```
pwlViewDefaultSet (
   string  MdlNameExt    // The full name of the model
   );
```

The function `pfcScript.Script.pwlMdlViewsGet` provides the number and names of all the views in the specified model. The syntax is as follows:

```
pwlMdlViewsGet (
   string  MdlNameExt    // The full name of the model
   );
Additional return fields:
   integer NumViews;     // The number of views
   string  ViewNames[];  // The list of view names
```

Selection

This section describes the Web.Link functions that enable you to highlight and select objects.

Selection Functions

Functions Introduced:

- **pfcScript.pwlSelect()**
- **pfcScript.pwlSelectionCreate()**
- **pfcScript.pwlSelectionParse()**

The function `pfcScript.pwlSelect()` enables the user to perform interactive selection on a Creo Parametric object. The syntax is as follows:

```
pwlSelect (
    string  SelectableFilter,  // The selection filter.
    integer MaxSelectable      // The maximum number
                               // of items that can be
                               // selected. If this is
                               // a negative number,
                               // there is an unlimited
                               // number of selections.
                               // Use parseInt with
                               // this argument.
);
Additional return fields:
    integer NumSelections;     // The number of
                               // selections made.
    string  Selections[];      // The selections.
```

The valid selection filter is one or more of the following in a comma-separated list (with no spaces):

- "feature"
- "dimension"
- "part"
- "prt_or_asm"

Any other filter option causes a `PWL_GENERAL_ERROR`.

The function `pfcScript.pwlSelectionCreate()` creates a selection string. The function syntax is as follows:

```
pwlSelectionCreate (
    string  TopModel,          // The top-level model.
    integer NumComponents,     // The number of
                               // components in the
                               // component path. Use
                               // parseInt with this
                               // argument.
    string  ComponentPath[],   // The model names for
                               // each level of the
                               // component path.
    integer ComponentIDs[],    // The model identifiers
                               // for each level of the
                               // component path. Use
                               // parseInt with this
                               // argument.
    integer ItemType,          // The type of selection
                               // item. Use parseInt
                               // with this argument.
    integer ItemID             // The identifier of
                               // the selection item.
                               // Use parseInt with this
                               // argument.
);
Additional return field:
    string  Selection;         // The selection string.
```

The "component path" is the path down from the root assembly to the model that owns the database item being referenced.

The possible values for *ItemType* are as follows:

- `PWL_DIMENSION`
- `PWL_FEATURE`
- `PWL_TYPE_UNUSED`

Note that you must always pass the *TopModel*. If the selection does not involve an assembly, *NumComponents* should be 0, and *ComponentPath* and *ComponentIDs* can be null. The *ComponentIDs* argument can also be null if the *ComponentPath* is enough to describe the selection. If *ItemType* and *ItemID* are `PWL_TYPE_UNUSED`, the

selection will be the model itself.

The function `pfcScript.Script.pwlSelectionParse` separates the specified selection string. The syntax is as follows:

```
pwlSelectionParse (
    string  SelString          // The selection string
                               // to parse
);
Addition return fields:
    string  TopModel;          // The top-level model
    integer NumComponents;     // The number of
                               // components in the
                               // component path
    string  ComponentPath[];   // The model names for
                               // each level of the
                               // component path
    integer ComponentIDs[];    // The model identifiers
                               // for each level of the
                               // component path
    integer ItemType;          // The type of selection
                               // item
    integer ItemID;            // The identifier of the
                               // selection item
```

Highlighting

Functions Introduced:

- **pfcScript.pwlItemHighlight()**
- **pfcScript.pwlItemUnhighlight()**

The function `pfcScript.pwlItemHighlight()` highlights the specified item, whereas `pfcScript.Script.pwlItemUnhighlight` removes the highlighting. Each function requires the full path to the item, and returns no additional fields. The syntax of the two functions is as follows:

```
pwlItemHighlight (
    string  SelString    // The selection string that
                         // identifies the item
);

pwlItemUnhighlight (
    string  SelString    // The selection string that
                         // identifies the item
);
```

Parts Materials

This chapter describes the Web.Link functions that enable you to access and manipulate part materials.

Setting Materials

Functions Introduced:

- **pfcScript.pwlPartMaterialCurrentGet()**
- **pfcScript.pwlPartMaterialCurrentSet()**
- **pfcScript.pwlPartMaterialSet()**
- **pfcScript.pwlPartMaterialsGet()**
- **pfcScript.pwlPartMaterialDataGet()**
- **pfcScript.pwlPartMaterialDataSet()**

The material properties functions are used to manipulate material properties and material property data for a Creo Parametric part.

The function `pfcScript.pwlPartMaterialCurrentGet()` gets the name of the current material used by the specified model. (The function `pfcScript.pwlPartMaterialGet()` is identical to `pfcScript.Script.pwlPartMaterialCurrentGet`, and is maintained for backward compatibility.) The syntax of the function is as follows:

```
pwlPartMaterialCurrentGet (
    string  MdlNameExt     // The full name of the model
);
Additional return field:
    string  MaterialName; // The name of the current
                          // material
```

To set the material for a part from a file, call the function pfcScript.pwlPartMaterialSet(). The material must be defined, or the function will fail. The syntax is as follows:

```
pwlPartMaterialSet (
    string  MdlNameExt,   // The full name of the model
    string  MaterialName  // The name of the material
                          // file
);
```

To set the material for a part, call the function pfcScript.pwlPartMaterialCurrentSet(). Note that the material must already be associated with the part, or the function will fail. The syntax is as follows:

```
pwlPartMaterialCurrentSet (
    string  MdlNameExt,   // The full name of the model
    string  MaterialName  // The name of the material
);
```

The function pfcScript.pwlPartMaterialsGet() provides the number and the list of all the materials used in the specified part. The syntax is as follows:

```
pwlPartMaterialsGet (
    string  MdlNameExt       // The full name of the
                             // model
);
Additional return fields:
    integer NumMaterials;   // The number of materials
                            // used in the part
    string  Materials[];    // The list of materials
```

The pfcScript.pwlPartMaterialDataGet() function gets the material data for the specified part and material. The syntax is as follows:

```
pwlPartMaterialdataGet (
    string  MdlNameExt,        // The full name of the
                               // model
    string  MaterialName       // The name of the
                               // material
);
Additional return fields:
    number  YoungModulus;      // The young modulus
    number  PoissonRatio;      // The Poisson ratio
    number  ShearModulus;      // The shear modulus
    number  MassDensity;       // The mass density
    number  ThermExpCoef;      // The thermal expansion
                               // coefficient
    number  ThermExpRefTemp;   // The thermal expansion
                               // reference temperature
    number  StructDampCoef;    // The structural
                               // damping coefficient
    number  StressLimTension;  // The stress limit
                               // for tension
    number  StressLimCompress; // The stress limit for
                               // compression
    number  StressLimShear;    // The stress limit for
                               // shear
    number  ThermConductivity; // The thermal
                               // conductivity
    number  Emissivity;        // The emissivity
    number  SpecificHeat;      // The specific heat
    number  Hardness;          // The hardness
    string  Condition;         // The condition
    number  InitBendYFactor;   // The initial bend
                               // Y factor
    string  BendTable;         // The bend table
```

To set the values of the material data elements, call the function pfcScript.Script.pwlPartMaterialDataSet. The syntax is as follows:

```
pwlPartMaterialdataSet (
```

```
    string   MdlNameExt,          // The full name of the
                                  // model
    string   MaterialName,        // The name of the
                                  // material
    number   YoungModulus,        // The young modulus
    number   PoissonRatio,        // The Poisson ratio
    number   ShearModulus,        // The shear modulus
    number   MassDensity,         // The mass density
    number   ThermExpCoef,        // The thermal expansion
                                  // coefficient
    number   ThermExpRefTemp,     // The thermal expansion
                                  // reference temperature
    number   StructDampCoef,      // The structural
                                  // damping coefficient
    number   StressLimTension,    // The stress limit
                                  // for tension
    number   StressLimCompress,   // The stress limit for
                                  // compression
    number   StressLimShear,      // The stress limit for
                                  // shear
    number   ThermConductivity,   // The thermal
                                  // conductivity
    number   Emissivity,          // The emissivity
    number   SpecificHeat,        // The specific heat
    number   Hardness,            // The hardness
    string   Condition,           // The condition
    number   InitBendYFactor,     // The initial bend
                                  // Y factor
    string   BendTable            // The bend table
);
```

Assemblies

This section describes the Web.Link functions that enable you to access assemblies and their components.

Assembly Components

Functions Introduced:

- **pfcScript.pwlAssemblyComponentsGet()**
- **pfcScript.pwlAssemblyComponentReplace()**

The function `pfcScript.pwlAssemblyComponentsGet()` provides a list of all the components in the specified assembly. This is a subset of the dependency list. (To get the entire list of dependencies, use the function `pfcScript.pwlMdlDependenciesGet()`.) The syntax is as follows:

```
pwlAssemblyComponentsGet (
    string AsmNameExt      // The full name of the
                           // assembly
);
Additional return fields:
    integer NumMdls;        // The number of components
    string  MdlNameExt[];   // The full names of the
                            // assembly components
    integer ComponentID[];  // The array of component
                            // identifiers
```

The function `pfcScript.Script.pwlAssemblyComponentReplace` enables you to replace one component for another. The syntax is as follows:

```
pwlAssemblyComponentReplace (
    string  AsmNameExt,          // The full name of
                                 // the assembly.
    string  NewComponentNameExt, // The full name of the
                                 // new component.
    integer NumComponentIDs,     // The number of
                                 // components to be
                                 // replaced. Use
                                 // parseInt with this
                                 // argument.
    integer ComponentIDs[])      // The identifiers of
                                 // the components to
```

```
                                // be replaced. Use
                                // parseInt with this
                                // argument.
```

The *ComponentIDs* argument is an array of component identifiers. If the parent has multiple occurrences of the component, this argument specifies which components to replace. The component identifiers are the feature identifiers. If this argument is an empty or null array, the function replaces all occurrences of the component.

The `pfcScript.Script.pwlAssemblyComponentReplace` function uses the following techniques, in order of precedence:

1. Automatic assembly from notebook mode
2. Family table membership
3. Interchange assembly

> **Note**
>
> If you want to use an interchange assembly, you must first load it into memory. Use the function `pfcScript.pwlMdlOpen()` and set the argument DisplayInWindow to false.

Exploded Assemblies

Functions Introduced:

- **pfcScript.pwlAssemblyExplodeStatusGet()**
- **pfcScript.pwlAssemblyExplodeStatusSet()**
- **pfcScript.pwlAssemblyExplodeDefaultSet()**
- **pfcScript.pwlAssemblyExplodeStatesGet()**
- **pfcScript.pwlAssemblyExplodeStateSet()**

These functions deal with the explode status and explode states of assemblies. The "explode status" specifies whether the given assembly is exploded, whereas the "explode state" describes what the assembly looks like when it is exploded.

The function `pfcScript.pwlAssemblyExplodeStatusGet` provides the explode status of the specified assembly. The *ExplodeStatus* is a Boolean value. If it is true, the assembly is exploded.

The syntax is as follows:

```
pwlAssemblyExplodeStatusGet (
    string  AsmNameExt      // The full name of the
                            // assembly.
);
Additional return field:
    boolean ExplodeStatus; // If this is true, the
                           // assembly is exploded.
```

Similarly, the `pfcScript.pwlAssemblyExplodeStatusSet()` function enables you to set the explode status for the specified assembly. The syntax is as follows:

```
pwlAssemblyExplodeStatusSet (
    string  AsmNameExt,     // The full name of the
                            // assembly
    boolean ExplodeStatus   // The new explode status
);
```

The function `pfcScript.pwlAssemblyExplodeDefaultSet()` sets the assembly's explode state to use the default component locations. The syntax is as follows:

```
pwlAssemblyExplodeDefaultSet (
    string AsmNameExt    // The full name of the assembly
);
```

The function `pfcScript.pwlAssemblyExplodeStatesGet()` provides the number and list of explode states for the specified assembly. The syntax is as follows:

```
pwlAssemblyExplodeStatesGet (
    string  AsmNameExt          // The full name of the
                                // assembly
);
```

```
Additional return fields:
    integer NumExpldstates;     // The number of explode
                                // states
    string  ExpldstateNames[]; // The names of the
                                // explode states
```

To set the explode state for a given assembly, call the function `pfcScript.Script.pwlAssemblyExplodeStateSet`. The syntax is as follows:

```
pwlAssemblyExplodeStateSet (
    string AsmNameExt,        // The full name of the
                              // assembly
    string ExpldstateName    // The name of a predefined
                              // explode state
);
```

Features

This section describes the Web.Link functions that enable you to access and manipulate features in Creo Parametric.

Feature Inquiry

Functions Introduced:

- **pfcScript.pwlMdlFeaturesGet()**
- **pfcScript.pwlFeatureInfoGetByID()**
- **pfcScript.pwlFeatureInfoGetByName()**
- **pfcScript.pwlFeatureParentsGet()**
- **pfcScript.pwlFeatureChildrenGet()**
- **pfcScript.pwlFeatureStatusGet()**

The `pfcScript.pwlMdlFeaturesGet()` function returns all the features that are known to the end user, including suppressed features. The syntax is as follows:

```
pwlMdlFeaturesGet (
    string  MdlNameExt,   // The full name of the model.
    integer FeatureType   // The type of feature to
                          // list. Use parseInt with
                          // this argument.
);
Additional return fields:
    integer NumFeatures;   // The number of features.
    integer FeatureIDs[]; // The list of feature
                          // identifiers.
```

Use −1 for the *FeatureType* argument to get a list of all the features. See the section <u>Web.Link Constants</u> for a complete list of the possible feature types.

The function `pfcScript.pwlFeatureInfoGetByID()` gets the information about the specified feature, given its identifier. The syntax is as follows:

```
pwlFeatureInfoGetByID (
    string  MdlNameExt,   // The full name of the model.
    integer FeatureID     // The feature identifier.
                          // Use parseInt with this
                          // argument.
);
Additional return fields:
    integer FeatureType;   // The feature type.
    integer FeatureID;     // The feature identifier.
    string  FeatureName;   // The name of the feature.
    string  FeatTypeName; // The string name of the
                          // feature type, such as
                          // "Hole."
```

The `pfcScriptScript..pwlFeatureInfoGetByName` is identical to `pfcScript.Script.pwlFeatureInfoGetByID`, except you specify the name of the feature instead of its identifier. The syntax is as follows:

```
pwlFeatureInfoGetByName (
    string MdlNameExt,     // The full name of the model
```

```
    string FeatureName    // The name of the feature
    );
    Additional return fields:
    integer FeatureType;   // The feature type
    integer FeatureID;     // The feature identifier
    string  FeatureName;   // The name of the feature
    string  FeatTypeName;  // The string name of the
                           // feature type, such as
                           // "Hole"
```

To get the parents of a feature, call the function `pfcScript.pwlFeatureParentsGet()`. The syntax is as follows:

```
    pwlFeatureParentsGet (
        string  MdlNameExt,    // The full name of the model.
        integer FeatureID      // The identifier of the
                               // child feature. Use
                               // parseInt with this
                               // argument.
    );
    Additional return fields:
    integer NumParents;    // The number of parents.
    integer ParentIDs[];   // The list of feature
                           // identifiers for the
                           // parents.
```

Similarly, the function `pfcScript.pwlFeatureChildrenGet()` provides the children of the specified feature. The syntax is as follows:

```
    pwlFeatureChildrenGet (
        string  MdlNameExt,    // The full name of the model.
        integer FeatureID      // The identifier of the
                               // parent feature. Use
                               // parseInt with this
                               // argument.
    );
    Additional return fields:
    integer NumChildren; // The number of children.
    integer ChildIDs[];  // The list of feature
                         // identifiers for the
                         // children.
```

The function `pfcScript.Script.pwlFeatureStatusGet` gets the status of the specified feature. The syntax is as follows:

```
    pwlFeatureStatusGet (
        string  MdlNameExt,  // The full name of the model.
        integer FeatureID    // The feature identifier.
                             // Use parseInt with this
                             // argument.
    );
    Additional return fields:
    integer FeatureStatus;        // The feature status.
    integer PatternStatus;        // The pattern status.
    integer GroupStatus;          // The group status.
    integer GroupPatternStatus;   // The group pattern
                                   // status.
```

The return fields are as follows:

• *FeatureStatus*—The feature status. The defined constants are as follows:

  ○ `PWL_FEAT_ACTIVE`—An ordinary feature.

  ○ `PWL_FEAT_INACTIVE`—A feature that is not suppressed, but is not currently in use for another reason. For example, a family table instance that excludes this feature.

  ○ `PWL_FEAT_FAMTAB_SUPPRESSED`—A feature suppressed by family table functionality.

  ○ `PWL_FEAT_SIMP_REP_SUPPRESSED`—A feature suppressed by simplified representation functionality.

  ○ `PWL_FEAT_PROG_SUPPRESSED`—A feature suppressed by Pro/PROGRAM™ functionality.

  ○ `PWL_FEAT_SUPPRESSED`—A suppressed feature.

  ○ `PWL_FEAT_UNREGENERATED`—A feature that is active, but not regenerated due to a regeneration failure that has not been fixed. This regeneration failure might result from an earlier feature.

- *PatternStatus*—The pattern status. The defined constants are as follows:
  - ○ `PWL_NONE`—The feature is not in a pattern.
  - ○ `PWL_LEADER`—The feature is the leader of a pattern.
  - ○ `PWL_MEMBER`—The feature is a member of the pattern.
- *GroupStatus*—The group status. The defined constants are as follows:
  - ○ `PWL_NONE`—The feature is not in a group pattern.
  - ○ `PWL_MEMBER`—The feature is in a group that is a group pattern member.
- *GroupPatternStatus*—The group pattern status. The defined constants are as follows:
  - ○ `PWL_NONE`—The feature is not in a group pattern.
  - ○ `PWL_LEADER`—The feature is the leader of the group pattern.
  - ○ `PWL_MEMBER`—The feature is a member of the group pattern.

Feature Names

Functions Introduced:

- **pfcScript.pwlFeatureNameGetByID()**
- **pfcScript.pwlFeatureNameSetByID()**

The function `pfcScript.pwlFeatureNameGetByID()` provides the name of the specified feature, given its identifier. The syntax is as follows:

```
pwlFeatureNameGetByID (
    string  MdlNameExt,    // The full name of the model.
    integer FeatureID      // The feature identifier. Use
                           // parseInt with this
                           // argument.
);
Additional return field:
    string FeatureName;    // The name of the feature.
```

To change the name of a feature, use the function `pfcScript.Script.pwlFeatureNameSetByID`. The syntax is as follows:

```
pwlFeatureNameSetByID (
    string  MdlNameExt,    // The full name of the model.
    integer FeatureID,     // The feature identifier.
                           // Use parseInt with this
                           // argument.
    string  FeatureName    // The new name of the
                           // feature.
);
```

Manipulating Features

The following sections describe the Web.Link functions that enable you to suppress, resume, and delete features.

Suppressing Features

Functions Introduced:

- **pfcScript.pwlFeatureSuppressByID()**
- **pfcScript.pwlFeatureSuppressByIDList()**
- **pfcScript.pwlFeatureSuppressByLayer()**
- **pfcScript.pwlFeatureSuppressByName()**

These functions enable you to suppress features by specifying their identifiers, identifier lists, layers, or names. The syntax for the functions is as follows:

```
pwlFeatureSuppressByID (
    string  MdlNameExt,    // The full name of the model.
    integer FeatureID      // The identifier of the
```

```
                              // feature to suppress. Use
                              // parseInt with this
                              // argument.
    );

    pwlFeatureSuppressByIDList (
        string  MdlNameExt,    // The full name of the model.
        integer NumFeatures,   // The number of feature
                               // identifiers in the list.
                               // Use parseInt with this
                               // argument.
        integer FeatureIDs[]   // The list of identifiers
                               // of features to suppress.
                               // Use parseInt with this
                               // argument.
    );

    pwlFeatureSuppressByLayer (
        string MdlNameExt,     // The full name of the model
        string LayerName       // The name of the layer
                                // to suppress
    );

    pwlFeatureSuppressByName (
        string MdlNameExt,     // The full name of the
                                // model
        string FeatureName     // The name of the feature
                                // to suppress
    );
```

Resuming Features

Functions Introduced:

- **pfcScript.pwlFeatureResumeByID()**
- **pfcScript.pwlFeatureResumeByIDList()**
- **pfcScript.pwlFeatureResumeByLayer()**
- **pfcScript.pwlFeatureResumeByName()**

These functions alow you to resume features by specifying their identifiers, identifier lists, layers, or names. These functions take the additional argument *ResumeParents*, of type Boolean. If you set this argument to true, the functions also resume the parents of the specified feature, if they are suppressed.

The syntax of the functions is as follows:

```
    pwlFeatureResumeByID (
        string  MdlNameExt,     // The full name of the
                                 // model.
        integer FeatureID,      // The identifier of the
                                 // feature to resume. Use
                                 // parseInt with this
                                 // argument.
        boolean ResumeParents   // Specifies whether to
                                 // resume the parents of
                                 // the feature.
    );

    pwlFeatureResumeByIDList (
        string  MdlNameExt,     // The full name of the
                                 // model.
        integer NumFeatures,    // The number of identifiers
                                 // in the list. Use parseInt
                                 // with this argument.
        integer FeatureIDs[],   // The list of identifiers
                                 // of features to resume.
                                 // Use parseInt with this
                                 // argument.
        boolean ResumeParents   // Specifies whether to
                                 // resume the parents of
                                 // the features.
    );

    pwlFeatureResumeByLayer (
        string  MdlNameExt,      // The full name of the
```

```
                            // model
    string  LayerName,      // The name of the layer
                            // to resume
    boolean ResumeParents   // Specifies whether to
                            // resume the parents of
                            // the features
);

pwlFeatureResumeByName (
    string  MdlNameExt,     // The full name of the
                            // model
    string  FeatureName,    // The name of the feature
                            // to resume
    boolean ResumeParents   // Specifies whether to
                            // resume the parents of
                            // the feature
);
```

Deleting Features

Functions Introduced:

- **pfcScript.pwlFeatureDeleteByID()**
- **pfcScript.pwlFeatureDeleteByIDList()**
- **pfcScript.pwlFeatureDeleteByLayer()**
- **pfcScript.pwlFeatureDeleteByName()**

These functions enable you to delete features by specifying their identifiers, identifier lists, layers, or names.

Note

If the feature has children, the children are also deleted.

The syntax of the functions is as follows:

```
pwlFeatureDeleteByID (
    string  MdlNameExt,  // The full name of the model.
    integer FeatureID    // The identifier of the
                         // feature to delete. Use
                         // parseInt with this
                         // argument.
);

pwlFeatureDeleteByIDList (
    string  MdlNameExt,   // The full name of the model.
    integer NumFeatures,  // The number of identifiers
                          // in the list. Use parseInt
                          // with this argument.
    integer FeatureIDs[]  // The list of identifiers
                          // of features to delete. Use
                          // parseInt with this
                          // argument.
);

pwlFeatureDeleteByLayer (
    string MdlNameExt,    // The full name of the model
    string LayerName      // The name of the layer to
                          // delete
);

pwlFeatureDeleteByName (
    string MdlNameExt,    // The full name of the model
    string FeatureName    // The name of the feature
                          // to delete
);
```

Displaying Parameters

Function introduced:

- **pfcScript.pwlFeatureParametersDisplay()**

The function `pfcScript.Script.pwlFeatureParametersDisplay` shows the specified parameter types for a feature

in the graphics window. The syntax is as follows:

```
pwlFeatureParametersDisplay (
    string  SelString, // The selection string that
                       // identifies the feature.
    integer ItemType   // The type of parameter to
                       // display. Use parseInt with
                       // this argument.
);
```

The possible values for *ItemType* are as follows:

- PWL_USER_PARAM

- PWL_DIM_PARAM

- PWL_PATTERN_PARAM

- PWL_REFDIM_PARAM

- PWL_ALL_PARAMS

- PWL_GTOL_PARAM

- PWL_SURFFIN_PARAM

Parameters

This section describes the Web.Link functions that enable you to access and manipulate user parameters.

Listing Parameters

Functions Introduced:

- **pfcScript.pwlMdlParametersGet()**
- **pfcScript.pwlFeatureParametersGet()**

The function pfcScript.pwlMdlParametersGet() retrieves all model parameters, given the name of the model. This does not include parameters on a feature. The syntax is as follows:

```
pwlMdlParametersGet (
    string  MdlNameExt      // The full name of the model
);
Additional return fields:
    integer NumParams;      // The number of parameters
                            // in the array ParamNames
    string  ParamNames[];   // The list of parameter
                            // names
```

The function pfcScript.Script.pwlFeatureParametersGet retrieves the parameters for the specified feature. The syntax is as follows:

```
pwlFeatureParametersGet (
    string  MdlNameExt,    // The full name of the
                           // model (part or assembly).
    integer FeatureID      // The identifier for which
                           // parameters should be
                           // found. Use parseInt with
                           // this argument.
);
Additional return fields:
    integer NumParams;     // The number of parameters
                           // in the array ParamNames.
    string  ParamNames[]; // The list of parameter
                           // names.
```

Note

This function applies only to parts and assemblies.

Identifying Parameters

Uniquely identifying a parameter requires more than the model and parameter names because the parameter name could be used by the model and several features. Therefore, two additional arguments are required for

all parameter functions—the item type and item identifier. The item type is either `PWL_FEATURE` or `PWL_MODEL`.

The item identifier does not apply to model parameters, but contains the feature identifier for feature parameters. The item identifier must be an integer value even if it is not used.

Reading and Modifying Parameters

Functions Introduced:

- **pfcScript.pwlParameterValueGet()**
- **pfcScript.pwlParameterValueSet()**
- **pfcScript.pwlParameterCreate()**
- **pfcScript.pwlParameterDelete()**
- **pfcScript.pwlParameterRename()**
- **pfcScript.pwlParameterReset()**

To retrieve the value of a parameter, use the function `pfcScript.pwlParameterValueGet()`. The syntax is as follows:

```
pwlParameterValueGet (
    string  MdlNameExt,        // The full name of the
                               // model.
    integer ItemType,          // Specifies whether it
                               // is a model (PWL_MODEL)
                               // or a feature parameter
                               // (PWL_FEATURE). Use
                               // parseInt with this
                               // argument.
    integer ItemID,            // The feature identifier.
                               // This is unused for
                               // model parameters. Use
                               // parseInt with this
                               // argument.
    string  ParamName          // The name of the
                               // parameter.
);
Additional return fields:
    integer ParamType;         // Specifies the data
```

The function returns five additional fields, but only two will be set. The field *ParamType* is always set, and its value determines what other field should be used, according to the following table.

Value of the ParamType	Additional Fields
PWL_VALUE_INTEGER	ParamIntVal
PWL_VALUE_DOUBLE	ParamDoubleVal
PWL_VALUE_STRING	ParamStringVal
PWL_VALUE_BOOLEAN	ParamBooleanVal

The following code fragment shows how to use this function.

```
<script language = "JavaScript">
function WlParameterGetValue()
{
  var mdl_ret = document.pwl.pwlMdlInfoGet (
      document.get_value.ModelNameExt.value);
  if (!mdl_ret.Status)
  {
    alert ("pwlMdlInfoGet failed (" + mdl_ret.ErrorCode + ")");
    return;
  }
  var ret = document.pwl.pwlParameterValueGet (
      document.get_value.ModelNameExt.value, parseInt (mdl_ret.MdlType),
      0 /* unused */, document.get_value.ParamName.value);
  if (!ret.Status)
  {
    alert ("pwlParameterValueGet failed (" + ret.ErrorCode + ")");
```

```
      return;
  }
  if (ret.ParamType == parseInt (document.pwl.PWL_VALUE_DOUBLE))
  {
    document.get_value.Value.value = ret.ParamDoubleVal;
  }
  else if (ret.ParamType == parseInt (document.pwl.PWL_VALUE_STRING))
  {
    document.get_value.Value.value = ret.ParamStringVal;
  }
  else if (ret.ParamType == parseInt (document.pwl.PWL_VALUE_INTEGER))
  {
    document.get_value.Value.value = ret.ParamIntVal;
  }
  else if (ret.ParamType == parseInt (document.pwl.PWL_VALUE_BOOLEAN))
  {
    document.get_value.Value.value = ret.ParamBooleanVal;
  }
}
</script>

<form name = "get_value">
<h4>Get a Value for a Parameter (Model Parameters Only)</h4>
<p>
<center>
<!-- Input arguments -->
Model: <input type = "text" name = "ModelNameExt">
Parameter: <input type = "text" name = "ParamName">
<p>
<!-- Buttons -->
<input type = "button" value = "Get Value"
     onclick = "WlParameterGetValue()">
<p>
<!-- Output arguments -->
Value: <input type = "text" name = "Value">
</center>
<hr>
</form>
```

Setting a parameter using the function `pfcScript.pwlParameterValueSet()` requires several arguments to allow for all the possibilities. The syntax is as follows:

```
   pwlParameterValueSet (
      string  MdlNameExt, // The full name of the model.
      integer ItemType,   // Specifies whether it is a
                          // model (PWL_MODEL) or a
                          // feature parameter
                          // (PWL_FEATURE). Use parseInt
                          // with this argument.
      integer ItemID,     // The feature identifier.
                          // This is unused for model
                          // parameters. Use parseInt
                          // with this argument.
      string  ParamName,  // The name of the parameter.
      integer ValueType,  // Specifies the data type of
                          // the value. Use parseInt
                          // with this argument.
      integer IntVal,     // The integer value. Use
                          // parseInt with this argument.
      number  DoubleVal,  // The number value. Use
                          // parseFloat with this
                          // argument.
      string  StringVal,  // The string value.
      Boolean BooleanVal  // The Boolean value.
   );
```

The value of *ValueType* determines which of the other four values will be used. Although only one of *IntVal*, *DoubleVal*, *StringVal*, and *BooleanVal* will be used, all must be the proper data types or an error will occur.

Creating and setting parameters are very similar. The function `pfcScript.pwlParameterCreate()` takes the same arguments and has the same return fields as `pfcScript.Script.pwlParameterValueSet`. However, creation fails if the parameter already exists, whereas setting the value succeeds only on existing parameters.

The following code fragment shows how to create a string parameter.

```
<script language = "JavaScript">
function WlParameterCreate()
{
  var mdl_ret = document.pwl.pwlMdlInfoGet (
      document.create.ModelNameExt.value);
  if (!mdl_ret.Status)
  {
    alert ("pwlMdlInfoGet failed (" + mdl_ret.ErrorCode + ")");
    return;
  }
  var ret = document.pwl.pwlParameterCreate (
      document.create.ModelNameExt.value, parseInt (mdl_ret.MdlType),
      0 /* unused */, document.create.ParamName.value,
      parseInt (document.pwl.PWL_VALUE_STRING), 0 /* unused */,
      0.0 /* unused */, document.create.Value.value, false /* unused */);
  if (!ret.Status)
  {
    alert ("pwlParameterCreate failed (" + ret.ErrorCode + ")");
    return;
  }
}
</script>

<form name = "create">
<h4>Create Parameter (Model Parameter with string Value Only)</h4>
<p>
<center>
<!-- Input arguments -->
Model: <input type = "text" name = "ModelNameExt">
Parameter: <input type = "text" name = "ParamName">
Value: <input type = "text" name = "Value">
<p>
<!-- Buttons -->
<input type = "button" value = "Create Parameter"
onclick = "WlParameterCreate()">
<p>
</center>
<hr>
</form>
```

The function `pfcScript.pwlParameterDelete()` deletes the specified parameter. The syntax is as follows:

```
pwlParameterDelete (
    string  MdlNameExt,   // The full name of the
                          // model.
    integer ItemType,     // Specifies whether it is
                          // a model (PWL_MODEL) or
                          // a feature parameter
                          // (PWL_FEATURE). Use
                          // parseInt with this
                          // argument.
    integer ItemID,       // The feature identifier.
                          // This is unused for model
                          // parameters. Use parseInt
                          // with this argument.
    string  ParamName     // The name of the parameter.
);
```

To rename a parameter, call the `pfcScript.pwlParameterRename()` function. The syntax is as follows:

```
pwlParameterRename (
    string  MdlNameExt,   // The full name of the
                          // model.
    integer ItemType,     // Specifies whether it is
                          // a model (PWL_MODEL) or
                          // a feature parameter
                          // (PWL_FEATURE). Use
                          // parseInt with this
                          // argument.
    integer ItemID,       // The feature identifier.
                          // This is unused for model
                          // parameters. Use parseInt
                          // with this argument.
    string  ParamName,    // The old name of the
                          // parameter.
```

```
    string  NewName        // The new name of the
                           // parameter.
    );
```

The function `pfcScript.Script.pwlParameterReset` restores the parameter's value to the one it had at the end of the last regeneration. The syntax is as follows:

```
pwlParameterReset (
    string  MdlNameExt,    // The full name of the
                           // model.
    integer ItemType,      // Specifies whether it is
                           // a model (PWL_MODEL) or
                           // a feature parameter
                           // (PWL_FEATURE). Use
                           // parseInt with this
                           // argument.
    integer ItemID,        // The feature identifier.
                           // This is unused for model
                           // parameters. Use parseInt
                           // with this argument.
    string  ParamName      // The name of the parameter.
    );
```

Designating Parameters

Functions Introduced:

- **pfcScript.pwlParameterDesignationAdd()**
- **pfcScript.pwlParameterDesignationRemove()**
- **pfcScript.pwlParameterDesignationVerify()**

These functions control the designation of model parameters for Pro/PDM™ and Pro/INTRALINK. A designated parameter will become visible within Pro/PDM or Pro/INTRALINK as an attribute when the model is next submitted.

The function `pfcScript.pwlParameterDesignationAdd()` designates an existing parameter. The syntax is as follows:

```
pwlParameterDesignationAdd (
    string  MdlNameExt,  // The full name of the model
    string  ParamName    // The name of the parameter
    );
```

The `pfcScript.pwlParameterDesignationRemove()` function removes the designation. The syntax is as follows:

```
pwlParameterDesignationAdd (
    string  MdlNameExt,  // The full name of the model
    string  ParamName    // The name of the parameter
    );
```

To verify whether a parameter is currently designated, call `pfcScript.Script.pwlParameterDesignationVerify`. The syntax is as follows:

```
pwlParameterDesignationVerify (
    string  MdlNameExt,  // The full name of the model.
    string  ParamName    // The name of the parameter.
    );
Additional return field:
    boolean Exists;      // If this is true, the
                         // parameter is currently
                         // designated.
```

Parameter Example

The following example shows how to use the Web.Link parameter functions.

```
<html xmlns:v="urn:schemas-microsoft-com:vml"
xmlns:o="urn:schemas-microsoft-com:office:office"
xmlns:w="urn:schemas-microsoft-com:office:word"
xmlns="http://www.w3.org/TR/REC-html40">

<head>
```

```html
<meta http-equiv=Content-Type content="text/html; charset=us-ascii">
<meta name=ProgId content=Word.Document>
<meta name=Generator content="Microsoft Word 9">
<meta name=Originator content="Microsoft Word 9">
<link rel=File-List href="./parameters_files/filelist.xml">
<link rel=Edit-Time-Data href="./parameters_files/editdata.mso">
<!--[if !mso]>
<style>
v\:* {behavior:url(#default#VML);}
o\:* {behavior:url(#default#VML);}
w\:* {behavior:url(#default#VML);}
.shape {behavior:url(#default#VML);}
</style>
<![endif]-->

<title>Web.Link Parameters Test</title>
<!--[if gte mso 9]><xml>
 <o:DocumentProperties>
  <o:Author>Scott Conover</o:Author>
  <o:LastAuthor>Scott Conover</o:LastAuthor>
  <o:Revision>5</o:Revision>
  <o:TotalTime>7</o:TotalTime>
  <o:Created>2002-11-22T15:28:00Z</o:Created>
  <o:LastSaved>2002-11-22T15:46:00Z</o:LastSaved>
  <o:Pages>1</o:Pages>
  <o:Words>151</o:Words>
  <o:Characters>863</o:Characters>
  <o:Company>PTC</o:Company>
  <o:Lines>7</o:Lines>
  <o:Paragraphs>1</o:Paragraphs>
  <o:CharactersWithSpaces>1059</o:CharactersWithSpaces>
  <o:Version>9.3821</o:Version>
 </o:DocumentProperties>
</xml><![endif]-->
<style>
<!--
 /* Style Definitions */
p.MsoNormal, li.MsoNormal, div.MsoNormal
     {mso-style-parent:"";
     margin:0in;
     margin-bottom:.0001pt;
     mso-pagination:widow-orphan;
     font-size:12.0pt;
     font-family:"Times New Roman";
     mso-fareast-font-family:"Times New Roman";}
p
     {margin-right:0in;
     mso-margin-top-alt:auto;
     mso-margin-bottom-alt:auto;
     margin-left:0in;
     mso-pagination:widow-orphan;
     font-size:12.0pt;
     font-family:"Times New Roman";
     mso-fareast-font-family:"Times New Roman";}
@page Section1
     {size:8.5in 11.0in;
     margin:1.0in 1.25in 1.0in 1.25in;
     mso-header-margin:.5in;
     mso-footer-margin:.5in;
     mso-paper-source:0;}
div.Section1
     {page:Section1;}
-->
</style>

<script  src="../jscript/pfcUtils.js">
</script>

<script  src="../jscript/wl_header.js">
</script>

<script>

function WlParametersGet()
//      Get the parameter list from the model or feature.
```

```
{
    var ret;
    var FunctionName;
    var ItemType;
     var FeatureID;

    if (document.list_parm.ModelNameExt.value == "")
    {
        return ;
    }
    ItemType = document.pwl.eval(document.list_parm.ParmType.options[
                    document.list_parm.ParmType.selectedIndex].value);
    if (parseInt(ItemType) == parseInt(document.pwlc.PWL_FEATURE))
    {
        if (document.list_parm.FeatureID.value == "")
        {
            return ;
        }

          FeatureID = parseInt(document.list_parm.FeatureID.value);
          if (isNaN (FeatureID))
          {
                alert ("Invalid feature ID!");
                return;
          }
        ret = document.pwl.pwlFeatureParametersGet(
                    document.list_parm.ModelNameExt.value,
                    parseInt(document.list_parm.FeatureID.value));
        FunctionName = "pwlFeatureParametersGet";
    }
    else
    {
          FeatureID = -1;
        ret = document.pwl.pwlMdlParametersGet(
                    document.list_parm.ModelNameExt.value);
        FunctionName = "pwlMdlParametersGet";
    }
    if (!ret.Status)
    {
        alert(FunctionName + " failed (" + ret.ErrorCode + ")");
        return ;
    }
    document.list_parm.Parameters.value = "";
    for (var i = 0; i < ret.NumParams; i++)
    {
        var val_ret = document.pwl.pwlParameterValueGet(
                            document.list_parm.ModelNameExt.value,
                            parseInt(ItemType),
                            FeatureID,
                            ret.ParamNames.Item(i));
        if (!val_ret.Status)
        {
            alert("pwlParameterValueGet failed (" + val_ret.ErrorCode + ")");
            return ;
        }
        var answer = "Undefined";
        if (val_ret.ParamType == parseInt(document.pwlc.PWL_VALUE_DOUBLE))
        {
            answer = val_ret.ParamDoubleVal;
        }
        else if (val_ret.ParamType == parseInt(document.pwlc.PWL_VALUE_STRING))
        {
            answer = val_ret.ParamStringVal;
        }
        else if (val_ret.ParamType == parseInt(document.pwlc.PWL_VALUE_INTEGER))
        {
            answer = val_ret.ParamIntVal;
        }
        else if (val_ret.ParamType == parseInt(document.pwlc.PWL_VALUE_BOOLEAN))
        {
            answer = (val_ret.ParamBooleanVal) ? "true" : "false";
        }
        document.list_parm.Parameters.value += ret.ParamNames.Item(i) + ": " +
                                                answer +"\n";
```

```javascript
        }
}

function WlParameterSetValue(FunctionName)
//      Set a parameter or create a new parameter, depending on the function
//        name.
{
    var ItemType;
    var StringValue = document.set_value.Value.value;
    var FloatValue = parseFloat(document.set_value.Value.value);
    var IntValue = parseInt(document.set_value.Value.value);
    var BoolValue = (document.set_value.Value.value.toLowerCase() == "true") ?
                      true : false;
    var ValueType = document.pwl.eval(document.set_value.ValueType.options[
                            document.set_value.ValueType.selectedIndex].value);

    // In order to create usable trail file FloatValue cannot be NaN
    if (isNaN(FloatValue))
    {
        FloatValue = 1.1;
    }
     if (isNaN (IntValue))
     {
          IntValue = -5;
     }

    ItemType = document.pwl.eval(document.set_value.ParmType.options[
                    document.set_value.ParmType.selectedIndex].value);
     if (ItemType == document.pwlc.PWL_MODEL)
          featureID = -1;
     else
          featureID = parseInt(document.set_value.FeatureID.value);

     if (FunctionName == "pwlParameterCreate")
     {

        var ret = document.pwl.pwlParameterCreate (
                  document.set_value.ModelNameExt.value,
                  ItemType,
                  featureID,
                  document.set_value.Parameter.value, ValueType,
                  IntValue, FloatValue, StringValue, BoolValue);
     }
     else
     {
         var ret = document.pwl.pwlParameterValueSet (
                         document.set_value.ModelNameExt.value,
                  ItemType,
                  featureID,
                  document.set_value.Parameter.value, ValueType,
                  IntValue, FloatValue, StringValue, BoolValue);
     }

    if (!ret.Status)
    {
        alert(FunctionName + " failed (" + ret.ErrorCode + ")");
        return ;
    }
}

function WlParameterMisc(FunctionName)
//      Run miscellaneous parameter functions that take only model name,
//      item type, item ID, and parameter name as arguments.
{
    var ItemType;

    ItemType = document.pwl.eval(document.misc_parm.ParmType.options[
                    document.misc_parm.ParmType.selectedIndex].value);
     if (ItemType == document.pwlc.PWL_MODEL)
     {
          FeatureID = -1;
     }
     else
     {
          FeatureID = parseInt(document.misc parm.FeatureID.value);
```

```
                if (isNaN (FeatureID))
                    {
                        alert ("Invalid feature id: "+FeatureID);
                        return;
                    }
        }
    if  (FunctionName == "pwlParameterReset")
        {
            var ret = document.pwl.pwlParameterReset(
                    document.misc_parm.ModelNameExt.value,
                    ItemType,
                    FeatureID,
                    document.misc_parm.Parameter.value);
        }
        else
        {
            var ret = document.pwl.pwlParameterDelete(
                    document.misc_parm.ModelNameExt.value,
                    ItemType,
                    FeatureID,
                    document.misc_parm.Parameter.value);

        }

    if (!ret.Status)
    {
        alert(FunctionName + " failed (" + ret.ErrorCode + ")");
        return ;
    }
}

function WlParameterRename()
//      Rename a parameter.
{
    var ItemType;

    ItemType = document.pwl.eval(document.misc_parm.ParmType.options[
                    document.misc_parm.ParmType.selectedIndex].value);
     if (ItemType == document.pwlc.PWL_MODEL)
     {
            FeatureID = -1;
     }
     else
     {
            FeatureID = parseInt(document.misc_parm.FeatureID.value);
            if (isNaN (FeatureID))
                    {
                        alert ("Invalid feature id: "+FeatureID);
                        return;
                    }
     }


    var ret = document.pwl.pwlParameterRename(
                    document.misc_parm.ModelNameExt.value,
                    ItemType,
                    FeatureID,
                    document.misc_parm.Parameter.value,
                    document.misc_parm.NewName.value);
    if (!ret.Status)
    {
        alert("pwlParameterRename failed (" + ret.ErrorCode + ")");
        return ;
    }
}

function WlParameterDesignate(FunctionName)
//      Run designate parameter functions that take only the model name
//      and parameter name as arguments and don't return anything.
{
 if (FunctionName == "pwlParameterDesignationAdd")
     {
    var ret = document.pwl.pwlParameterDesignationAdd(
                    document.desg_parm.ModelNameExt.value,
```

```javascript
                     document.desg_parm.Parameter.value);
        }
        else
        {
                var ret = document.pwl.pwlParameterDesignationRemove(
                        document.desg_parm.ModelNameExt.value,
                        document.desg_parm.Parameter.value);
        }

    if (!ret.Status)
    {
        alert(FunctionName + " failed (" + ret.ErrorCode + ")");
        return ;
    }
}

function WlParameterVerifyDesignation()
//     Verify that a parameter has been designated.
{
    var ret = document.pwl.pwlParameterDesignationVerify(
                    document.desg_parm.ModelNameExt.value,
                    document.desg_parm.Parameter.value);
    if (!ret.Status)
    {
        alert("pwlParameterDesignationVerify failed (" + ret.ErrorCode + ")");
        return ;
    }
    document.desg_parm.Exist.value = ret.Exists;
}

function NotApplicable(form)
//      Print N\A in the feature ID field when a model is selected.
{
    if (form.ParmType.options[form.ParmType.selectedIndex].value == "PWL_MODEL")
    {
        form.FeatureID.value = "N\\A";
    }
    else if (form.FeatureID.value == "N\\A")
    {
        form.FeatureID.value = "";
    }
}
</script>
<!--[if gte mso 9]><xml>
 <o:shapedefaults v:ext="edit" spidmax="1027"/>
</xml><![endif]--><!--[if gte mso 9]><xml>
 <o:shapelayout v:ext="edit">
  <o:idmap v:ext="edit" data="1"/>
 </o:shapelayout></xml><![endif]-->
</head>

<body lang=EN-US style='tab-interval:.5in'>
<div class=Section1>
<form name="list_parm">
<h4>List Parameters<o:p></o:p></h4>
<div align=center>
<table border=0 cellpadding=0 style='mso-cellspacing:1.5pt;mso-padding-alt:
 0in 0in 0in 0in'>
 <tr>
  <td style='padding:.75pt .75pt .75pt .75pt'>
  <p align=center style='text-align:center'><!-- Input arguments -->Model:</p>
  </td>
  <td style='padding:.75pt .75pt .75pt .75pt'>
  <p align=center style='text-align:center'>Display:</p>
  </td>
  <td style='padding:.75pt .75pt .75pt .75pt'>
  <p align=center style='text-align:center'>Feature ID:</p>
  </td>
 </tr>
 <tr>
  <td style='padding:.75pt .75pt .75pt .75pt'>
  <p class=MsoNormal align=center style='text-align:center'>
<INPUT TYPE="text" SIZE="20" NAME="ModelNameExt"><o:p></o:p></p>
  </td>
  <td style='padding:.75pt .75pt .75pt .75pt'>
```

```html
   <p class=MsoNormal align=center style='text-align:center'><SELECT NAME="ParmType"
   onchange="NotApplicable(document.list_parm)">
<OPTION SELECTED VALUE="PWL_MODEL">By Model
<OPTION VALUE="PWL_FEATURE">By Feature ID
</SELECT></p>
   </td>
   <td style='padding:.75pt .75pt .75pt .75pt'>
   <p class=MsoNormal align=center style='text-align:center'><INPUT TYPE="text" SIZE="20" NAME="FeatureID"
VALUE="N\A"></p>
   </td>
 </tr>
</table>
</div>
<!-- Buttons -->
<p align=center style='text-align:center'>
<input type=button value="Get Parameters" onclick="WlParametersGet()">

<!-- Output arguments -->Parameters:<br>
<TEXTAREA COLS="50" NAME="Parameters"></TEXTAREA></p>
<div class=MsoNormal align=center style='text-align:center'>
<hr size=2 width="auto" align=center>
</div>
</form>
<form name="set_value">
<h4>Create or Set a Parameter<o:p></o:p></h4>
<div align=center>
<table border=0 cellpadding=0 style='mso-cellspacing:1.5pt;mso-padding-alt:
 0in 0in 0in 0in'>
 <tr>
  <td style='padding:.75pt .75pt .75pt .75pt'>
  <p class=MsoNormal><!-- Input arguments -->Model:</p>
  </td>
  <td style='padding:.75pt .75pt .75pt .75pt'>
  <p class=MsoNormal><INPUT TYPE="text" SIZE="20" NAME="ModelNameExt"></p>
  </td>
  <td style='padding:.75pt .75pt .75pt .75pt'>
  <p class=MsoNormal>Parameter Type:<o:p></o:p></p>
  </td>
  <td style='padding:.75pt .75pt .75pt .75pt'>
  <p class=MsoNormal><SELECT NAME="ParmType"
  onchange="NotApplicable(document.set_value)">
<OPTION SELECTED VALUE="PWL_MODEL">By Model
<OPTION VALUE="PWL_FEATURE">By Feature ID
</SELECT></p>
  </td>
 </tr>
 <tr>
  <td style='padding:.75pt .75pt .75pt .75pt'>
  <p class=MsoNormal>Feature ID:</p>
  </td>
  <td style='padding:.75pt .75pt .75pt .75pt'>
  <p class=MsoNormal><INPUT TYPE="text" SIZE="20" NAME="FeatureID" VALUE="N\A"></p>
  </td>
  <td style='padding:.75pt .75pt .75pt .75pt'>
  <p class=MsoNormal>Parameter:</p>
  </td>
  <td style='padding:.75pt .75pt .75pt .75pt'>
  <p class=MsoNormal><INPUT TYPE="text" SIZE="20" NAME="Parameter"></p>
  </td>
 </tr>
 <tr>
  <td style='padding:.75pt .75pt .75pt .75pt'>
  <p class=MsoNormal>Value Type:</p>
  </td>
  <td style='padding:.75pt .75pt .75pt .75pt'>
  <p class=MsoNormal><SELECT NAME="ValueType">
<OPTION VALUE="PWL_VALUE_BOOLEAN">Boolean
<OPTION VALUE="PWL_VALUE_DOUBLE">Double
<OPTION VALUE="PWL_VALUE_INTEGER">Integer
<OPTION VALUE="PWL_VALUE_STRING">String
</SELECT></p>
  </td>
  <td style='padding:.75pt .75pt .75pt .75pt'>
  <p class=MsoNormal>Value:</p>
```

```
   </td>
   <td style='padding:.75pt .75pt .75pt .75pt'>
   <p class=MsoNormal><INPUT TYPE="text" SIZE="20" NAME="Value"></p>
   </td>
 </tr>
</table>
</div>


<!-- Buttons -->
<div class=MsoNormal align=center style='text-align:center'>
<hr size=2 width="auto" align=center>
<input type=button value=Create onclick="WlParameterSetValue('pwlParameterCreate')">
<input type=button value="Set Value" onclick="WlParameterSetValue('pwlParameterValueSet')">
</div>
</form>
<form name="misc_parm">
<h4>Misc Parameters<o:p></o:p></h4>
<div align=center>
<table border=0 cellpadding=0 style='mso-cellspacing:1.5pt;mso-padding-alt:
 0in 0in 0in 0in'>
 <tr>
   <td style='padding:.75pt .75pt .75pt .75pt'>
   <p align=center style='text-align:center'><!-- Input arguments -->Model:</p>
   </td>
   <td style='padding:.75pt .75pt .75pt .75pt'>
   <p align=center style='text-align:center'>Display:</p>
   </td>
   <td style='padding:.75pt .75pt .75pt .75pt'>
   <p align=center style='text-align:center'>Feature ID:</p>
   </td>
 </tr>
 <tr>
   <td style='padding:.75pt .75pt .75pt .75pt'>
   <p class=MsoNormal align=center style='text-align:center'>
<INPUT TYPE="text" SIZE="20" NAME="ModelNameExt"><o:p></o:p></p>
   </td>
   <td style='padding:.75pt .75pt .75pt .75pt'>
   <p class=MsoNormal align=center style='text-align:center'><SELECT NAME="ParmType"
   onchange="NotApplicable(document.misc_parm)">
<OPTION SELECTED VALUE="PWL_MODEL">By Model
<OPTION VALUE="PWL_FEATURE">By Feature ID
</SELECT></p>
   </td>
   <td style='padding:.75pt .75pt .75pt .75pt'>
   <p class=MsoNormal align=center style='text-align:center'>
<INPUT TYPE="text" SIZE="20" NAME="FeatureID" VALUE="N\A"></p>
   </td>
 </tr>
</table>
</div>
<p align=center style='text-align:center'>Parameter:
<INPUT TYPE="text" SIZE="20" NAME="Parameter"><o:p></o:p></p>
<p align=center style='text-align:center'><!-- Buttons -->
<input type=button value=Delete onclick="WlParameterMisc('pwlParameterDelete')">
<input type=button value=Reset onclick="WlParameterMisc('pwlParameterReset')">
<input type=button value=Rename onclick="WlParameterRename()">

<!-- Extra input arguments and a button for rename -->New Name:
<INPUT TYPE="text" SIZE="20" NAME="NewName">
<spacer size=20>
<o:p></o:p></p>
<div class=MsoNormal align=center style='text-align:center'>
<hr size=2 width="auto" align=center>
</div>
</form>
<form name="desg_parm">
<h4>Designate Model Parameters<o:p></o:p></h4>
<p align=center style='text-align:center'><!-- Input arguments -->Model:
 <INPUT TYPE="text" SIZE="20" NAME="ModelNameExt">
<spacer size=20>
Parameter: <INPUT TYPE="text" SIZE="20" NAME="Parameter"><o:p></o:p></p>
<p align=center style='text-align:center'><!-- Buttons -->
<input type=button value="Add Designation" onclick=
"WlParameterDesignate('pwlParameterDesignationAdd')">
<input type=button value="Remove Designation" onclick=
```

```
"WlParameterDesignate('pwlParameterDesignationRemove')">
<br>
<!-- Extra output arguments and a button for verifying designations -->
<input type=button value="Verify Designation" onclick="WlParameterVerifyDesignation()">
<spacer size=20>
Exists: <INPUT TYPE="text" SIZE="20" NAME="Exist"></p>
<div class=MsoNormal align=center style='text-align:center'>
<hr size=2 width="auto" align=center>
</div>
</form>
</div>
</body>
</html>
```

The following figures show the results of this example, as seen in the browser. Note that the first figure does not include the standard header. See section JavaScript Header for more information on the `wl_header.js` header.

Misc Parameters

Model:	Display:	Feature ID:
	By Model ▾	N\A

Parameter:

Delete   Reset

New Name: Rename

Designate Model Parameters

Model: Parameter:

Add Designation   Remove Designation

Verify Designation   Exists:

Dimensions

This section describes the Web.Link functions that enable you to access and manipulate dimensions in Creo Parametric.

Note

These functions are supported for parts, assemblies, and drawings, but are not supported for sections.

Reading and Modifying Dimensions

Functions Introduced:

- **pfcScript.pwlMdlDimensionsGet()**
- **pfcScript.pwlFeatureDimensionsGet()**
- **pfcScript.pwlDimensionInfoGetByID()**
- **pfcScript.pwlDimensionInfoGetByName()**
- **pfcScript.pwlDimensionValueSetByID()**

To retrieve the dimensions or reference dimensions on a model, use the function
`pfcScript.pwlMdlDimensionsGet()`. The syntax is as follows:

```
pwlMdlDimensionsGet (
   string  MdlNameExt,  // The full name of the model
                         // to which the dimensions
                         // belong.
   integer DimType       // The dimension type. Use
                         // parseInt with this argument.
);
Additional return fields:
   integer NumDims;      // The number of dimensions.
   integer DimIDs[];     // The dimension identifiers.
```

The valid values for *DimType* are `PWL_DIMENSION` and `PWL_REF_DIMENSION`, which determine whether the function should retrieve standard (geometry) or reference dimensions, respectively.

The function `pfcScript.pwlFeatureDimensionsGet()` gets the dimensions for the specified feature. Note that this function does not return any reference dimensions. (Features do not own reference dimensions—the model does.) The syntax is as follows:

```
pwlFeatureDimensionsGet (
    string  MdlNameExt,  // The full name of the model
                         // to which the dimensions
                         // belong.
    integer FeatID       // The feature to which the
                         // dimensions belong. Use
                         // parseInt with this argument.
);
Additional return fields:
    integer NumDims;     // The number of dimensions.
    integer DimIDs[];    // The dimension identifiers.
```

Web.Link provides two powerful functions to retrieve information about dimensions—`pfcScript.pwlDimensionInfoGetByID()` and `pfcScript.pwlDimensionInfoGetByName()`. The syntax for the functions is as follows:

```
pwlDimensionInfoGetByID (
    string  MdlNameExt,     // The full name of the
                            // model to which the
                            // dimension belongs.
    integer DimensionID,    // The integer identifier
                            // of the dimension. Use
                            // parseInt with this
                            // argument.
    integer DimensionType   // The dimension type
                            // (PWL_DIMENSION or
                            // PWL_REF_DIMENSION). Use
                            // parseInt with this
                            // argument.
);

pwlDimensionInfoGetByName (
    string  MdlNameExt,     // The full name of the
                            // model to which the
                            // dimension belongs.
    string  DimensionName,  // The name of the
                            // dimension.
    integer DimensionType   // The dimension type
                            // (PWL_DIMENSION or
                            // PWL_REF_DIMENSION). Use
                            // parseInt with this
                            // argument.
);
```

Both functions return the following additional fields:

```
number  DimValue;    // The value of the dimension
integer DimID;       // The dimension identifier
string  DimName;     // The name of the dimension
integer DimStyle;    // The dimension style
integer TolType,     // The tolerance type
number  TolPlus;     // The tolerance amount above
                     // the nominal value
number  TolMinus     // The tolerance amount below
                     // the nominal value
```

The possible values for the *DimStyle* field are as follows:

* `PWL_LINEAR_DIM`—Linear dimension

* `PWL_RADIAL_DIM`—Radial dimension

* `PWL_DIAMETRICAL_DIM`—Diametrical dimension

* `PWL_ANGULAR_DIM`—Angular dimension

The possible values for *TolType* are as follows:

- `PWL_TOL_DEFAULT`—Displays the nominal tolerance.

- `PWL_TOL_PLUS_MINUS`—Displays the nominal tolerance with a plus/minus.

- `PWL_TOL_LIMITS`—Displays the upper and lower tolerance limits.

- `PWL_TOL_PLUS_MINUS_SYM`—Displays the tolerance as $\pm x$, +/-x, +/-, where x is the plus tolerance. The value of the minus tolerance is irrelevant and unused.

The function `pfcScript.pwlDimensionValueSetByID` enables you to set the value of a dimension. The syntax is as follows:

```
pwlDimensionValueSetByID (
    string  MdlNameExt,   // The full name of the model
                          // to which the dimension
                          // belongs.
    integer DimensionID,  // The integer identifier of
                          // the dimension. Use parseInt
                          // with this argument.
    number Value          // The new value of the
                          // dimension. Use parseFloat
                          // with this argument.
    );
```

The function `pfcScript.pwlDimensionValueSetByID` does not require a dimension type because you cannot set reference dimensions.

Note

This function works for solids only (parts, assemblies, and derivative types).

Dimension Tolerance

Function introduced:

- **pfcScript.pwlDimensionToleranceSetByID()**

To set the dimension tolerance, call the function `pfcScript.Script.pwlDimensionToleranceSetByID`. The syntax is as follows:

```
pwlDimensionToleranceSetByID (
    string  MdlNameExt,  // The full name of the model
                         // to which the dimension
                         // belongs.
    integer DimensionID, // The integer identifier
                         // of the dimension. Use
                         // parseInt with this argument.
    number  TolPlus,     // The positive element of
                         // the tolerance. Use
                         // parseFloat with this
                         // argument.
    number  TolMinus     // The negative element of
                         // the tolerance. Use
                         // parseFloat with this
                         // argument.
    );
```

Note

This function works for solids only (parts, assemblies, and derivative types).

Dimension Example

The following example shows how to use the Web.Link dimension functions.

```
<html>
<head>
<title>Web.Link Dimensions Test</title>

<script src="../jscript/pfcUtils.js">
</script>
<script src="../jscript/wl_header.js">
document.writeln ("Error loading Web.Link header!");
```

```
</script>

<script language="JavaScript">
function WlDimensionGet()
//      Gets the dimensions, reference dimensions, or feature dimensions.
{
    var ret;
    var FunctionName;
    var DimType;

    if (document.list_dim.ModelNameExt.value == "")
    {
         return ;
    }

    if (document.list_dim.DimType.options[
            document.list_dim.DimType.selectedIndex].value == "BY_FEATURE")
    {
        if (document.list_dim.FeatureID.value == "")
        {
             return ;
        }
          FeatureID = parseInt (document.list_dim.FeatureID.value);
          if (isNaN (FeatureID))
              {
                   alert ("Feature ID invalid: "+document.list_dim.FeatureID.value);
                   return;
              }
        ret = document.pwl.pwlFeatureDimensionsGet(
                 document.list_dim.ModelNameExt.value,
                 FeatureID);
        FunctionName = "pwlFeatureDimensionsGet";
        DimType = document.pwlc.PWL_DIMENSION_STANDARD;
    }
    else
    {
          DimType = document.pwl.eval(document.list_dim.DimType.options[
                          document.list_dim.DimType.selectedIndex].value);
          if (isNaN (DimType) || DimType == -10001)
              {
                   alert ("Could not recognize dim type");
              }
/*
          if (DimType == document.pwlc.PWL_DIMENSION_STANDARD)
          {
              pfcDimType = 10;  // pfcITEM_DIMENSION
          }
          else
          {
              pfcDimType = 11;  // pfcITEM_REF_DIMENSION
          }

        var s = glob.GetProSession ();

          var m = s.GetModelFromFileName (document.list_dim.ModelNameExt.value);

          if (m == void null)
          {
              alert ("Couldn't find model: "+document.list_dim.ModelNameExt.value);
              return;
          }
          var items = m.ListItems (pfcDimType);
          if (items == void null)
          {
              alert ("Model items were null!");
              return;
          }
          if (items.Count == 0)
          {
              alert ("Empty items!");
          }
          else
              alert ("Items :" + items.Count);
     }
*/
```

```
            ret = document.pwl.pwlMdlDimensionsGet(
                    document.list_dim.ModelNameExt.value,
                    DimType);
        FunctionName = "pwlMdlDimensionsGet";
    }
    if (!ret.Status)
    {
        alert(FunctionName + " failed (" + ret.ErrorCode + "): " + ret.ErrorString);
        return ;
    }
    document.list_dim.DimValues.value = "";
      for (var i = 0; i < ret.NumDims; i++)
    {
        var info_ret = document.pwl.pwlDimensionInfoGetByID(
                            document.list_dim.ModelNameExt.value,
                            ret.DimIDs.Item(i), DimType);
        if (!info_ret.Status)
        {
            alert("pwlDimensionInfoGetByID failed (" + info_ret.ErrorCode +
                    ")");
            return ;
        }
        document.list_dim.DimValues.value += info_ret.DimName + " (#" +
            info_ret.DimID + "):  " + info_ret.DimValue +
            ((info_ret.DimStyle == parseInt(document.pwlc.PWL_ANGULAR_DIM)) ?
             " degrees" : "") + " (-" + info_ret.TolMinus + "/+" +
            info_ret.TolPlus + ")\n";
    }
}

function WlDimensionGetByName()
//      Gets a dimension by name.
{
    if (document.get_value_name.ModelNameExt.value == "" ||
        document.get_value_name.DimName.value == "")
    {
        return ;
    }
    var DimType = document.pwl.eval(document.get_value_name.DimType.options[
                        document.get_value_name.DimType.selectedIndex].value);
     if (isNaN (DimType) || DimType == -10001)
                {
                    alert ("Could not recognize dim type");
                }
    var ret = document.pwl.pwlDimensionInfoGetByName(
                    document.get_value_name.ModelNameExt.value,
                    document.get_value_name.DimName.value,
                    DimType);
    if (!ret.Status)
    {
        alert("pwlDimensionInfoGetByName failed (" + ret.ErrorCode + ")");
        return ;
    }
    document.get_value_name.DimValue.value = ret.DimName + " (#" +
        ret.DimID + "):  " + ret.DimValue +
        ((ret.DimStyle == parseInt(document.pwlc.PWL_ANGULAR_DIM)) ?
         " degrees" : "") + " (-" + ret.TolMinus + "/+" + ret.TolPlus + ")";
}

function WlDimensionSetByID()
//      Sets the value of a dimension.
{
    if (document.set_value_id.ModelNameExt.value == "" ||
        document.set_value_id.DimID.value == "" ||
        document.set_value_id.DimValue.value == "")
    {
        return ;
    }
     DimensionID = parseInt(document.set_value_id.DimID.value);
     if (isNaN (DimensionID))
     {
         alert ("Invalid dimension id: "+document.set_value_id.DimID.value);
         return;
     }
```

```javascript
        DimensionValue = parseFloat(document.set_value_id.DimValue.value);
        if (isNaN (DimensionValue))
        {
             alert ("Invalid dimension value: "+document.set_value_id.DimValue.value);
             return;
        }


     var ret = document.pwl.pwlDimensionValueSetByID(
                     document.set_value_id.ModelNameExt.value,
                     DimensionID,
                     DimensionValue);
     if (!ret.Status)
     {
         alert("pwlDimensionValueSetByID failed (" + ret.ErrorCode + ")");
         return ;
     }
}

function WlDimensionSetToleranceByID()
//      Sets the tolerance of a dimension.
{
     var ret = document.pwl.pwlDimensionToleranceSetByID(
                     document.set_tol.ModelNameExt.value,
                     parseInt(document.set_tol.DimID.value),
                     parseFloat(document.set_tol.TolPlus.value),
                     parseFloat(document.set_tol.TolMinus.value));
     if (!ret.Status)
     {
         alert("pwlDimensionValueToleranceByID failed (" + ret.ErrorCode + ")");
         return ;
     }
}

function NotApplicable(form)
//      Prints N\A in the feature ID field when a model is selected.
{
     if (form.DimType.options[form.DimType.selectedIndex].value != "BY_FEATURE")
     {
         form.FeatureID.value = "N\\A";
     }
     else if (form.FeatureID.value == "N\\A")
     {
         form.FeatureID.value = "";
     }
}

</script>
</head>
<body>
<form name="list_dim">
  <h4>List Dimensions</h4>
  <div align="center"><center><p><!-- Input arguments --> </p>
  </center></div><div align="center"><center><table>
    <tr>
      <td><div align="center"><center><p>Model:</td>
      <td align="center"><div align="center"><center><p>Display:</td>
      <td align="center"><div align="center"><center><p>Feature ID:</td>
    </tr>
    <tr align="center">
      <td><input type="text" name="ModelNameExt" size="20"></td>
      <td><select name="DimType" onchange="NotApplicable(document.list_dim)" size="1">
        <option value="PWL_DIMENSION_STANDARD" selected>Dimensions</option>
        <option value="PWL_DIMENSION_REFERENCE">Reference Dimensions</option>
        <option value="BY_FEATURE">By Feature ID</option>
      </select></td>
      <td><input type="text" name="FeatureID" value="N\A" size="20"></td>
    </tr>
  </table>
  </center></div><!-- Buttons -->
<div align="center"><center><p><input type="button" value="Get Dimensions"
  onclick="WlDimensionGet()"></p>
  </center></div><div align="center"><center><p><!-- Output arguments --> Dimensions:<br>
  <textarea name="DimValues" rows="4" cols="50"></textarea> </p>
```

```
    </center></div><hr align="center">
</form>
<form name="get_value_name">
  <h4>Get Dimension Value by Name</h4>
  <div align="center"><center><p><!-- Input arguments --> </p>
  </center></div><div align="center"><center><table>
    <tr>
      <td><div align="center"><center><p>Model:</td>
      <td align="center"><div align="center"><center><p>Dimension Name:</td>
      <td align="center"><div align="center"><center><p>Type:</td>
    </tr>
    <tr align="center">
      <td><input type="text" name="ModelNameExt" size="20"></td>
      <td><input type="text" name="DimName" size="20"></td>
      <td><select name="DimType" size="1">
        <option value="PWL_DIMENSION_STANDARD" selected>Dimensions</option>
        <option value="PWL_DIMENSION_REFERENCE">Reference Dimensions</option>
      </select></td>
    </tr>
  </table>
  </center></div><!-- Buttons -->
<div align="center"><center><p><input type="button"
  value="Get Dimension Value" onclick="WlDimensionGetByName()"></p>
  </center></div><div align="center"><center><p><!-- Output arguments -->
Value: <input type="text" name="DimValue"
  size="20"> </p>
  </center></div><hr align="center">
</form>
<form name="set_value_id">
  <h4>Set Dimension Value by ID</h4>
  <div align="center"><center><p><!-- Input arguments --> </p>
  </center></div><div align="center"><center><table>
    <tr>
      <td><div align="center"><center><p>Model:</td>
      <td align="center"><div align="center"><center><p>Dimension ID:</td>
      <td align="center"><div align="center"><center><p>Value:</td>
    </tr>
    <tr align="center">
      <td><input type="text" name="ModelNameExt" size="20"></td>
      <td><input type="text" name="DimID" size="20"></td>
      <td><input type="text" name="DimValue" size="20"></td>
    </tr>
  </table>
  </center></div><!-- Buttons -->
<div align="center"><center><p><input type="button"
  value="Set Dimension Value" onclick="WlDimensionSetByID()"> </p>
  </center></div><hr align="center">
</form>
<form name="set_tol">
  <h4>Set Dimension Tolerance by ID</h4>
  <div align="center"><center><p><!-- Input arguments --> </p>
  </center></div><div align="center"><center><table>
    <tr>
      <td>Model:</td>
      <td><input type="text" name="ModelNameExt" size="20"></td>
      <td>Dimension ID:</td>
      <td><input type="text" name="DimID" size="20"></td>
    </tr>
    <tr>
      <td>Plus Tolerance:</td>
      <td><input type="text" name="TolPlus" size="20"></td>
      <td>Minus Tolerance:</td>
      <td><input type="text" name="TolMinus" size="20"></td>
    </tr>
  </table>
  </center></div><!-- Buttons -->
<div align="center"><center><p><input type="button"
  value="Set Dimension Tolerance" onclick="WlDimensionSetToleranceByID()"> </p>
  </center></div><hr>
</form>
</body>
</html>
```

The following figures show the results of this example, as seen in the browser. Note that the first figure does

not include the standard header. Refer to the section JavaScript Header for more information on the `wl_header.js` header.

List Dimensions

Model	Display:	Feature ID:
[ ]	[Dimensions ▼]	[N\A]

[Get Dimensions]

Dimensions:

[   ]

Get Dimension Value by Name

Model	Dimension Name:	Type:
[ ]	[ ]	[Dimensions ▼]

[Get Dimension Value]

Value: [   ]

Set Dimension Value by ID

Model:	Dimension ID:	Value:
[ ]	[ ]	[ ]

[Set Dimension Value]

Set Dimension Tolerance by ID

Model:	[ ]	Dimension ID:	[ ]
Plus Tolerance:	[ ]	Minus Tolerance:	[ ]

[Set Dimension Tolerance]

Simplified Representations

This section describes the Web.Link functions that enable you to access and manipulate simplified

representations.

Retrieving Simplified Representations

Functions Introduced:

- **pfcScript.pwlSimprepOpen()**
- **pfcScript.pwlGraphicsSimprepOpen()**
- **pfcScript.pwlGeomSimprepOpen()**
- **pfcScript.pwlMdlSimprepsGet()**

The function `pfcScript.pwlSimprepOpen()` opens the specified simplified representation. The syntax is as follows:

```
pwlSimprepOpen (
    string  AsmNameExt,       // The full name of
                              // the part or assembly.
    string  Path,             // The full path to the
                              // model.
    string  RepName,          // The name of
                              // the simplified
                              // representation.
    boolean DisplayInWindow   // If this is true,
                              // display the simplified
                              // representation in a
                              // window.
);
Additional return field:
    integer WindowID;         // The identifier of
                              // the window in which
                              // the simplified
                              // representation is
                              // displayed.
```

There is a difference between opening a simplified representation and activating it. Opening a simplified representation requires Creo Parametric to read from the disk and open the appropriate representation of the specified model. Activating a simplified representation does not require Creo Parametric to read from the disk because the model containing the simplified representation is already open. This operation is analogous to the Creo Parametric operation of setting a view to be current (the model is already in session; only the model display changes).

The function `pfcScript.pwlGraphicsSimprepOpen()` retrieves the graphics of the specified assembly. The syntax is as follows:

```
pwlGraphicsSimprepOpen (
    string  AsmNameExt,       // The full name of the
                              // part or assembly
    string  Path,             // The full path to the
                              // model
    boolean DisplayInWindow   // Specifies whether to
                              // display the simplified
                              // representation in a
                              // window
);
Additional return field:
    integer WindowID;         // The identifier of the
                              // window in which
                              // the simplified
                              // representation is
                              // displayed
```

The `pfcScript.pwlGeomSimprepOpen()` function provides the geometry of the specified part or assembly. The syntax is as follows:

```
pwlGeomSimprepOpen (
    string  AsmNameExt,       // The full name of
                              // the part or assembly
    string  Path,             // The full path to
                              // the model
    boolean DisplayInWindow   // Specifies whether to
                              // display the simplified
                              // representation in a
```

```
                                    // window
    );
    Additional return field:
        integer WindowID;          // The identifier of the
                                    // window in which
                                    // the simplified
                                    // representation is
                                    // displayed
```

If you try to open a model that already exists in memory, the open functions ignore the *Path* argument.

Note

The open functions will successfully open the simplified representation that is in memory—even if the specified Path is incorrect.

Use the function `pfcScript.Script.pwlMdlSimprepsGet` to list all the simplified representations in the specified model. The syntax is as follows:

```
pwlMdlSimprepsGet (
    string   MdlNameExt       // The full name of the model
);
Additional return fields:
    integer NumSimpreps;      // The number of simplified
                              // representations
    string   Simpreps[];      // The list of simplified
                              // representations
```

Activating Simplified Representations

Functions Introduced:

- **pfcScript.pwlSimprepActivate()**
- **pfcScript.pwlSimprepMasterActivate()**
- **pfcScript.pwlGeomSimprepActivate()**
- **pfcScript.pwlGraphicsSimprepActivate()**

The function `pfcScript.pwlSimprepActivate()` activates the specified simplified representation. The syntax is as follows:

```
pwlSimprepActivate (
    string   MdlNameExt,   // The full name of the model
    string   SimprepName   // The name of the simplified
                           // representation
);
```

To activate the master simplified representation, call the function `pfcScript.pwlSimprepMasterActivate()`. The syntax is as follows:

```
pwlSimprepMasterActivate (
    string   MdlNameExt      // The full name of the model
);
```

The function `pfcScript.pwlGeomSimprepActivate()` activates the geometry simplified representation of the specified model. The syntax is as follows:

```
pwlGeomSimprepActivate (
    string   MdlNameExt      // The full name of the model
);
```

The `pfcScript.Script.pwlGraphicsSimprepActivate` function activates the graphics simplified representation of the specified model. The syntax is as follows:

```
pwlGraphicsSimprepActivate (
    string   MdlNameExt      // The full name of the model
);
```

Solids

This section describes the Web.Link functions that enable you to access and manipulate solids and their contents.

Mass Properties

Function introduced:

- **pfcScript.pwlSolidMassPropertiesGet()**

The function `pfcScript.Script.pwlSolidMassPropertiesGet` provides information about the distribution of mass in the specified part or assembly. It can provide the information relative to a coordinate system datum, which you name, or the default one if you provide an empty string or null as the name. The syntax is as follows:

```
pwlSolidMassPropertiesGet (
    string   MdlNameExt,           // The full name of
                                   // the model.
    string   CoordinateSys         // The coordinate
                                   // system used to get
                                   // the mass properties.
);
Additional return fields:
    number   Volume;               // The volume.
    number   SurfaceArea;          // The surface area.
    number   Density;              // The density. The
                                   // density value is
                                   // 1.0, unless a
                                   // material has been
                                   // assigned.
    number   Mass;                 // The mass.
    number   CenterOfGravity[3];   // The center of
                                   // gravity (COG).
    number   Inertia[9];           // The inertia matrix.
    number   InertiaTensor[9];     // The inertia tensor.
    number   CogInertiaTensor[9];  // The inertia about
                                   // the COG.
    number   PrincipalMoments[3];  // The principal
                                   // moments of inertia
                                   // (the eigenvalues
                                   // of the COG inertia).
    number   PrincipalAxes[9];     // The principal
                                   // axes (the
                                   // eigenvectors of
                                   // the COG inertia).
```

Cross Sections

Functions Introduced:

- **pfcScript.pwlSolidXSectionDisplay()**
- **pfcScript.pwlSolidXSectionsGet()**

To display a cross section, use the function `pfcScript.pwlSolidXSectionDisplay()`. The syntax is as follows:

```
pwlSolidXSectionDisplay (
    string   MdlNameExt,           // The full name of
                                   // the model
    string   CrossSectionName      // The name of the
                                   // cross section to
                                   // display
);
```

The function `pfcScript.Script.pwlSolidXSectionGet` returns all the cross sections on the specified part or assembly. The syntax is as follows:

```
pwlSolidXSectionsGet (
    string   MdlNameExt            // The full name of the
                                   // model (part or
                                   // assembly)
);
Additional return fields:
    integer  NumCrossSections;     // The number of cross
                                   // sections
    string   CrossSectionNames[];  // The names of the
                                   // cross sections
```

Family Tables

This section describes the Web.Link functions that enable you to access and manipulate family tables.

Overview

Family table functions are divided into three groups, distinguished by the prefix of the function name:

* `pwlFamtabItem`—Functions that modify table-driven items
* `pwlFamtabInstance`—Functions that modify an instance in the family table
* `pwlInstance`—Functions that perform file-management operations for family table instances

The following sections describe these functional groups in detail.

Family Table Items

Functions Introduced:

* **pfcScript.pwlFamtabItemsGet()**
* **pfcScript.pwlFamtabItemAdd()**
* **pfcScript.pwlFamtabItemRemove()**

Every item in a family table is uniquely identified by two values—its name and the family item type. The following table lists the possible values of the family item type.

Constant	Description
PWL_FAM_USER_PARAM	A user-defined parameter
PWL_FAM_DIMENSION	A dimension
PWL_FAM_IPAR_NOTE	A parameter in a pattern
PWL_FAM_FEATURE	A feature
PWL_FAM_ASMCOMP	A single instance of a component in an assembly
PWL_FAM_UDF	A user-defined feature
PWL_FAM_ASMCOMP_MODEL	All instances of a component in an assembly
PWL_FAM_GTOL	A geometric tolerance
PWL_FAM_TOL_PLUS	The plus value of a tolerance
PWL_FAM_TOL_MINUS	The minus value of a tolerance
PWL_FAM_TOL_PLUSMINUS	A tolerance
PWL_FAM_SYSTEM_PARAM	A system parameter
PWL_FAM_EXTERNAL_REFERENCE	An external reference

The function `pfcScript.Script.Script.pwlFamtabItemsGet` returns a list of all the table-driven elements currently in the family table. The syntax is as follows:

```
pwlFamtabItemsGet (
    string MdlNameExt        // The full name of the
                             // model
);
Additional return fields:
    integer    NumItems;        // The number of items
    integer    FamItemTypes[];  // The types of items
    string Items[];             // The list of table-driven
                                // elements
```

The first item in the table would have the item type *FamItemTypes[0]* and name *Items[0]*. Similar correspondence between the two arrays exist for every item in the table.

To add or remove items from a family table, use the functions `pfcScript.pwlFamtabItemAdd()` and `pfcScript.Script.pwlFamtabItemRemove`, respectively. The syntax of the functions is as follows:

```
pwlFamtabItemAdd (
    string  MdlNameExt,  // The full name of the model.
    integer FamItemType, // The type of family item.
                         // Use parseInt with this
                         // argument.
    string  Name         // The name of the item to add.
);

pwlFamtabItemRemove (
    string  MdlNameExt,  // The full name of the
                         // model.
    integer FamItemType, // The type of family item.
                         // Use parseInt with this
                         // argument.
    string  Name         // The name of the item to
                         // remove.
);
```

Adding and Deleting Family Table Instances

Functions Introduced:

- **pfcScript.pwlFamtabInstancesGet()**
- **pfcScript.pwlFamtabInstanceAdd()**
- **pfcScript.pwlFamtabInstanceRemove()**

To get a list of all the instances of a generic, call the function `pfcScript.pwlFamtabInstancesGet()`. The syntax is as follows:

```
pwlFamtabInstancesGet (
    string  MdlNameExt         // The full name of the
                                  model
);
Additional return fields:
    integer NumInstances;      // The number of instances
    string  InstanceNames[]; // The names of the
                                 // instances
```

The function `pfcScript.pwlFamtabInstanceAdd()` creates a new instance. The syntax is as follows:

```
pwlFamtabInstanceAdd (
    string MdlNameExt,    // The full name of the model
    string Name           // The name of the instance
                          // to add
);
```

Initially, a new instance will have all asterisks in the family table, making it equal to the generic.

To remove an existing instance from a family table, call the function `pfcScript.Script.pwlFamtabInstanceRemove`. The syntax is as follows:

```
pwlFamtabInstanceRemove (
    string MdlNameExt,    // The full name of the model
    string Name           // The name of the instance to
                          // remove
);
```

If the instance has been opened, the object continues to exist but will no longer be associated with the family table.

Family Table Instance Values

Functions Introduced:

- **pfcScript.pwlFamtabInstanceValueGet()**
- **pfcScript.pwlFamtabInstanceValueSet()**

The function `pfcScript.pwlFamtabInstanceValueGet()` provides the value for the specified instance and item. The syntax is as follows:

```
pwlFamtabInstanceValueGet (
    string  MdlNameExt,    // The full name of the
                           // model.
    string  Name,          // The name of the instance.
    integer FamItemType,   // The type of family item.
                           // Use parseInt with this
                           // argument.
    string  ItemName       // The name of the item.
);
Additional return fields:
    integer ValueType;     // Specifies which value
                           // argument to use.
    integer IntVal;        // The integer value.
    number  DoubleVal;     // The number value.
    string  StringVal;     // The string value.
    boolean BooleanVal;    // The Boolean value.
```

The function returns five additional fields, but only two will be set to anything. The field *ValueType* is always set and its value determines what other field should be used, according to the following table.

Value of the ValueType	Additional Field Used
PWL_VALUE_INTEGER	IntVal
PWL_VALUE_DOUBLE	DoubleVal
PWL_VALUE_STRING	StringVal
PWL_VALUE_BOOLEAN	BooleanVal

Setting the value of an instance item using the function `pfcScript.Script.Script.pwlFamtabInstanceValueSet` requires several arguments to allow for all the possibilities. The syntax is as follows:

```
pwlFamtabInstanceValueSet (
    string  MdlNameExt,    // The full name of the
                           // model.
    string  Name,          // The name of the instance.
    integer FamItemType,   // The type of the family
                           // table item. Use parseInt
                           // with this argument.
    string  ItemName,      // The name of the item.
    integer ValueType,     // Specifies which value
                           // argument to use. Use
                           // parseInt with this
                           // argument.
    integer IntVal,        // The integer value. Use
                           // parseInt with this
                           // argument.
    number  DoubleVal,     // The number value. Use
                           // parseFloat with this
                           // argument.
    string  StringVal,     // The string value.
    boolean BooleanVal     // The Boolean value.
);
```

The value of *ValueType* determines which of the other four values will be used. Although only one of *IntVal*, *DoubleVal*, *StringVal*, and *BooleanVal* will be used, all must be the proper data types or the Java function will not be found, and an error will occur.

Locking Family Table Instances

Functions Introduced:

- **pfcScript.pwlFamtabInstanceLockGet()**
- **pfcScript.pwlFamtabInstanceLockAdd()**
- **pfcScript.pwlFamtabInstanceLockRemove()**

The function `pfcScript.pwlFamtabInstanceLockGet()` determines whether the specified model is locked for modification. The function returns this information in the Boolean field *Locked*. A locked instance cannot be modified. The syntax is as follows:

```
pwlFamtabInstanceLockGet (
    string  MdlNameExt,    // The full name of the model.
    string  Name           // The name of the instance.
);
Additional return fields:
    boolean Locked;        // If this is true, the model
                           // is locked for modification.
```

To add a lock, call the function `pfcScript.pwlFamtabInstanceLockAdd()`. The syntax is as follows:

```
pwlFamtabInstanceLockAdd (
    string MdlNameExt,     // The full name of the model
    string Name            // The name of the instance
                           // to which to add the lock
);
```

Call the function `pfcScript.Script.pwlFamtabInstanceLockRemove` to remove the specified lock. The syntax is as follows:

```
pwlFamtabInstanceLockRemove (
    string MdlNameExt,     // The full name of the model
    string Name            // The name of the instance
                           // from which to remove the
                           // lock
);
```

File Management Functions for Instances

Functions Introduced:

- **pfcScript.pwlInstanceOpen()**
- **pfcScript.pwlInstanceErase()**

The function `pfcScript.pwlInstanceOpen()` enables you to open an instance in a manner similar to `pfcScript.pwlMdlOpen()`. The syntax is as follows:

```
pwlInstanceOpen (
    string  MdlNameExt,      // The full name of the
                             // model.
    string  InstanceName,    // The name of the instance
                             // to open.
    boolean DisplayInWindow  // If this is true, display
                             // the instance in a window.
);
Additional return fields:
    integer WindowID;        // The identifier of the
                             // window in which the
                             // instance is displayed.
```

The generic must already reside in memory. The Boolean argument determines whether the instance should be displayed. If the instance is to be displayed in a window, the function returns the additional field, *WindowID*.

To erase an instance from memory, use the function `pfcScript.Script.pwlInstanceErase`. The syntax is as follows:

```
pwlInstanceErase (
    string MdlNameExt,    // The full name of the model
    string Name           // The name of the instance to
                          // erase from memory
);
```

Layers

This section describes the Web.Link functions that enable you to access and manipulate layers.

Layer Functions

Functions Introduced:

- **pfcScript.pwlMdlLayersGet()**
- **pfcScript.pwlLayerCreate()**

- **pfcScript.pwlLayerDelete()**
- **pfcScript.pwlLayerDisplayGet()**
- **pfcScript.pwlLayerDisplaySet()**
- **pfcScript.pwlLayerItemsGet()**
- **pfcScript.pwlLayerItemAdd()**
- **pfcScript.pwlLayerItemRemove()**

The function `pfcScript.pwlMdlLayersGet()` provides the number and a list of all the layers in the specified model. The syntax is as follows:

```
pwlMdlLayersGet (
    string  MdlNameExt      // The full name of the model
);
Additional return fields:
    integer NumLayers;       // The number of layers in
                              // the returned array
    string  LayerNames[];  // The array of layer names
```

The function `pfcScript.pwlLayerCreate()` enables you to create a new layer. The syntax is as follows:

```
pwlLayerCreate (
    string MdlNameExt,  // The full name of the model
    string LayerName    // The name of the layer to
                        // create
);
```

To delete a layer, call the function `pfcScript.pwlLayerDelete()`.

The syntax is as follows:

```
pwlLayerDelete (
    string MdlNameExt,  // The full name of the model
    string LayerName    // The name of the layer to
                        // delete
);
```

The function `pfcScript.pwlLayerDisplayGet()` provides the display type of the specified layer. The syntax is as follows:

```
pwlLayerDisplayGet (
    string  MdlNameExt,   // The full name of the model
    string  LayerName     // The name of the layer
);
Additional return field:
    integer DisplayType;  // The display type
```

The valid values for *DisplayType* are as follows:

- `PWL_DISPLAY_TYPE_NORMAL`—A normal layer.

- `PWL_DISPLAY_TYPE_DISPLAY`—A layer selected for display.

- `PWL_DISPLAY_TYPE_BLANK`—A blanked layer.

- `PWL_DISPLAY_TYPE_HIDDEN`—A hidden layer. This applies to Assembly mode only.

To set the display type of a layer, use the function `pfcScript.pwlLayerDisplaySet()`. The syntax is as follows:

```
pwlLayerDisplaySet (
    string  MdlNameExt,  // The full name of the model.
    string  LayerName,   // The name of the layer.
    integer DisplayType  // The new display type. Use
                         // parseInt with this argument.
);
```

The function `pfcScript.pwlLayerItemsGet()` lists the items assigned to the specified layer. The syntax is as follows:

```
pwlLayerItemsGet (
    string    MdlNameExt,   // The full name of the model
    string    LayerName     // The name of the layer
);
Additional return fields:
    integer   NumItems;      // The number of items in
```

```
                         // ItemIDs
    ItemType ItemTypes[];  // The array of item types
    integer  ItemIDs[];    // The array of item
                           // identifiers
    string   ItemOwners[]; // The array of item owners
```

The possible values for *ItemType* are as follows:

- PWL_PART

- PWL_FEATURE

- PWL_DIMENSION

- PWL_REF_DIMENSION

- PWL_GTOL

- PWL_ASSEMBLY

- PWL_QUILT

- PWL_CURVE

- PWL_POINT

- PWL_NOTE

- PWL_IPAR_NOTE

- PWL_SYMBOL_INSTANCE

- PWL_DRAFT_ENTITY

- PWL_DIAGRAM_OBJECT

To add an item to a layer, call the function pfcScript.pwlLayerItemAdd(). The syntax is as follows:

```
pwlLayerItemAdd (
    string  MdlNameExt, // The full name of the model.
    string  LayerName,  // The name of the layer.
    integer ItemType,   // The type of layer item. Use
                        // parseInt with this
                        // argument.
    integer ItemID,     // The identifier of the item
                        // to add. Use parseInt with
                        // this argument.
    string  ItemOwner   // The owner of the item.
);
```

If ItemOwner is null or an empty string, the item owner is the same as the layer. Otherwise, it is a selection string to the component that owns the item.

To delete an item from a layer, call the function pfcScript.Script.pwlLayerItemRemove. The syntax is as follows:

```
pwlLayerItemRemove (
    string  MdlNameExt, // The full name of the model.
    string  LayerName,  // The name of the layer.
    integer ItemType,   // The type of layer item. Use
                        // parseInt with this
                        // argument.
    integer ItemID,     // The identifier of the item
                        // to remove. Use parseInt
                        // with this argument.
    string  ItemOwner   // The item owner. If this is
                        // null or an empty string,
                        // the item owner is the same
                        // as the layer. Otherwise,
                        // it is a selection string
                        // to the component that
                        // owns the item.
);
```

Notes

This section describes the Web.Link functions that enable you to access the notes created in Creo Parametric.

Notes Inquiry

Functions Introduced:

- **pfcScript.pwlMdlNotesGet()**
- **pfcScript.pwlFeatureNotesGet()**
- **pfcScript.pwlNoteOwnerGet()**

The function `pfcScript.pwlMdlNotesGet()` returns the number and a list of all the note identifiers in the specified model. The syntax is as follows:

```
pwlMdlNotesGet (
    string  MdlNameExt    // The full name of the model
);
Additional return fields:
    integer NumNotes;      // The number of notes in
                           // the array NoteIDs
    integer NoteIDs[];     // The array of note
                           // identifiers
```

The function `pfcScript.pwlFeatureNotesGet()` provides the number and a list of all the note identifiers for the specified feature in the model. Note that this function does not apply to drawings. The syntax is as follows:

```
pwlFeatureNotesGet (
    string  MdlNameExt,   // The full name of the model.
    integer FeatureID     // The feature whose notes
                          // should be found. Use
                          // parseInt with this
                          // argument.
);
Additional return fields:
    integer NumNotes;      // The number of notes in
                           // the array NoteIDs.
    integer NoteIDs[];     // The array of note
                           // identifiers.
```

The `pfcScript.Script.pwlNoteOwnerGet` function gets the owner of the specified note. The syntax is as follows:

```
pwlNoteOwnerGet (
    string  MdlNameExt,   // The full name of the
                          // model.
    integer NoteID        // The identifier of the
                          // note whose owner you want.
                          // Use parseInt with this
                          // argument.
);
Additional return fields:
    integer ItemType;      // The item type.
    integer NoteOwnerID;   // The identifier of the
                           // note's owner. This
                           // field is not applicable
                           // if ItemType is PWL_MODEL.
```

Note

- *You cannot modify the owner of the note.*
- *The function `pfcScript.Script.pwlNoteOwnerGet` does not apply to drawings.*

Note Names

Functions Introduced:

- **pfcScript.pwlNoteNameGet()**
- **pfcScript.pwlNoteNameSet()**

The `pfcScript.pwlNoteNameGet()` function returns the name of the specified note, given its identifier. The syntax is as follows:

```
pwlNoteNameGet (
    string  MdlNameExt,   // The full name of the model.
    integer NoteID        // The note identifier. Use
                          // parseInt with this
                          // argument.
```

```
   );
Additional return field:
   string   NoteName;     // The name of the note.
```

To set the name of a note, call the function `pfcScript.Script.pwlNoteNameSet`. The syntax is as follows:

```
pwlNoteNameSet (
   string  MdlNameExt,    // The full name of the model.
   integer NoteID,        // The note identifier. Use
                          // parseInt with this
                          // argument.
   string  NewName        // The new name of the note.
);
```

Note

These functions do not apply to drawings.

Note Text

Functions Introduced:

- **pfcScript.pwlNoteTextGet()**
- **pfcScript.pwlNoteTextSet()**

The function `pfcScript.pwlNoteTextGet()` returns the number of lines and the text strings for the specified note in the model. Symbols are replaced by an asterisk (*). The syntax is as follows:

```
pwlNoteTextGet (
   string  MdlNameExt,    // The full name of the model.
   integer NoteID         // The note identifier. Use
                          // parseInt with this
                          // argument.
);
Additional return fields:
   integer NumTextLines;  // The number of lines of
                          // text in the note.
   string  NoteText[];    // The text of the note.
```

To set the text of a note, call the function `pfcScript.Script.pwlNoteTextSet`. This function supports standard ASCII characters only. The syntax is as follows:

```
pwlNoteTextSet (
   string  MdlNameExt,    // The full name of the
                          // model.
   integer NoteID,        // The note identifier. Use
                          // parseInt with this
                          // argument.
   integer NumTextLines,  // The number of lines of
                          // text in the note. Use
                          // parseInt with this
                          // argument.
   string  NewNoteText[]  // The text of the new note.
);
```

Note

You cannot set symbols.

Note URLs

Functions Introduced:

- **pfcScript.pwlNoteURLGet()**
- **pfcScript.pwlNoteURLSet()**

The `pfcScript.pwlNoteURLGet()` provides the Uniform Resource Locator (URL) of the specified note, given its identifier. The syntax is as follows:

```
pwlNoteURLGet (
   string  MdlNameExt,  // The full name of the model.
   integer NoteID       // The note identifier. Use
                        // parseInt with this argument.
```

```
    );
    Additional return field:
        string  NoteURL;      // The URL of the note.
```

To set the URL, call the function `pfcScript.Script.pwlNoteURLSet`. The syntax is as follows:

```
    pwlNoteURLSet (
        string  MdlNameExt,   // The full name of the model.
        integer NoteID,       // The note identifier. Use
                              // parseInt with this argument.
        string  NoteURL       // The URL of the note.
    );
```

Note

These functions do not apply to drawings.

Utilities

This section describes the utility functions provided by the old Pro/Web.Link module. The utility functions enable you to manipulate directories and arrays, and to get the value of a given environment variable.

Environment Variables

Function introduced:

- **pfcScript.pwlEnvVariableGet()**

The function `pfcScript.Script.pwlEnvVariableGet` returns the value of the specified environment variable. The syntax is as follows:

```
    pwlEnvVariableGet (
        string  VarName   // The name of the environment
                          // variable whose value you want
    );
        Additional return field:
        string  Value;    // The value
```

The following code fragment shows how to get the value of the environment variable `NPX_PLUGIN_PATH`.

```
<SCRIPT language = "JavaScript">
.

function EnvVar()
{
  ret = document.pwl.pwlEnvVariableGet ("NPX_PLUGIN_PATH");

  if (ret.Status)
  {
    document.ui.RETVAL.value = "Success: Value is " + ret.Value;
  }
  else
  {
    document.ui.RETVAL.value = stat.ErrorCode+": " + ret.ErrorString;
  }
}
</script>
```

Manipulating Directories

Functions Introduced:

- **pfcScript.pwlDirectoryCurrentGet()**
- **pfcScript.pwlDirectoryCurrentSet()**
- **pfcScript.pwlDirectoryFilesGet()**

The function `pfcScript.pwlDirectoryCurrentGet()` provides the path to the current directory. The syntax is as follows:

```
    pwlDirectoryCurrentGet();
    Additional return field:
        string  DirectoryPath;    // The path to the
```

```
                              // current directory
```

The `pfcScript.pwlDirectoryCurrentSet()` function sets the current directory to the one specified by the argument *DirectoryPath*. The syntax is as follows:

```
pwlDirectoryCurrentSet (
    string  DirectoryPath    // The directory to make
                             // current
);
```

The function `pfcScript.Script.pwlDirectoryFilesGet` lists the files and subdirectories for the specified directory. Note that you can pass a filter to get only those files that have the specified extensions. The function returns the number of files found and a list of file names. The syntax is as follows:

```
pwlDirectoryFilesGet (
    string  DirectoryPath, // The directory whose
                           // files and subdirectories
                           // you want to find. If this
                           // is null, the function
                           // lists the files in the
                           // current Creo Parametric
                           // directory.
    string  Filter         // The filter string for the
                           // file extensions,
                           // separated by commas. For
                           // example. "*.prt," "*.txt".
                           // If this is null, the
                           // function lists all the
                           // files and directories.
);
Additional return fields:
    integer NumFiles;      // The number of files in
                           // FileNames.
    string  FileNames[];   // The list of file names.
    integer NumSubdirs;    // The number of
                           // subdirectories.
    string  SubdirNames;   // The list of subdirectory
                           // names.
```

Allocating Arrays

Functions Introduced:

- **pfcScript.pwluBooleanArrayAlloc()**
- **pfcScript.pwluDoubleArrayAlloc()**
- **pfcScript.pwluIntArrayAlloc()**
- **pfcScript.pwluStringArrayAlloc()**

These functions allocate arrays of Boolean values, doubles, integers, and strings, respectively. Each function takes a single argument.

Note

These functions return the allocated array.

The syntax of the functions is as follows:

```
boolean pwluBooleanArrayAlloc (
    integer ArraySize // The size of the Boolean
                      // array to allocate. Use
                      // parseInt with this argument.
);

number[] pwluDoubleArrayAlloc (
    integer ArraySize // The size of the number array
                      // to allocate. Use parseInt
                      // with this argument.
);

integer[] pwluIntArrayAlloc (
    integer ArraySize // The size of the integer array
                      // to allocate. Use parseInt
                      // with this argument.
```

```
    );

    string[] pwluStringArrayAlloc (
        integer ArraySize // The size of the string array
                          // to allocate. Use parseInt with
                          // this argument.
    );
```

Refer to section [Features](#) for a code example that uses the `pfcScript.pwluIntArrayAlloc()` function. See the section [Selection](#) for a code example that uses the `pfcScript.Script.pwluStringArrayAlloc` function.

Superseded Methods

Due to the changes in the connection and security model in the embedded browser version of Web.Link, the following methods belonging to the older version of Pro/Web.Link are obsolete:

- `pwlProEngineerStartAndConnect`

- `pwlProEngineerConnect`

- `pwlProEngineerDisconnectAndStop`

- `pwlAccessRequest`

   ### Note

   These functions are provided in the embedded browser Web.Link in order to avoid scripting errors. They are not useful in developing any applications and can be removed.

Web.Link Constants

This section lists the constants defined for Web.Link. The constants are arranged alphabetically in each category (Dimension Styles, Dimension Types, Family Table Types, and so on).

Dimension Styles

The class `pfcPWLConstants` contains the following constants:

Constant	Description
PWL_LINEAR_DIM	Linear dimension
PWL_RADIAL_DIM	Radial dimension
PWL_DIAMETRICAL_DIM	Diametrical dimension
PWL_ANGULAR_DIM	Angular dimension
PWL_UNKNOWN_STYLE_DIM	Unknown dimension

Dimension Types

The class `pfcPWLConstants` contains the following constants:

Constant	Description
PWL_DIMENSION_STANDARD	Standard dimension
PWL_DIMENSION_REFERENCE	Reference dimension

Family Table Types

The class `pfcPWLConstants` contains the following constants:

Constant	Description
PWL_FAM_TYPE_UNUSED	

Constant	Description
	Unused
PWL_FAM_USER_PARAM	User parameter
PWL_FAM_DIMENSION	Dimension
PWL_FAM_IPAR_NOTE	IPAR note
PWL_FAM_FEATURE	Feature
PWL_FAM_ASMCOMP	Assembly component
PWL_FAM_UDF	User-defined feature
PWL_FAM_ASMCOMP_MODEL	Assembly component model
PWL_FAM_GTOL	Geometric tolerance
PWL_FAM_TOL_PLUS	Displays the nominal tolerance with a plus
PWL_FAM_TOL_MINUS	Displays the nominal tolerance with a minus
PWL_FAM_TOL_PLUSMINUS	Displays the nominal tolerance with a plus/minus
PWL_FAM_SYSTEM_PARAM	System parameter
PWL_FAM_EXTERNAL_REFERENCE	External reference

Feature Group Pattern Statuses

The class pfcPWLConstants contains the following constants:

Constant	Description
PWL_GRP_PATTERN_INVALID	Invalid group pattern.
PWL_GRP_PATTERN_NONE	The feature is not in a group pattern.
PWL_GRP_PATTERN_LEADER	The feature is the leader of the group pattern.
PWL_GRP_PATTERN_MEMBER	The feature is a member of the group pattern.

Feature Group Statuses

The class pfcPWLConstants contains the following constants:

Constant	Description
PWL_GROUP_INVALID	Invalid group.
PWL_GROUP_NONE	The feature is not in a group pattern.
PWL_GROUP_MEMBER	The feature is in a group that is a group pattern member.

Feature Pattern Statuses

The class pfcPWLConstants contains the following constants:

Constant	Description
PWL_PATTERN_INVALID	Invalid pattern.
PWL_PATTERN_NONE	The feature is not in a pattern.
PWL_PATTERN_LEADER	The feature is the leader of a pattern.

PWL_PATTERN_MEMBER	The feature is a member of the pattern.

Feature Types

The class pfcPWLFeatureConstants contains the following constants:

Constant	Feature Type
PWL_FEAT_FIRST_FEAT	First feature
PWL_FEAT_HOLE	Hole
PWL_FEAT_SHAFT	Shaft
PWL_FEAT_ROUND	Round
PWL_FEAT_CHAMFER	Chamfer
PWL_FEAT_SLOT	Slot
PWL_FEAT_CUT	Cut
PWL_FEAT_PROTRUSION	Protrusion
PWL_FEAT_NECK	Neck
PWL_FEAT_FLANGE	Flange
PWL_FEAT_RIB	Rib
PWL_FEAT_EAR	Ear
PWL_FEAT_DOME	Dome
PWL_FEAT_DATUM	Datum
PWL_FEAT_LOC_PUSH	Local push
PWL_FEAT_FEAT_UDF	User-defined feature (UDF)
PWL_FEAT_DATUM_AXIS	Datum axis
PWL_FEAT_DRAFT	Draft
PWL_FEAT_SHELL	Shell
PWL_FEAT_DOME2	Second dome
PWL_FEAT_CORN_CHAMF	Corner chamfer
PWL_FEAT_DATUM_POINT	Datum point
PWL_FEAT_IMPORT	Import
PWL_FEAT_COSMETIC	Cosmetic
PWL_FEAT_ETCH	Etch
PWL_FEAT_MERGE	Merge
PWL_FEAT_MOLD	Mold
PWL_FEAT_SAW	Saw
PWL_FEAT_TURN	Turn
PWL_FEAT_MILL	Mill
PWL_FEAT_DRILL	Drill

PWL_FEAT_OFFSET	Offset
PWL_FEAT_DATUM_SURF	Datum surface
PWL_FEAT_REPLACE_SURF	Replacement surface
PWL_FEAT_GROOVE	Groove
PWL_FEAT_PIPE	Pipe
PWL_FEAT_DATUM_QUILT	Datum quilt
PWL_FEAT_ASSEM_CUT	Assembly cut
PWL_FEAT_UDF_THREAD	Thread
PWL_FEAT_CURVE	Curve
PWL_FEAT_SRF_MDL	Surface model
PWL_FEAT_WALL	Wall
PWL_FEAT_BEND	Bend
PWL_FEAT_UNBEND	Unbend
PWL_FEAT_CUT_SMT	Sheetmetal cut
PWL_FEAT_FORM	Form
PWL_FEAT_THICKEN	Thicken
PWL_FEAT_BEND_BACK	Bend back
PWL_FEAT_UDF_NOTCH	UDF notch
PWL_FEAT_UDF_PUNCH	UDF punch
PWL_FEAT_INT_UDF	For internal use
PWL_FEAT_SPLIT_SURF	Split surface
PWL_FEAT_GRAPH	Graph
PWL_FEAT_SMT_MFG_PUNCH	Sheetmetal manufacturing punch
PWL_FEAT_SMT_MFG_CUT	Sheetmetal manufacturing cut
PWL_FEAT_FLATTEN	Flatten
PWL_FEAT_SET	Set
PWL_FEAT_VDA	VDA
PWL_FEAT_SMT_MFG_FORM	Sheetmetal manufacturing for milling
PWL_FEAT_SMT_PUNCH_PNT	Sheetmetal punch point
PWL_FEAT_LIP	Lip
PWL_FEAT_MANUAL	Manual
PWL_FEAT_MFG_GATHER	Manufacturing gather
PWL_FEAT_MFG_TRIM	Manufacturing trim
PWL_FEAT_MFG_USEVOL	Manufacturing use volume
PWL_FEAT_LOCATION	Location
PWL_FEAT_CABLE_SEGM	Cable segment

PWL_FEAT_CABLE	Cable
PWL_FEAT_CSYS	Coordinate system
PWL_FEAT_CHANNEL	Channel
PWL_FEAT_WIRE_EDM	Wire EDM
PWL_FEAT_AREA_NIBBLE	Area nibble
PWL_FEAT_PATCH	Patch
PWL_FEAT_PLY	Ply
PWL_FEAT_CORE	Core
PWL_FEAT_EXTRACT	Extract
PWL_FEAT_MFG_REFINE	Manufacturing refine
PWL_FEAT_SILH_TRIM	Silhouette trim
PWL_FEAT_SPLIT	Split
PWL_FEAT_EXTEND	Extend
PWL_FEAT_SOLIDIFY	Solidify
PWL_FEAT_INTERSECT	Intersect
PWL_FEAT_ATTACH	Attach
PWL_FEAT_XSEC	Cross section
PWL_FEAT_UDF_ZONE	UDF zone
PWL_FEAT_UDF_CLAMP	UDF clamp
PWL_FEAT_DRL_GRP	Drill group
PWL_FEAT_ISEGM	Ideal segment
PWL_FEAT_CABLE_COSM	Cable cosmetic
PWL_FEAT_SPOOL	Spool
PWL_FEAT_COMPONENT	Component
PWL_FEAT_MFG_MERGE	Manufacturing merge
PWL_FEAT_FIXSETUP	Fixture setup
PWL_FEAT_SETUP	Setup
PWL_FEAT_FLAT_PAT	Flat pattern
PWL_FEAT_CONT_MAP	Contour map
PWL_FEAT_EXP_RATIO	Exponential ratio
PWL_FEAT_RIP	Rip
PWL_FEAT_OPERATION	Operation
PWL_FEAT_WORKCELL	Workcell
PWL_FEAT_CUT_MOTION	Cut motion
PWL_FEAT_BLD_PATH	Build path
PWL_FEAT_DRV_TOOL_SKETCH	Driven tool sketch

`PWL_FEAT_DRV_TOOL_EDGE`	Driven tool edge
`PWL_FEAT_DRV_TOOL_CURVE`	Driven tool curve
`PWL_FEAT_DRV_TOOL_SURF`	Driven tool surface
`PWL_FEAT_MAT_REMOVAL`	Material removal
`PWL_FEAT_TORUS`	Torus
`PWL_FEAT_PIPE_SET_START`	Piping set start point
`PWL_FEAT_PIPE_PNT_PNT`	Piping point
`PWL_FEAT_PIPE_EXT`	Pipe extension
`PWL_FEAT_PIPE_TRIM`	Pipe trim
`PWL_FEAT_PIPE_FOLL`	Follow (pipe routing)
`PWL_FEAT_PIPE_JOIN`	Pipe join
`PWL_FEAT_AUXILIARY`	Auxiliary
`PWL_FEAT_PIPE_LINE`	Pipe line
`PWL_FEAT_LINE_STOCK`	Line stock
`PWL_FEAT_SLD_PIPE`	Solid pipe
`PWL_FEAT_BULK_OBJECT`	Bulk object
`PWL_FEAT_SHRINKAGE`	Shrinkage
`PWL_FEAT_PIPE_JOINT`	Pipe joint
`PWL_FEAT_PIPE_BRANCH`	Pipe branch
`PWL_FEAT_DRV_TOOL_TWO_CNTR`	Driven tool (two centers)
`PWL_FEAT_SUBHARNESS`	Subharness
`PWL_FEAT_SMT_OPTIMIZE`	Sheetmetal optimize
`PWL_FEAT_DECLARE`	Declare
`PWL_FEAT_SMT_POPULATE`	Sheetmetal populate
`PWL_FEAT_OPER_COMP`	Operation component
`PWL_FEAT_MEASURE`	Measure
`PWL_FEAT_DRAFT_LINE`	Draft line
`PWL_FEAT_REMOVE_SURFS`	Remove surfaces
`PWL_FEAT_RIBBON_CABLE`	Ribbon cable
`PWL_FEAT_ATTACH_VOLUME`	Attach volume
`PWL_FEAT_BLD_OPERATION`	Build operation
`PWL_FEAT_UDF_WRK_REG`	UDF working region
`PWL_FEAT_SPINAL_BEND`	Spinal bend
`PWL_FEAT_TWIST`	Twist
`PWL_FEAT_FREE_FORM`	Free-form
`PWL_FEAT_ZONE`	Zone

PWL_FEAT_WELDING_ROD	Welding rod
PWL_FEAT_WELD_FILLET	Welding fillet
PWL_FEAT_WELD_GROOVE	Welding groove
PWL_FEAT_WELD_PLUG_SLOT	Welding plug slot
PWL_FEAT_WELD_SPOT	Welding spot
PWL_FEAT_SMT_SHEAR	Sheetmetal shear
PWL_FEAT_PATH_SEGM	Path segment
PWL_FEAT_RIBBON_SEGM	Ribbon segment
PWL_FEAT_RIBBON_PATH	Ribbon path
PWL_FEAT_RIBBON_EXTEND	Ribbon extend
PWL_FEAT_ASMCUT_COPY	Assembly cut copy
PWL_FEAT_DEFORM_AREA	Deform area
PWL_FEAT_RIBBON_SOLID	Ribbon solid
PWL_FEAT_FLAT_RIBBON_SEGM	Flat ribbon segment
PWL_FEAT_POSITION_FOLD	Position fold
PWL_FEAT_SPRING_BACK	Spring back
PWL_FEAT_BEAM_SECTION	Beam section
PWL_FEAT_SHRINK_DIM	Shrink dimension
PWL_FEAT_THREAD	Thread
PWL_FEAT_SMT_CONVERSION	Sheetmetal conversion
PWL_FEAT_CMM_MEASSTEP	CMM measured step
PWL_FEAT_CMM_CONSTR	CMM construct
PWL_FEAT_CMM_VERIFY	CMM verify
PWL_FEAT_CAV_SCAN_SET	CAV scan set
PWL_FEAT_CAV_FIT	CAV fit
PWL_FEAT_CAV_DEVIATION	CAV deviation
PWL_FEAT_SMT_ZONE	Sheetmetal zone
PWL_FEAT_SMT_CLAMP	Sheetmetal clamp
PWL_FEAT_PROCESS_STEP	Process step
PWL_FEAT_EDGE_BEND	Edge bend
PWL_FEAT_DRV_TOOL_PROF	Drive tool profile
PWL_FEAT_EXPLODE_LINE	Explode line
PWL_FEAT_GEOM_COPY	Geometric copy
PWL_FEAT_ANALYSIS	Analysis
PWL_FEAT_WATER_LINE	Water line
PWL_FEAT_UDF_RMDT	Rapid mold design tool

Constant	Description
PWL_FEAT_USER_FEAT	User feature

Layer Display Types

The class pfcPWLConstants contains the following constants:

Constant	Description
PWL_DISPLAY_TYPE_NONE	No layer
PWL_DISPLAY_TYPE_NORMAL	A normal layer
PWL_DISPLAY_TYPE_DISPLAY	A layer selected for display
PWL_DISPLAY_TYPE_BLANK	A blanked layer
PWL_DISPLAY_TYPE_HIDDEN	A hidden layer

Object Types

The class pfcPWLConstants contains the following constants:

Constant	Description
PWL_MODEL	Model (for parameter functions)
PWL_TYPE_UNUSED	Unused
PWL_ASSEMBLY	Assembly
PWL_PART	Part
PWL_FEATURE	Feature
PWL_DRAWING	Drawing
PWL_SURFACE	Surface
PWL_EDGE	Edge
PWL_3DSECTION	Three-dimensional section
PWL_DIMENSION	Dimension
PWL_2DSECTION	Two-dimensional section
PWL_LAYOUT	Notebook
PWL_AXIS	Axis
PWL_CSYS	Coordinate system
PWL_REF_DIMENSION	Reference dimension
PWL_GTOL	Geometric tolerance
PWL_DWGFORM	Drawing form
PWL_SUB_ASSEMBLY	Subassembly
PWL_MFG	Manufacturing object
PWL_QUILT	Quilt
PWL_CURVE	Curve
PWL_POINT	Point

PWL_NOTE	Note
PWL_IPAR_NOTE	IPAR note
PWL_EDGE_START	Start of the edge
PWL_EDGE_END	End of the edge
PWL_CRV_START	Start of the curve
PWL_CRV_END	End of the curve
PWL_SYMBOL_INSTANCE	Symbol instance
PWL_DRAFT_ENTITY	Draft entity
PWL_DRAFT_GROUP	Draft group
PWL_DRAW_TABLE	Drawing table
PWL_VIEW	View
PWL_REPORT	Report
PWL_MARKUP	Markup
PWL_LAYER	Layer
PWL_DIAGRAM	Diagram
PWL_SKETCH_ENTITY	Sketched entity
PWL_DATUM_PLANE	Datum plane
PWL_COMP_CRV	Composite curve
PWL_BND_TABLE	Bend table
PWL_PARAMETER	Parameter
PWL_DIAGRAM_OBJECT	Diagram object
PWL_DIAGRAM_WIRE	Diagram wire
PWL_SIMP_REP	Simplified representation
PWL_WELD_PARAMS	Weld parameters
PWL_EXTOBJ	External object
PWL_EXPLD_STATE	Explode state
PWL_RELSET	Set of relations
PWL_CONTOUR	Contour
PWL_GROUP	Group
PWL_UDF	User-defined feature
PWL_FAMILY_TABLE	Family table
PWL_PATREL_FIRST_DIR	Pattern direction 1
PWL_PATREL_SECOND_DIR	Pattern direction 2

Parameter Types

The class `pfcPWLConstants` contains the following constants:

Constant	Description
PWL_USER_PARAM	User parameter
PWL_DIM_PARAM	Dimension parameter
PWL_PATTERN_PARAM	Pattern parameter
PWL_DIMTOL_PARAM	Dimension tolerance parameter
PWL_REFDIM_PARAM	Reference dimension parameter
PWL_ALL_PARAMS	All parameters
PWL_GTOL_PARAM	Geometric tolerance parameter
PWL_SURFFIN_PARAM	Surface finish parameter

ParamType Field Values

The class `pfcPWLConstants` contains the following constants:

Constant	Description
PWL_VALUE_DOUBLE	Double value
PWL_VALUE_STRING	String value
PWL_VALUE_INTEGER	Integer value
PWL_VALUE_BOOLEAN	Boolean value
PWL_VALUE_NOTEID	Note identifier

ParamValue Values

The class `pfcPWLConstants` contains the following constants:

Constant	Description
PWL_PARAMVALUE_DOUBLE	Double value
PWL_PARAMVALUE_STRING	String value
PWL_PARAMVALUE_INTEGER	Integer value
PWL_PARAMVALUE_BOOLEAN	Boolean value
PWL_PARAMVALUE_NOTEID	Note identifier

Tolerance Types

The class `pfcPWLConstants` contains the following constants:

Constant	Description
PWL_TOL_DEFAULT	Displays the nominal tolerance.
PWL_TOL_PLUS_MINUS	Displays the nominal tolerance with a plus/minus.
PWL_TOL_LIMITS	Displays the upper and lower tolerance limits.
PWL_TOL_PLUS_MINUS_SYM	Displays the tolerance as +/-x, where x is the plus tolerance. The value of the minus tolerance is irrelevant and unused.

The Web.Link Online Browser

This section describes how to use the online browser provided with Web.Link.

Online Documentation Web.Link APIWizard

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

The Web.Link APIWizard contains the following items:

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

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

Installing the APIWizard

The Creo Parametric installation procedure automatically installs the Web.Link APIWizard. The files reside in a directory under the Creo Parametric load point. The location for the Web.Link APIWizard files is:

```
<creo_weblink_loadpoint>\weblinkdoc
```

Starting the APIWizard

Start the Web.Link APIWizard by pointing your browser to:

```
<creo_weblink_loadpoint>\weblinkdoc\index.html
```

Your web browser will display the Web.Link APIWizard data in a new window.

Web Browser Environments

The APIWizard supports Netscape Firefox and Internet Explorer.

For APIWizard use with Internet Explorer, the recommended browser environment requires installation of the Java plug-in.

For Netscape Firefox, the recommended browser environment requires installation of the Java Swing foundation class. If this class is not loaded on your computer, the APIWizard can load it for you. This takes several minutes, and is not persistant between sessions.

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.

APIWizard Interface

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

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 **Web.Link Modules**, **Web.Link Classes**, **Web.Link Exceptions**, **Web.Link Enumerated Types**, or the **Web.Link User's Guide**.

In **Web.Link Modules** mode, this frame displays an alphabetical list of the Web.Link modules. A module is a logical subdivision of functionality within Web.Link: for example, the `pfcFamily` module contains classes,

enumerated types, and collections related to family table operations. The frame can also display Web.Link classes, enumerated types and methods as subnodes of the modules.

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

In **Web.Link Exceptions** mode, this frame displays an alphabetical list of named exceptions in the Web.Link library.

In **Web.Link Enumerated Types** mode, this frame displays an alphabetical list of the Web.Link enumerated type classes.

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

The **Modules/Classes/Topic** frame includes a **Find** button for data searches of the *Web.Link User's Guide* or of API specifications taken from header files. See the section APIWizard Search Feature (Find) for more information on the **Find** feature.

Display Frame

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

- Web.Link module defintions
- Web.Link class defintions and their hierarchial relationships
- Web.Link method descriptions
- User's Guide content
- Code examples for Web.Link methods

The following figure displays the APIWizard interface layout.

O:\portsrc\spg\system_1\ap

Google

Convert ▾ Select

Favorites | Web.Link Creo Parametric 2.0 (JSCRIP...

Find

Web.Link User's Guide

Web.Link User's Guide
- About This Guide
- Introduction
- Setting Up Web.Link
- The Basics of Web.Link
- The Web.Link Online Brows
- Session Objects
- Selection
- Models
- Drawings
- Solid
- Windows and Views
- ModelItem
- Features
- Datum Features
- Geometry Evaluation
- Dimensions and Parameter
- Relations
- Assemblies and Componen
- Family Tables
- Interface
- Simplified Representations
- Task Based Application Libr
- Graphics
- External Data
- Windchill Connectivity APIs
- Application Conversion
- Summary of Technical Char
- Sample Applications
- Digital Rights Management
- Geometry Traversal

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 Web.Link, a tool that links the World Wide Web (WWW, or Web) to Creo Parametric, enabling you to use the Web as a tool to automate and streamline parts of your engineering process.

Note

Web.Link is supported only with Creo Parametric. It is not supported with the other Creo applications.

Audience

This manual is intended for experienced Creo Parametric users who know HyperText Markup Language (HTML) and JavaScript.

Prerequisites

This manual assumes you have the following knowledge:

- Creo Parametric
- HTML
- JavaScript

Documentation

The documentation for Web.Link includes the following:

- *Web.Link User's Guide*
- An online browser that describes the syntax of the Web.Link functions and provides a link to the online version of this manual. This includes the Embedded Browser-based *Web.Link User's Guide*.

Navigating the Modules/Classes/Topic Selection Tree

Access all Web.Link APIWizard online documentation for modules, classes, enumerated types, methods, or the *Web.Link User's Guide* from the **Modules/Classes/Topic** frame. This frame displays a tree structure of the data. Expand and collapse the tree as described below to navigate this data.

To expand the tree structure, first select **Web.Link Modules**, **Web.Link Classes**, **Web.Link Exceptions**, **Web.Link Enumerated Types**, or **Web.Link User's Guide** at the top of the Selection frame. The APIWizard displays the tree structure in a collapsed form. The switch icon to the far left of a node (i.e. a module, a class, an exception, 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 Web.Link Modules

View the Web.Link modules by choosing **Web.Link Modules** at the top of the **Modules/Classes/Topic** frame. In this mode, the APIWizard displays all the Web.Link modules in the alphabetical order.

The **Display** frame for each Web.Link module displays the information about the classes, enumerated types, and collections that belong to the module. Click the switch icon next to the desired module name, or double-click the module name text to view the clasess or enumerated types. You can also view the methods for each class in the expanded tree by clicking the switch icon next to the class name, or by double-clicking the name.

The following figure shows the collapased tree layout for the Web.Link modules.

Web.Link Modules
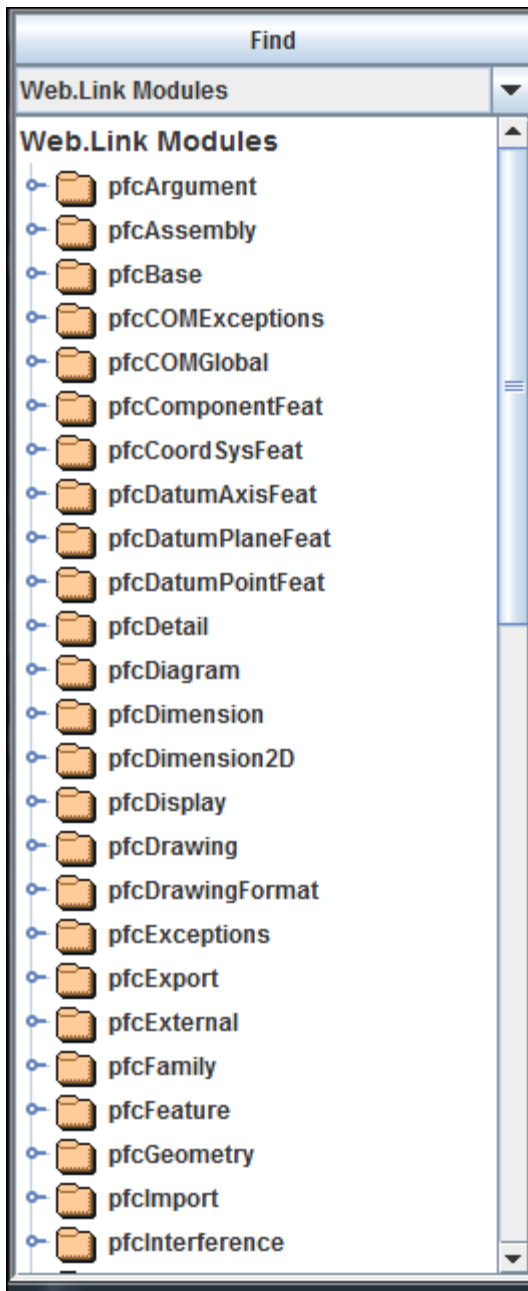
Browsing the Web.Link User's Guide

View the *Web.Link User's Guide* by choosing **Web.Link User's Guide** at the top of the **Modules/Classes/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 following figure shows the collapsed tree layout for the *Web.Link User's Guide* table of contents.

User's Guide Table of Contents
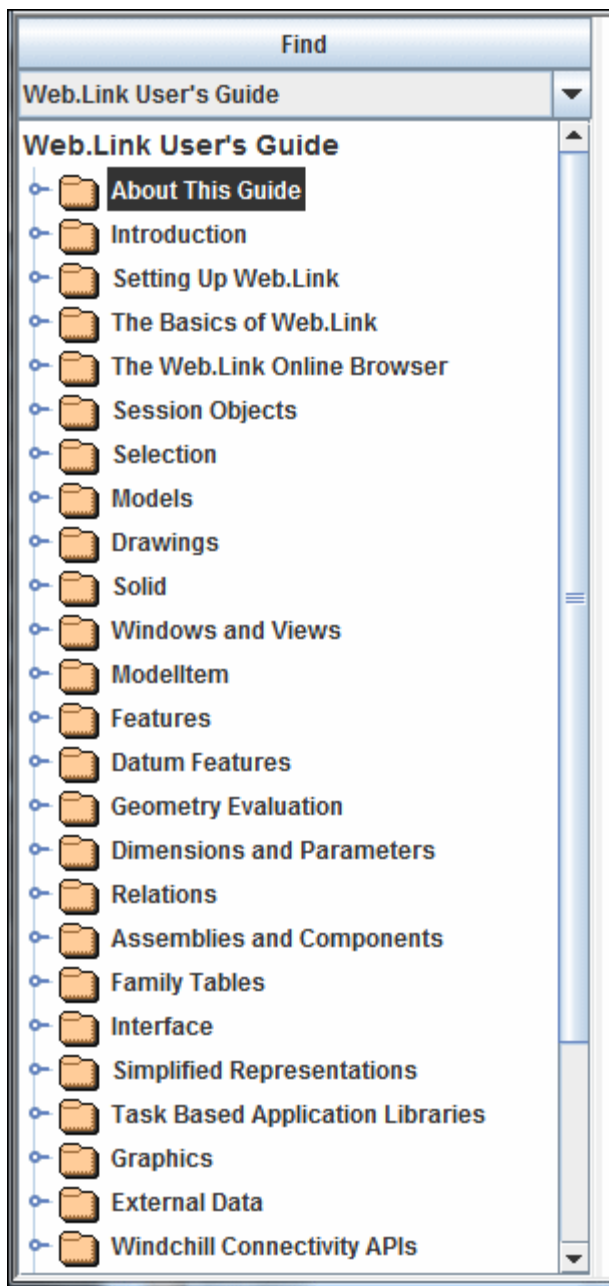
APIWizard Search Feature (Find)

The APIWizard supports searches for specified strings against both the *Web.Link User's Guide* and API definition files. Click the **Find** button on the **Modules/Classes/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.*

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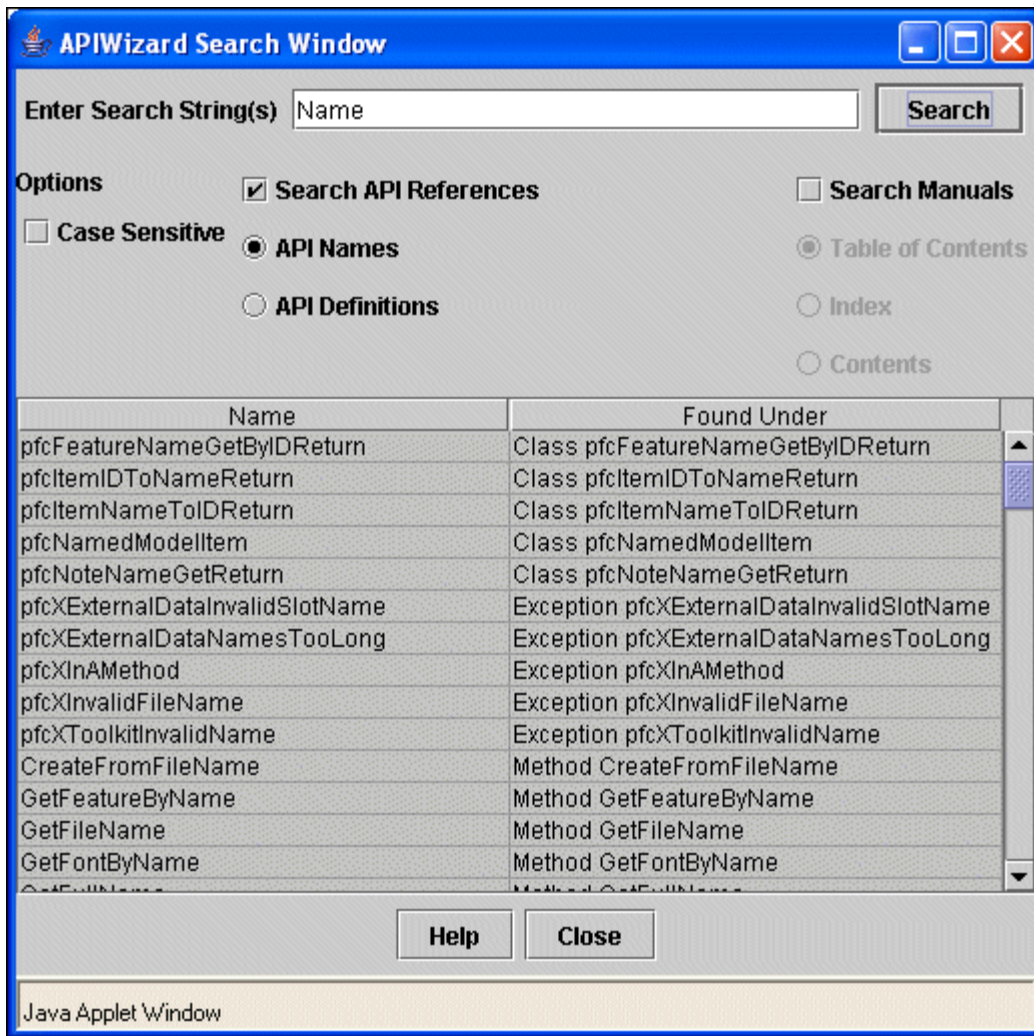
The **Search** dialog box contains the following fields, buttons, and frames:

- **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 **API Definitions** button to search the API method names and definitions for specific strings.

- **Search Manuals**

  Select this button to search the *Web.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 *Web.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, *Web.Link User's Guide* data, or both
- Search of API data by API names only or by API names and definitions
- Search of *Web.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 `pfcModel.FullName` and `pfcModelDescriptor.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 **Modules/Classes/Topic** frame.
- Specify the string or strings to be searched for in the **Enter Search String(s)** 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 Manuals** 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 **Found Under** 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 **Found Under** frames to display the online help data for that string. The APIWizard first updates the **Modules/Classes/Topic** frame tree, and then presents in the **Display** frame the online help data for the selected string.

Session Objects

This section describes how to program on the session level using Web.Link.

Overview of Session Objects

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

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

- Accessing models and windows (described in the Models and Windows chapters).
- Working with the 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.

Getting the Session Object

Method Introduced:

- **MpfcCOMGlobal.GetProESession()**

The method `pfcCOMGlobal.pfcCOMGlobal` `.GetProESession()` gets a `pfcSession` object.

Note

You can make multiple calls to this method but each call will give you a handle to the same object.

Getting Session Information

Methods Introduced:

- **MpfcCOMGlobal.GetProEArguments()**
- **MpfcCOMGlobal.GetProEVersion()**
- **MpfcCOMGlobal.GetProEBuildCode()**

The method `pfcCOMGlobal` `.pfcCOMGlobal.GetProEArguments()` returns an array containing the command line arguments passed to Creo Parametric if these arguments follow one of two formats:

- Any argument starting with a plus sign (+) followed by a letter character.
- Any argument starting with a minus (-) followed by a capitalized letter.

The first argument passed in the array is the full path to the Creo Parametric executable.

The method `pfcCOMGlobal.pfcCOMGlobal` `.GetProEVersion()` returns a string that represent the Creo Parametric version, for example "Wildfire".

The method `pfcCOMGlobal.pfcCOMGlobal` `.GetProEBuildCode()` returns a string that represents the build code of the Creo Parametric session.

Note

The preceding methods can only access information in synchronous mode.

Example: Accessing the Creo Parametric Command Line Arguments

The sample code in the file `pfcProEArgumentsExample.js` located at `<creo_weblink_loadpoint>/weblinkexamples/jscript` describes the use of the `GetProEArguments` method to access the Creo Parametric command line arguments. The first argument is always the full path to the Creo Parametric executable. For this application the next two arguments can be either ("+runtime" or "+development") or ("-Unix" or "-NT"). Based on these values 2 boolean variables are set and passed on to

another method which makes use of this information.

Directories

Methods Introduced:

- **pfcBaseSession.GetCurrentDirectory()**
- **pfcBaseSession.ChangeDirectory()**

The method `pfcBaseSession.GetCurrentDirectory()` returns the absolute path name for the current working directory of Creo Parametric.

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

File Handling

Methods Introduced:

- **pfcBaseSession.ListFiles()**
- **pfcBaseSession.ListSubdirectories()**

The method `pfcBaseSession.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 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 `pfcFileListOpt` 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:

- **pfcBaseSession.GetConfigOptionValues()**
- **pfcBaseSession.SetConfigOption()**
- **pfcBaseSession.LoadConfigFile()**

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

Use the method `pfcBaseSession.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 `pfcBaseSession.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 Creo Parametric.

Macros

Method Introduced:

- **pfcBaseSession.RunMacro()**

The method `pfcSession.BaseSession.RunMacro` runs a macro string. A Web.Link macro string is equivalent to a Creo Parametric mapkey minus the key sequence and the mapkey name. To generate a macro string, create a

mapkey in Creo Parametric. Refer to the 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 Creo Parametric.

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

Colors and Line Styles

Methods Introduced:

- **pfcBaseSession.SetStdColorFromRGB()**
- **pfcBaseSession.GetRGBFromStdColor()**
- **pfcBaseSession.SetTextColor()**
- **pfcBaseSession.SetLineStyle()**

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

Use the method `pfcBaseSession.SetStdColorFromRGB()` to customize any of the Creo Parametric standard colors.

To change the color of any text in the window, use the method `pfcBaseSession.SetTextColor()`.

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

Accessing the Creo Parametric Interface

The `pfcSession` object has methods that work with the Creo Parametric interface. These methods provide access to the message window.

The Text Message File

A text message file is where you define strings that are displayed in the Creo Parametric user interface. This includes the strings on the command buttons that you add to the 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.

Note

Remember that Web.Link applications, as unregistered web pages, do not currently support setting of the Creo Parametric text directory. You can force Creo Parametric to load a message file by registering a Creo Parametric TOOLKIT or J-Link application via the web page, and calling a function or method that requires the message file. Once the file has been loaded by Creo Parametric, Web.Link applications may use any of its keystrings for displaying messages in the message window.

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 Creo Parametric. Duplicate message file names or message key strings can cause Creo Parametric to exhibit unexpected behavior. To avoid conflicts with the names of 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 Creo Parametric only once during a session. If you make a change to the message file while Creo Parametric is running you must exit and restart 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 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.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 `pfcMessageDialogOptions` 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 `pfcMessageDialogOptions.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 `pfcMessageDialogOptions.Buttons` 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 `pfcMessageDialogOptions.DefaultButton` 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 `pfcMessageDialogOptions.DialogLabel` 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 `pfcMessageDialogOptions.MessageDialogType` to set this option.

Accessing the Message Window

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

- Writing Messages to the Message Window

-

Writing Messages to the Message Window

Methods Introduced:

- **pfcSession.UIDisplayMessage()**
- **pfcSession.UIDisplayLocalizedMessage()**
- **pfcSession.UIClearMessage()**

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

The input arguments to the methods `pfcSession.UIDisplayMessage()` and `pfcSession.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:

- **pfcBaseSession.GetMessageContents()**
- **pfcBaseSession.GetLocalizedMessageContents()**

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

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

Message Classification

Messages displayed in Web.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.

Web.Link applications can now display any or all of the following message symbols:

- `Prompt`—This Web.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 Web.Link message is preceded by a blue dot. Info message types contain information such as user requests or feedback from Web.Link or 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 Web.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 Web.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 Web.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 located at 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.UIReadIntMessage()**
- **pfcSession.UIReadRealMessage()**
- **pfcSession.UIReadStringMessage()**

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

The `pfcSession.UIReadIntMessage()` and `pfcSession.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.UIDisplayFeatureParams()**

The method`pfcSession.Session.UIDisplayFeatureParams` forces 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.UIOpenFile()**
- **pfcFileOpenOptions.Create()**
- **pfcFileOpenOptions.FilterString**
- **pfcFileOpenOptions.PreselectedItem**
- **pfcFileUIOptions.DefaultPath**
- **pfcFileUIOptions.DialogLabel**
- **pfcFileUIOptions.Shortcuts**
- **pfcFileOpenShortcut.Create()**
- **pfcFileOpenShortcut.ShortcutName**
- **pfcFileOpenShortcut.ShortcutPath**
- **pfcSession.UISaveFile()**
- **pfcFileSaveOptions.Create()**
- **pfcSession.UISelectDirectory()**
- **pfcDirectorySelectionOptions.Create()**

The method `pfcSession.UIOpenFile()` opens the relevant Creo Parametric dialog box for opening files and browsing directories. The method lets you specify several options through the input arguments `pfcFileOpenOptions` and `pfcFileUIOptions`.

Use the method `pfcFileOpenOptions.Create()` to create a new instance of the `pfcFileOpenOptions` 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`. Use the property `pfcFileOpenOptions.FilterString` to set this option.

- `PreselectedItem`—Specifies the name of an item to preselect in the dialog box. Use the property `pfcFileOpenOptions.PreselectedItem` to set this option.

The `pfcFileUIOptions` object contains the following options:

- `DefaultPath`—Specifies the name of the path to be opened by default in the dialog box. Use the property `pfcFileUIOptions.DefaultPath` to set this option.

- `DialogLabel`—Specifies the title of the dialog box. Use the property `pfcFileUIOptions.DialogLabel` to set this option.

- `Shortcuts`—Specifies an array of file shortcuts of the type `pfcFileOpenShortcut`. 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 property `pfcFileUIOptions.Shortcuts` 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.UISaveFile()` opens the Creo Parametric dialog box for saving a file. The method accepts options similar to `pfcSession.Session.UIOpenFile` through the `pfcFileSaveOptions` and `pfcFileUIOptions` objects. Use the method `pfcFileSaveOptions.Create()` to create a new instance of the `pfcFileSaveOptions` 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.UISelectDirectory()` prompts the user to select a directory using the Creo Parametric dialog box for browsing directories. The method accepts options through the `pfcDirectorySelectionOptions` object which is similar to the `pfcFileUIOptions` 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 `pfcDirectorySelectionOptions` 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 Creo Parametric Navigation Area

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

Adding Custom Web Pages

To add custom Web pages to the navigation area, the Web.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.NavigatorPaneBrowserAdd()**
- **pfcSession.NavigatorPaneBrowserIconSet()**
- **pfcSession.NavigatorPaneBrowserURLSet()**

The method `pfcSession.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.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 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 Creo Parametric icon is used.

    The default search paths for finding the icons are:

    - `<creo_loadpoint>\Common Files\<datecode>\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.

Selection

This section describes how to use Interactive Selection in Web.Link.

Interactive Selection

Methods and Properties Introduced:

- **pfcBaseSession.Select()**
- **pfcSelectionOptions.Create()**
- **pfcSelectionOptions.MaxNumSels**
- **pfcSelectionOptions.OptionKeywords**

The method `pfcBaseSession.Select()` activates the standard Creo Parametric menu structure for selecting objects and returns a `pfcSelections` 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 `pfcSelections` sequence to the method. The returned `pfcSelections` sequence will contain the input sequence and any new objects.

The method `pfcSelectionOptions.Create()` and the property `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*.

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
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 `pfcSelectionOptions.MaxNumSels` must be an Integer.

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

Accessing Selection Data

Properties Introduced:

- **pfcSelection.SelModel**
- **pfcSelection.SelItem**
- **pfcSelection.Path**
- **pfcSelection.Params**
- **pfcSelection.TParam**
- **pfcSelection.Point**
- **pfcSelection.Depth**
- **pfcSelection.SelView2D**
- **pfcSelection.SelTableCell**
- **pfcSelection.SelTableSegment**

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

- For a selected model or model item use `pfcSelection.SelModel` or `pfcSelection.SelItem`.
- For an assembly component use `pfcSelection.Path`.
- For UV parameters of the selection point on a surface use `pfcSelection.Params`.
- For the T parameter of the selection point on an edge or curve use `pfcSelection.TParam`.
- For a three-dimensional point object that contains the selected point use `pfcSelection.Point`.
- For selection depth, in screen coordinates use `pfcSelection.Depth`.
- For the selected drawing view, if the selection was from a drawing, use `pfcSelection.SelView2D`.
- For the selected table cell, if the selection was from a table, use `pfcSelection.SelTableCell`.
- For the selected table segment, if the selection was from a table, use `pfcSelect.Selection.GetSelTableSegment`.

Controlling Selection Display

Methods Introduced:

- **pfcSelection.Highlight()**
- **pfcSelection.UnHighlight()**
- **pfcSelection.Display()**

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

The method `pfcSelection.Highlight()` highlights the selection in the current window. This highlight is the same as the one used by 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.Repaint();` it is not removed if you use `pfcWindow.Refresh()`.

The method `pfcSelection.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 `pfcSelectionExamples.js` located at `<creo_weblink_loadpoint>/weblinkexamples/jscript` demonstrates how to use Web.Link to allow interactive selection.

Programmatic Selection

Web.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 and Properties Introduced:

- **MpfcSelect.CreateModelItemSelection()**
- **MpfcSelect.CreateComponentSelection()**
- **MpfcSelect.CreateSelectionFromString()**
- **pfcSelection.SelItem**
- **pfcSelection.SelTableCell**
- **pfcSelection.SelView2D**

The method `MpfcSelect.CreateModelItemSelection()` creates a selection out of any model item object. It takes a `pfcModelItem` and optionally a `pfcComponentPath` object to identify which component in an assembly the Selection Object belongs to.

The method `MpfcSelect.CreateComponentSelection()` creates a selection out of any component in an assembly. It takes a `pfcComponentPath` object. For more information about `pfcComponentPath` objects, see the section Getting a Solid Object in Solid.

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

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

The selected item using the method `pfcSelection.SelItem`.

The selected table cell using the method `pfcSelection.SelTableCell`.

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 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, Creo Parametric offers the default selection buffer, the Edit selection buffer, and other more specialized buffers. Other 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.

Web.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

Properties Introduced:

- **pfcSession.CurrentSelectionBuffer**
- **pfcSelectionBuffer.Contents**

The property `pfcSession.CurrentSelectionBuffer` 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 property `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:

- **pfcSelectionBuffer.RemoveSelection()**
- **pfcSelectionBuffer.Clear()**

Use the method `pfcSelectionBuffer.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 `pfcSelectionBuffer.Contents`.

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:

- **pfcSelectionBuffer.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.

Models

This section describes how to program on the model level using Web.Link.

Overview of Model Objects

Models can be any Creo Parametric file type, including parts, assemblies, drawings, sections, and notebook. The classes in the module `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 and Properties Introduced:

* **pfcFamilyTableRow.CreateInstance()**
* **pfcSelection.SelModel**
* **pfcBaseSession.GetModel()**
* **pfcBaseSession.CurrentModel**
* **pfcBaseSession.GetActiveModel()**
* **pfcBaseSession.ListModels()**
* **pfcBaseSession.GetByRelationId()**
* **pfcWindow.Model**

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

The property `pfcSelection.SelModel` returns the model that was interactively selected.

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

The property `pfcBaseSession.CurrentModel` returns the current active model.

The method `pfcBaseSession.GetActiveModel()` returns the active Creo Parametric model.

Use the method `pfcBaseSession.ListModels()` to return a sequence of all the models in session.

For more methods that return solid models, refer to the section Solid.

Model Descriptors

Methods and Properties Introduced:

* **pfcModelDescriptor.Create()**
* **pfcModelDescriptor.CreateFromFileName()**
* **pfcModelDescriptor.GenericName**
* **pfcModelDescriptor.InstanceName**
* **pfcModelDescriptor.Type**
* **pfcModelDescriptor.Host**
* **pfcModelDescriptor.Device**
* **pfcModelDescriptor.Path**
* **pfcModelDescriptor.FileVersion**
* **pfcModelDescriptor.GetFullName()**
* **pfcModel.FullName**

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 Creo Parametric to find the specific model you want.

The static utility method `pfcModelDescriptor.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 `pfcModelDescriptor.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:

- **pfcBaseSession.RetrieveModel()**
- **pfcBaseSession.RetrieveModelWithOpts()**
- **pfcBaseSession.OpenFile()**
- **pfcSolid.HasRetrievalErrors()**

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

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

The method `pfcBaseSession.RetrieveModelWithOpts()` retrieves the specified model into the Creo ParametricNGINER 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 `pfcRetrieveModelOptions` object) provides the user with the option to specify simplified representations.

The method `pfcBaseSession.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 property `pfcWindow.Model`.

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 and Properties Introduced:

- **pfcModel.FileName**
- **pfcModel.CommonName**
- **pfcModel.IsCommonNameModifiable()**
- **pfcModel.FullName**
- **pfcModel.GenericName**
- **pfcModel.InstanceName**
- **pfcModel.Origin**

- **pfcModel.RelationId**
- **pfcModel.Descr**
- **pfcModel.Type**
- **pfcModel.IsModified**
- **pfcModel.Version**
- **pfcModel.Revision**
- **pfcModel.Branch**
- **pfcModel.ReleaseLevel**
- **pfcModel.VersionStamp**
- **pfcModel.ListDependencies()**
- **pfcModel.ListDeclaredModels()**
- **pfcModel.CheckIsModifiable()**
- **pfcModel.CheckIsSaveAllowed()**

The property `pfcModel.FileName` retrieves the model file name in the "name"."type" format.

The property `pfcModel.CommonName` retrieves the common name for the model. This name is displayed for the model in Windchill PDMLink.

Use the method `pfcModel.IsCommonNameModifiable()` 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 Windchill PDMLink, or in certain situations if the configuration option `let_proe_rename_pdm_objects` is set to `yes`.

The property `pfcModel.FullName` retrieves the full name of the model in the instance <generic> format.

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

The property `pfcModel.InstanceName` retrieves the name of the model. If the model is an instance, this method retrieves the instance name.

The property `pfcModel.Origin` returns the complete path to the file from which the model was opened. This path can be a location on disk from a Windchill workspace, or from a downloaded URL.

The property `pfcModel.RelationId` retrieves the relation identifier of the specified model. It can be `NULL`.

The property `pfcModel.Descr` 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 `pfcXtoolkitCantOpen` if called on a model instance whose immediate generic is not in session. Handle this exception and typecast the model as `pfcSolid`, which in turn can be typecast as `pfcFamilyMember`, and use the method `pfcFamilyMember.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 property `pfcModel.Type` returns the type of model in the form of the `pfcModelType` 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

The property `pfcModel.IsModified` identifies whether the model has been modified since it was last saved.

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

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

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

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

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

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

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

The method `pfcModel.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 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 and Property Introduced:

- **pfcModel.Backup()**
- **pfcModel.Copy()**
- **pfcModel.CopyAndRetrieve()**
- **pfcModel.Rename()**
- **pfcModel.Save()**
- **pfcModel.Erase()**
- **pfcModel.EraseWithDependencies()**
- **pfcModel.Delete()**
- **pfcModel.Display()**
- **pfcModel.CommonName**

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

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

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

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

The method `pfcModel.Rename()` renames a specified model.

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

The method `pfcModel.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.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.

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

The method `pfcModel.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 `pfcDisplay`.

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

Running ModelCHECK

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

Methods and Properties Introduced:

- **pfcBaseSession.ExecuteModelCheck()**
- **pfcModelCheckInstructions.Create()**
- **pfcModelCheckInstructions.ConfigDir**
- **pfcModelCheckInstructions.Mode**
- **pfcModelCheckInstructions.OutputDir**
- **pfcModelCheckInstructions.ShowInBrowser**
- **pfcModelCheckResults.NumberOfErrors**
- **pfcModelCheckResults.NumberOfWarnings**
- **pfcModelCheckResults.WasModelSaved**

You can run ModelCHECK from an external application using the method `pfcBaseSession.ExecuteModelCheck()`. This method takes the model *Model* on which you want to run ModelCHECK and instructions in the form of the object `pfcModelCheckInstructions` 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 ModelCHECK configuration files are used.

- `Mode`—Specifies the mode in which you want to run 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 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 `pfcModelCheckInstructions.Create()` creates the `pfcModelCheckInstructions` object containing the ModelCHECK instructions described above.

Use the methods and properties `pfcModelCheckInstructions.ConfigDir`, `pfcModelCheckInstructions.Mode`, `pfcModelCheckInstructions.OutputDir`, and `pfcModelCheckInstructions.ShowInBrowser` to modify the ModelCHECK instructions.

The method `pfcSession.BaseSession.ExecuteModelCheck` returns the results of the ModelCHECK run in the form of the `pfcModelCheckResults` 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 properties `pfcModelCheckResults.NumberOfErrors`, `pfcModelCheck.ModelCheckResults.GetNumberOfWarning`, and `pfcModelCheck.ModelCheckResults.GetWasModelSaved` to access the results obtained.

Drawings

This section describes how to program drawing functions using Web.Link.

Overview of Drawings in Web.Link

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

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

Creating Drawings from Templates

Drawing templates simplify the process of creating a drawing using Web.Link. 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:

- **pfcBaseSession.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

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 exception message will list the details for each error including its type, sheet number, view name, and (if applicable) item name,The types of errors are as follows:

- *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.

Example: Drawing Creation from a Template

The sample code in the file `pfcDrawingExamples.js` located at
`<creo_weblink_loadpoint>/weblinkexamples/jscript` creates a new drawing using a predefined template.

Obtaining Drawing Models

This section describes how to obtain drawing models.

Methods Introduced:

- **pfcBaseSession.RetrieveModel()**
- **pfcBaseSession.GetModel()**
- **pfcBaseSession.GetModelFromDescr()**
- **pfcBaseSession.ListModels()**
- **pfcBaseSession.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 `pfcBaseSession.GetModel()` returns a drawing based on its name and type, whereas `pfcBaseSession.GetModelFromDescr()` returns a drawing specified by the model descriptor. The model must be in session.

Use the method `pfcBaseSession.ListModels()` to return a sequence of all the drawings in session.

Drawing Information

Methods and Property Introduced:

- **pfcModel2D.ListModels()**
- **pfcModel2D.GetCurrentSolid()**
- **pfcModel2D.ListSimplifiedReps()**
- **pfcModel2D.TextHeight**

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

The method `pfcModel2D.GetCurrentSolid()` returns the current solid model of the drawing.

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

The property `pfcModel2D.Model2D.GetTextHeight` returns the text height of the drawing.

Drawing Operations

Methods Introduced:

- **pfcModel2D.AddModel()**
- **pfcModel2D.DeleteModel()**
- **pfcModel2D.ReplaceModel()**
- **pfcModel2D.SetCurrentSolid()**
- **pfcModel2D.AddSimplifiedRep()**
- **pfcModel2D.DeleteSimplifiedRep()**
- **pfcModel2D.Regenerate()**
- **pfcModel2D.CreateDrawingDimension()**
- **pfcModel2D.CreateView()**

The method `pfcModel2D.AddModel()` adds a new solid model to the drawing.

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

The method `pfcModel2D.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.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 `pfcSheetOwner.RegenerateSheet()`.

The method `pfcModel2D.AddSimplifiedRep()` associates the drawing with the simplified representation of an assembly

The method `pfcModel2D.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.Regenerate()` to regenerate the drawing draft entities and appearance.

The method `pfcModel2D.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.

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.

Example: Replace Drawing Model Solid with its Generic

The sample code in the file `pfcDrawingExamples.js` located at `<creo_weblink_loadpoint>/weblinkexamples/jscript` 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 Web.Link are identified by the same sheet numbers seen by a 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 and Properties Introduced

- **pfcSheetOwner.GetSheetTransform()**
- **pfcSheetOwner.GetSheetInfo()**
- **pfcSheetOwner.GetSheetScale()**

- **pfcSheetOwner.GetSheetFormat()**
- **pfcSheetOwner.GetSheetFormatDescr()**
- **pfcSheetOwner.GetSheetBackgroundView()**
- **pfcSheetOwner.NumberOfSheets**
- **pfcSheetOwner.CurrentSheetNumber**
- **pfcSheetOwner.GetSheetUnits()**

Superseded Method:

- **pfcSheetOwner.GetSheetData()**

The method `pfcSheetOwner.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 `pfcSheetOwner.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 class `pfcSheetData` have been deprecated. Use the method `pfcSheet.SheetOwner.GetSheetInfo` and theclass `pfcSheetInfo` instead.

The method `pfcSheetOwner.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 `pfcSheetOwner.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 `pfcSheetOwner.GetSheetFormatDescr()` returns the model descriptor of the drawing format used for the specified drawing sheet.

The method `pfcSheetOwner.GetSheetBackgroundView()` returns the view object representing the background view of the sheet specified by the sheet number.

The property `pfcSheetOwner.NumberOfSheets` returns the number of sheets in the model.

The property `pfcSheetOwner.CurrentSheetNumber` 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 `pfcSheetOwner.GetSheetUnits()` returns the units used by the sheet specified by the sheet number.

Drawing Sheet Operations

Methods Introduced:

- **pfcSheetOwner.AddSheet()**
- **pfcSheetOwner.DeleteSheet()**
- **pfcSheetOwner.ReorderSheet()**
- **pfcSheetOwner.RegenerateSheet()**
- **pfcSheetOwner.SetSheetScale()**
- **pfcSheetOwner.SetSheetFormat()**

The method `pfcSheetOwner.AddSheet()` adds a new sheet to the model and returns the number of the new sheet.

The method `pfcSheetOwner.DeleteSheet()` removes the sheet specified by the sheet number from the model.

Use the method `pfcSheetOwner.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 `pfcSheetOwner.RegenerateSheet()` regenerates the sheet specified by the sheet number.

Note

You can regenerate a sheet only if it is displayed.

Use the method `pfcSheetOwner.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.

Example: Listing Drawing Sheets

The sample code in the file `pfcDrawingExamples.js` located at `<creo_weblink_loadpoint>/weblinkexamples/jscript` 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 class `pfcView2D`. 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.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 cl?ss `pfcView2DCreateInstructions` 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 class `pfcGeneralViewCreateInstructions` contains details on how to create general drawing views.

Methods and Properties Introduced:

- **pfcGeneralViewCreateInstructions.Create()**
- **pfcGeneralViewCreateInstructions.ViewModel**
- **pfcGeneralViewCreateInstructions.Location**
- **pfcGeneralViewCreateInstructions.SheetNumber**
- **pfcGeneralViewCreateInstructions.Orientation**
- **pfcGeneralViewCreateInstructions.Exploded**
- **pfcGeneralViewCreateInstructions.Scale**

The method `pfcGeneralViewCreateInstructions.Create()` creates the `pfcGeneralViewCreateInstructions` ata object used for creating general drawing views.

Use the property `pfcGeneralViewCreateInstructions.ViewModel` to assign the solid model to display in the created general drawing view.

Use the property `pfcGeneralViewCreateInstructions.Location` to assign the location in a drawing sheet to place

the created general drawing view.

Use the property `pfcGeneralViewCreateInstructions.SheetNumber` to set the number of the drawing sheet in which the general drawing view is created.

The property `pfcGeneralViewCreateInstructions.Orientation` assigns the orientation of the model in the general drawing view in the form of the `pfcTransform3D` 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 `pfcTransform3D.SetOrigin()`, and remove any scaling factor by normalizing the matrix.

Use the property `pfcGeneralViewCreateInstructions.Exploded` to set the created general drawing view to be an exploded view.

Use the property `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 class `pfcProjectionViewCreateInstructions` contains details on how to create general drawing views.

Methods and Properties Introduced:

- **pfcProjectionViewCreateInstructions.Create()**
- **pfcProjectionViewCreateInstructions.ParentView**
- **pfcProjectionViewCreateInstructions.Location**
- **pfcProjectionViewCreateInstructions.Exploded**

The method `pfcProjectionViewCreateInstructions.Create()` creates the `pfcProjectionViewCreateInstructions` data object used for creating projected drawing views.

Use the property `pfcProjectionViewCreateInstructions.ParentView` to assign the parent view for the projected drawing view.

Use the property `pfcProjectionViewCreateInstructions.Location` to assign the location of the projected drawing view. This location determines how the drawing view will be oriented.

Use the property `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.js` located at `<creo_weblink_loadpoint>/weblinkexamples/jscript` adds a new sheet to a drawing and creates three views of a selected model.

Obtaining Drawing Views

Methods and Property Introduced:

- **pfcSelection.SelView2D**
- **pfcModel2D.List2DViews()**
- **pfcModel2D.GetViewByName()**
- **pfcModel2D.GetViewDisplaying()**
- **pfcSheetOwner.GetSheetBackgroundView()**

The property `pfcSelection.SelView2D` 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.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.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.GetViewDisplaying()` returns the drawing view that displays a dimension. This method returns a null value if the dimension is not displayed in the drawing.

This method works for solid and drawing dimensions.

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

Drawing View Information

Methods and Properties Introduced:

- **pfcChild.DBParent**
- **pfcView2D.GetSheetNumber()**
- **pfcView2D.IsBackground**
- **pfcView2D.GetModel()**
- **pfcView2D.Scale**
- **pfcView2D.GetIsScaleUserdefined()**
- **pfcView2D.Outline**
- **pfcView2D.GetLayerDisplayStatus()**
- **pfcView2D.IsViewdisplayLayerDependent**
- **pfcView2D.Display**
- **pfcView2D.GetTransform()**
- **pfcView2D.Name**
- **pfcView2D.GetSimpRep()**

The inherited property `pfcChild.DBParent`, when called on a `pfcView2D` object, provides the drawing model which owns the specified drawing view. The return value of the method can be downcast to a `pfcModel2D` object.

The method `pfcView2D.GetSheetNumber()` returns the sheet number of the sheet that contains the drawing view.

The property `pfcView2D.IsBackground` returns a value that indicates whether the view is a background view or a model view.

The method `pfcView2D.GetModel()` returns the solid model displayed in the drawing view.

The property `pfcView2D.Scale` returns the scale of the drawing view.

The method `pfcView2D.GetIsScaleUserdefined()` specifies if the drawing has a user-defined scale.

The property `pfcView2D.Outline` returns the position of the view in the sheet in world units.

The method `pfcView2D.GetLayerDisplayStatus()` returns the display status of the specified layer in the drawing view.

The property `pfcView2D.Display` 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.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 property `pfcView2D.Name` 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.js` located at `<creo_weblink_loadpoint>/weblinkexamples/jscript` 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.Translate()**
- **pfcView2D.Delete()**
- **pfcView2D.Regenerate()**
- **pfcView2D.SetLayerDisplayStatus()**

The method `pfcView2D.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.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.

Drawing Dimensions

This section describes the Web.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 class `pfcDimension2D`.

Obtaining Drawing Dimensions

Methods and Property Introduced:

- **pfcModelItemOwner.ListItems()**
- **pfcModelItemOwner.GetItemById()**
- **pfcSelection.SelItem**

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

Creating Drawing Dimensions

Methods Introduced:

- **pfcDrawingDimCreateInstructions.Create()**

- **pfcModel2D.CreateDrawingDimension()**
- **pfcEmptyDimensionSense.Create()**
- **pfcPointDimensionSense.Create()**
- **pfcSplinePointDimensionSense.Create()**
- **pfcTangentIndexDimensionSense.Create()**
- **pfcLinAOCTangentDimensionSense.Create()**
- **pfcAngleDimensionSense.Create()**
- **pfcPointToAngleDimensionSense.Create()**

The method `pfcDimension2D.pfcDrawingDimCreateInstructions_Create` creates an instructions object that describes how to create a drawing dimension using the method `pfcModel2D.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 `pfcEmptyDimensionSense.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 `pfcPointDimensionSense.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 `pfcSplinePointDimensionSense.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 `pfcTangentIndexDimensionSense.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 `pfcLinAOCTangentDimensionSense.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 `pfcAngleDimensionSense.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 and Properties Introduced:

- **pfcDimension2D.IsAssociative**
- **pfcDimension2D.GetIsReference()**
- **pfcDimension2D.IsDisplayed**
- **pfcDimension2D.GetAttachmentPoints()**
- **pfcDimension2D.GetDimensionSenses()**
- **pfcDimension2D.GetOrientationHint()**
- **pfcDimension2D.GetBaselineDimension()**
- **pfcDimension2D.Location**
- **pfcDimension2D.GetView()**
- **pfcDimension2D.GetTolerance()**
- **pfcDimension2D.IsToleranceDisplayed**

The property `pfcDimension2D.IsAssociative` returns whether the dimension or reference dimension in a drawing is associative.

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

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

The method `pfcDimension2D.GetAttachmentPoints()` returns a sequence of attachment points. The dimension senses array returned by the method `pfcDimension2D.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.GetOrientationHint()` returns the orientation hint for placing the drawing dimensions. The orientation hint determines how 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.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 property `pfcDimension2D.Location` returns the placement location of the dimension.

The method `pfcDimension2D.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.GetTolerance()` retrieves the upper and lower tolerance limits of the drawing dimension in the form of the `pfcDimTolerance` 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.ConvertToLinear()**
- **pfcDimension2D.ConvertToOrdinate()**
- **pfcDimension2D.ConvertToBaseline()**
- **pfcDimension2D.SwitchView()**
- **pfcDimension2D.SetTolerance()**
- **pfcDimension2D.EraseFromModel2D()**
- **pfcModel2D.SetViewDisplaying()**

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

The method `pfcDimension2D.ConvertToOrdinate()` converts a linear drawing dimension to an ordinate baseline dimension.

The method `pfcDimension2D.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.SwitchView()` changes the view where a dimension created in the drawing is displayed.

The method `pfcDimension2D.SetTolerance()` assigns the upper and lower tolerance limits of the drawing dimension.

The method `pfcDimension2D.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.js` located at `<creo_weblink_loadpoint>/weblinkexamples/jscript` 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 Web.Link is represented by the class `pfcTable`. It is a child of the `pfcModelItem` class.

Some drawing table methods operate on specific rows or columns. The row and column numbers in Web.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 class `pfcTableCell` is used to represent a drawing table cell.

Creating Drawing Cells

Method Introduced:

- **pfcTableCell.Create()**

The method `pfcTable.pfcTable.TableCell_Create` creates the `pfcTableCell` 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 and Properties Introduced:

- **pfcBaseSession.Select()**
- **pfcSelection.SelItem**
- **pfcSelection.SelTableCell**
- **pfcSelection.SelTableSegment**

Tables may be selected using the method `pfcBaseSession.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 property `pfcSelection.SelItem` returns the selected table handle. It is a model item that can be cast to a `pfcTable` object.

The property `pfcSelection.SelTableCell` returns the row and column indices of the selected table cell.

The property `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:

- **pfcTableCreateInstructions.Create()**
- **pfcTableOwner.CreateTable()**

The method `pfcTableCreateInstructions.Create()` creates the `pfcTableCreateInstructions` 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 `pfcTableCreateInstructions` data object.

Retrieving Drawing Tables

Methods Introduced

- **pfcTableRetrieveInstructions.Create()**
- **pfcTableOwner.RetrieveTable()**

The method `pfcTableRetrieveInstructions.Create()` creates the `pfcTableRetrieveInstructions` data object that describes how to retrieve a drawing table using the method `pfcTable.TableOwner.RetrieveTable`. 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*—The location of left top corner of the retrieved table.

The method `pfcTable.TableOwner.RetrieveTable` retrieves a table specified by the `pfcTableRetrieveInstructions` 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`.

Drawing Tables Information

Methods Introduced:

- **pfcTableOwner.ListTables()**
- **pfcTableOwner.GetTable()**
- **pfcTable.GetRowCount()**
- **pfcTable.GetColumnCount()**
- **pfcTable.CheckIfIsFromFormat()**
- **pfcTable.GetRowSize()**
- **pfcTable.GetColumnSize()**
- **pfcTable.GetText()**
- **pfcTable.GetCellNote()**

The method `pfcTableOwner.ListTables()` returns a sequence of tables found in the model.

The method `pfcTableOwner.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.GetRowCount()` returns the number of rows in the table.

The method `pfcTable.GetColumnCount()` returns the number of columns in the table.

The method `pfcTable.CheckIfIsFromFormat()` verifies if the drawing table was created using the format of the drawing sheet specified by the sheet number. The method returns a true value if the table was created by applying the drawing format.

The method `pfcTable.GetRowSize()` returns the height of the drawing table row specified by the segment identifier and the row number.

The method `pfcTable.GetColumnSize()` returns the width of the drawing table column specified by the segment identifier and the column number.

The method `pfcTable.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.Erase()**
- **pfcTable.Display()**
- **pfcTable.RotateClockwise()**
- **pfcTable.InsertRow()**
- **pfcTable.InsertColumn()**
- **pfcTable.MergeRegion()**
- **pfcTable.SubdivideRegion()**
- **pfcTable.DeleteRow()**
- **pfcTable.DeleteColumn()**
- **pfcTable.SetText()**
- **pfcTableOwner.DeleteTable()**

The method `pfcTable.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.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.RotateClockwise()` rotates a table clockwise by the specified amount of rotation.

The method `pfcTable.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.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.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.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.DeleteRow()` and `pfcTable.DeleteColumn()` delete any specified row or column from the table. The methods also remove the text from the affected cells.

The method `pfcTable.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 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.js` located at `<creo_weblink_loadpoint>/weblinkexamples/jscript` 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 and Property Introduced:

- **pfcSelection.SelTableSegment**

- **pfcTable.GetSegmentCount()**
- **pfcTable.GetSegmentSheet()**
- **pfcTable.MoveSegment()**
- **pfcTable.GetInfo()**

The property `pfcSelection.SelTableSegment` 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.GetSegmentCount()` returns the number of segments in the table.

The method `pfcTable.GetSegmentSheet()` determines the sheet number that contains a specified drawing table segment.

The method `pfcTable.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.IsCommentCell()**
- **pfcTable.GetCellComponentModel()**
- **pfcTable.GetCellReferenceModel()**
- **pfcTable.GetCellTopModel()**
- **pfcTableOwner.UpdateTables()**

The methods `pfcTable.IsCommentCell()`, `pfcTable.GetCellComponentModel()`, `pfcTable.GetCellReferenceModel()`, `pfcTable.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 Web.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 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 Creo Parametric drawing file.

Listing Detail Items

Methods Introduced:

- **pfcModelItemOwner.ListItems()**
- **pfcDetailItemOwner.ListDetailItems()**
- **pfcModelItemOwner.GetItemById()**
- **pfcDetailItemOwner.CreateDetailItem()**

The method `pfcModelItemOwner.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 `pfcDetailItemOwner.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 `pfcModelItemOwner.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:

- **pfcDetailItemOwner.CreateDetailItem()**
- **pfcDetailGroupInstructions.Create()**

The method `pfcDetailItemOwner.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 Web.Link is represented by the class `pfcDetailEntityItem`. It is a child of the `pfcDetailItem` interfaceclass.

The class `pfcDetailEntityInstructions` contains specific information used to describe a detail entity item.

Instructions

Methods and Properties Introduced:

- **pfcDetailEntityInstructions.Create()**
- **pfcDetailEntityInstructions.Geometry**
- **pfcDetailEntityInstructions.IsConstruction**
- **pfcDetailEntityInstructions.Color**
- **pfcDetailEntityInstructions.FontName**
- **pfcDetailEntityInstructions.Width**
- **pfcDetailEntityInstructions.View**

The method `pfcDetailEntityInstructions.Create()` creates an instructions object that describes how to construct a detail entity, for use in the methods `pfcDetailItemOwner.CreateDetailItem()`, `pfcDetailSymbolDefItem.CreateDetailItem()`, and `pfcDetailEntityItem.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.List2DViews()` or a drawing sheet background view returned by the method `pfcSheetOwner.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 `pfcDetailEntityInstructions` object do not take effect until that instructions object is used to modify the entity using `pfcDetail.DetailEntityItem.Modify`.

The property `pfcDetailEntityInstructions.Geometry` returns the geometry of the detail entity item.

The property `pfcDetailEntityInstructions.IsConstruction` returns a value that specifies whether the entity is a construction entity.

The property `pfcDetailEntityInstructions.Color` returns the color of the detail entity item.

The property `pfcDetailEntityInstructions.FontName` returns the line style used to draw the entity. The method returns a null value if the default line style is used.

The property `pfcDetailEntityInstructions.Width` returns the value of the width of the entity line. The method returns a null value if the default line width is used.

The property `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.

Example: Create a Draft Line with Predefined Color

The sample code in the file `pfcDrawingExamples.js` located at `<creo_weblink_loadpoint>/weblinkexamples/jscript` shows a utility that creates a draft line in one of the colors predefined in Creo Parametric.

Detail Entities Information

Methods and Property Introduced:

- **pfcDetailEntityItem.GetInstructions()**
- **pfcDetailEntityItem.SymbolDef**

The method `pfcDetailEntityItem.GetInstructions()` returns the instructions data object that is used to construct the detail entity item.

The property `pfcDetail.DetailEntityItem.GetSymbolDef` returns the symbol definition that contains the entity. This property returns a null value if the entity is not a part of a symbol definition.

Detail Entities Operations

Methods Introduced:

- **pfcDetailEntityItem.Draw()**
- **pfcDetailEntityItem.Erase()**
- **pfcDetailEntityItem.Modify()**

The method `pfcDetailEntityItem.Draw()` temporarily draws a detail entity item, so that it is removed during the next draft regeneration.

The method `pfcDetailEntityItem.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 Creo Parametric. You can create and insert supported OLE objects into a two-dimensional 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 and Properties Introduced:

- **pfcDetailOLEObject.ApplicationType**
- **pfcDetailOLEObject.Outline**
- **pfcDetailOLEObject.Path**
- **pfcDetailOLEObject.Sheet**

The method `pfcDetailOLEObject.ApplicationType` returns the type of the OLE object as a string, for example, `Microsoft Word Document`.

The property `pfcDetailOLEObject.Outline` returns the extent of the OLE object embedded in the drawing.

The property `pfcDetailOLEObject.Path` returns the path to the external file for each OLE object, if it is linked to an external file.

The property `pfcDetail.DetailOLEObject.GetSheet` returns the sheet number for the OLE object.

Detail Notes

A detail note in Web.Link is represented by the class `pfcDetailNoteItem`. It is a child of the `pfcDetailItem` class.

The class `pfcDetailNoteInstructions` contains specific information that describes a detail note.

Instructions

Methods and Properties Introduced:

- **pfcDetailNoteInstructions.Create()**
- **pfcDetailNoteInstructions.TextLines**
- **pfcDetailNoteInstructions.IsDisplayed**
- **pfcDetailNoteInstructions.IsReadOnly**
- **pfcDetailNoteInstructions.IsMirrored**
- **pfcDetailNoteInstructions.Horizontal**
- **pfcDetailNoteInstructions.Vertical**
- **pfcDetailNoteInstructions.Color**
- **pfcDetailNoteInstructions.Leader**
- **pfcDetailNoteInstructions.TextAngle**

The method `pfcDetailNoteInstructions.Create()` creates a data object that describes how a detail note item should be constructed when passed to the methods `pfcDetailItemOwner.CreateDetailItem()`,

`pfcDetailSymbolDefItem.CreateDetailItem()`, or `pfcDetailNoteItem.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 `apfcDetailNoteInstructions` object do not take effect until that instructions object is used to modify the note using `pfcDetailNoteItem.Modify`

The property `pfcDetailNoteInstructions.TextLines` returns the description of text line contents in the note.

The property `pfcDetailNoteInstructions.IsDisplayed` returns a boolean indicating if the note is currently displayed.

The property `pfcDetailNoteInstructions.IsReadOnly` determines whether the note can be edited by the user.

The property `pfcDetailNoteInstructions.IsMirrored` determines whether the note is mirrored.

The property `pfcDetailNoteInstructions.Horizontal` returns the value of the horizontal justification of the note.

The property `pfcDetailNoteInstructions.Vertical` returns the value of the vertical justification of the note.

The property `pfcDetailNoteInstructions.Color` returns the color of the detail note item. The method returns a null value to represent the default drawing color.

The property `pfcDetailNoteInstructions.Leader` returns the locations of the detail note item and information about the leaders.

The property `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.

Example: Create Drawing Note at Specified Location with Leader to Surface and Surface Name

The sample code in the file `pfcDrawingExamples.js` located at `<creo_weblink_loadpoint>/weblinkexamples/jscript` 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 and Property Introduced:

- **pfcDetailNoteItem.GetInstructions()**
- **pfcDetailNoteItem.SymbolDef**
- **pfcDetailNoteItem.GetLineEnvelope()**
- **pfcDetailNoteItem.GetModelReference()**

The method `pfcDetailNoteItem.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

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 property `pfcDetailNoteItem.SymbolDef` 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 `pfcDetailNoteItem.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.
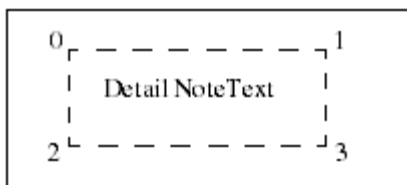
The ordering of the points is maintained even if the notes are mirrored or are at an angle.

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:

- **pfcDetailNoteItem.Draw()**
- **pfcDetailNoteItem.Show()**
- **pfcDetailNoteItem.Erase()**
- **pfcDetailNoteItem.Remove()**
- **pfcDetailNoteItem.Modify()**

The method `pfcDetailNoteItem.Draw()` temporarily draws a detail note item, so that it is removed during the next draft regeneration.

The method `pfcDetailNoteItem.Show()` displays the note item, such that it is repainted during the next draft regeneration.

The method `pfcDetailNoteItem.Erase()` undraws a detail note item temporarily, so that it is redrawn during the next draft regeneration.

The method `pfcDetailNoteItem.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 Web.Link is represented by the class `pfcDetailGroupItem`. It is a child of the `pfcDetailItem` class.

The class `pfcDetailGroupInstructions` contains information used to describe a detail group item.

Instructions

Method and Properties Introduced:

- **pfcDetailGroupInstructions.Create()**
- **pfcDetailGroupInstructions.Name**
- **pfcDetailGroupInstructions.Elements**
- **pfcDetailGroupInstructions.IsDisplayed**

The method `pfcDetailGroupInstructions.Create()` creates an instruction data object that describes how to construct a detail group for use in `pfcDetailItemOwner.CreateDetailItem()` and `pfcDetailGroupItem.Modify()`.

Note

Changes to the values of a `pfcDetailGroupInstructions` object do not take effect until that instructions object is used to modify the group using `pfcDetailGroupItem.Modify`.

The property `pfcDetailGroupInstructions.Name` returns the name of the detail group.

The property `pfcDetailGroupInstructions.Elements` returns the sequence of the detail items(notes, groups and entities) contained in the group.

The property `pfcDetail.DetailGroupInstructions.GetIsDisplayed` returns whether the detail group is displayed in the drawing.

Detail Groups Information

Method Introduced:

- **pfcDetailGroupItem.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:

- **pfcDetailGroupItem.Draw()**
- **pfcDetailGroupItem.Erase()**
- **pfcDetailGroupItem.Modify()**

The method `pfcDetailGroupItem.Draw()` temporarily draws a detail group item, so that it is removed during the next draft generation.

The method `pfcDetailGroupItem.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.js` located at `<creo_weblink_loadpoint>/weblinkexamples/jscript` creates a group from a set of selected detail items.

Detail Symbols

Detail Symbol Definitions

A detail symbol definition in Web.Link is represented by the class `pfcDetailSymbolDefItem`. It is a child of the `pfcDetailItem` class.

The class `pfcDetailSymbolDefInstructions` 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 and Properties Introduced:

- **pfcDetailSymbolDefInstructions.Create()**
- **pfcDetailSymbolDefInstructions.SymbolHeight**
- **pfcDetailSymbolDefInstructions.HasElbow**
- **pfcDetailSymbolDefInstructions.IsTextAngleFixed**
- **pfcDetailSymbolDefInstructions.ScaledHeight**
- **pfcDetailSymbolDefInstructions.Attachments**
- **pfcDetailSymbolDefInstructions.FullPath**
- **pfcDetailSymbolDefInstructions.Reference**

The method `pfcDetailSymbolDefInstructions.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 `pfcDetailSymbolDefInstructions` object do not take effect until that instructions object is used to modify the definition using the method `pfcDetailSymbolDefItem.Modify()`.

The property `pfcDetailSymbolDefInstructions.SymbolHeight` 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 property `pfcDetailSymbolDefInstructions.HasElbow` determines whether the symbol definition includes an elbow.

The property `pfcDetailSymbolDefInstructions.IsTextAngleFixed` returns whether the text of the angle is fixed.

The property `pfcDetailSymbolDefInstructions.ScaledHeight` returns the height of the symbol definition in inches.

The property `pfcDetailSymbolDefInstructions.Attachments` returns the value of the sequence of the possible instance attachment points for the symbol definition.

The property `pfcDetailSymbolDefInstructions.FullPath` returns the value of the complete path of the symbol definition file.

The property `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`.

Detail Symbol Definitions Information

Methods Introduced:

- **pfcDetailSymbolDefItem.ListDetailItems()**
- **pfcDetailSymbolDefItem.GetInstructions()**

The method `pfcDetailSymbolDefItem.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:

- **pfcDetailSymbolDefItem.CreateDetailItem()**
- **pfcDetailSymbolDefItem.Modify()**

The method `pfcDetailSymbolDefItem.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:

- **pfcDetailItemOwner.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 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 Web.Link is represented by the class `pfcDetailSymbolInstItem`. It is a child of the `pfcDetailItem` class.

The class `pfcDetail.DetailSymbolInstInstructions` contains information that describes a symbol instance. It can be used when creating symbol instances and while accessing existing groups.

Instructions

Methods and Properties Introduced:

- **pfcDetailSymbolInstInstructions.Create()**
- **pfcDetailSymbolInstInstructions.IsDisplayed**
- **pfcDetailSymbolInstInstructions.Color**
- **pfcDetailSymbolInstInstructions.SymbolDef**
- **pfcDetailSymbolInstInstructions.AttachOnDefType**
- **pfcDetailSymbolInstInstructions.DefAttachment**
- **pfcDetailSymbolInstInstructions.InstAttachment**
- **pfcDetailSymbolInstInstructions.Angle**
- **pfcDetailSymbolInstInstructions.ScaledHeight**
- **pfcDetailSymbolInstInstructions.TextValues**
- **pfcDetailSymbolInstInstructions.CurrentTransform**
- **pfcDetailSymbolInstInstructions.SetGroups()**

The method `pfcDetailSymbolInstInstructions.Create()` creates a data object that contains information about the placement of a symbol instance.

Note

Changes to the values of a `pfcDetailSymbolInstInstructions` object do not take effect until that instructions object is used to modify the instance using `pfcDetailSymbolInstItem.Modify`.

The property `pfcDetailSymbolInstInstructions.IsDisplayed` returns a value that specifies whether the instance of the symbol is displayed.

The property `pfcDetailSymbolInstInstructions.Color` 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 property `pfcDetailSymbolInstInstructions.SymbolDef` returns the symbol definition used for the instance.

The property `pfcDetailSymbolInstInstructions.AttachOnDefType` 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 property `pfcDetailSymbolInstInstructions.DefAttachment` returns the value that represents the way in which the instance is attached to the symbol definition.

The property `pfcDetailSymbolInstInstructions.InstAttachment` returns the value of the attachment of the instance that includes location and leader information.

The property `pfcDetailSymbolInstInstructions.Angle` 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 property `pfcDetailSymbolInstInstructions.ScaledHeight` 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 Creo Parametric User Interface.

Note

The scaled height obtained using the above property is partially based on the properties of the symbol definition assigned using the property `pfcDetail.DetailSymbolInstInstructions.GetSymbolDef`. Changing the

symbol definition may change the calculated value for the scaled height.

The property `pfcDetailSymbolInstInstructions.TextValues` returns the sequence of variant text values used while placing the symbol instance.

The property `pfcDetailSymbolInstInstructions.CurrentTransform` returns the coordinate transformation matrix to place the symbol instance.

The method `pfcDetail.DetailSymbolInstInstructions.SetGroups` `pfcDetailSymbolGroupOption`

- `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 for more information on detail symbol groups.

Detail Symbol Instances Information

Method Introduced:

- **pfcDetailSymbolInstItem.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:

- **pfcDetailSymbolInstItem.Draw()**
- **pfcDetailSymbolInstItem.Erase()**
- **pfcDetailSymbolInstItem.Show()**
- **pfcDetailSymbolInstItem.Remove()**
- **pfcDetailSymbolInstItem.Modify()**

The method `pfcDetailSymbolInstItem.Draw()` draws a symbol instance temporarily to be removed on the next draft regeneration.

The method `pfcDetailSymbolInstItem.Erase()` undraws a symbol instance temporarily from the display to be redrawn on the next draft generation.

The method `pfcDetailSymbolInstItem.Show()` displays a symbol instance to be repainted on the next draft regeneration.

The method `pfcDetailSymbolInstItem.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.js` located at `<creo_weblink_loadpoint>/weblinkexamples/jscript` 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 `pfcDrawingExamples5.js` located at `<creo_weblink_loadpoint>/weblinkexamples/jscript` creates a free instance of a symbol definition with drawing unit heights, variable text and groups.

Detail Symbol Groups

A detail symbol group in Web.Link is represented by the class `pfcDetailSymbolGroup`. It is a child of the

`pfcObject` class. A detail symbol group is accessible only as a part of the contents of a detail symbol definition or instance.

The class `pfcDetailSymbolGroupInstructions` 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 and Properties Introduced:

- **pfcDetailSymbolGroupInstructions.Create()**
- **pfcDetailSymbolGroupInstructions.Items**
- **pfcDetailSymbolGroupInstructions.Name**

The method `pfcDetailSymbolGroupInstructions.Create()` creates the `pfcDetailSymbolGroupInstructions` 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 `pfcDetailSymbolGroupInstructions` data object do not take effect until this object is used to modify the instance using the method `pfcDetailSymbolGroup.Modify.`

The property `pfcDetailSymbolGroupInstructions.Items` returns the list of detail items included in the symbol group.

The property `pfcDetail.DetailSymbolGroupInstructions.GetName` returns the name of the symbol group.

Detail Symbol Group Information

Methods Introduced:

- **pfcDetailSymbolGroup.GetInstructions()**
- **pfcDetailSymbolGroup.ParentGroup**
- **pfcDetailSymbolGroup.ParentDefinition**
- **pfcDetailSymbolGroup.ListChildren()**
- **pfcDetailSymbolDefItem.ListSubgroups()**
- **pfcDetailSymbolDefItem.IsSubgroupLevelExclusive()**
- **pfcDetailSymbolInstItem.ListGroups()**

The method `pfcDetailSymbolGroup.GetInstructions()` returns the `pfcDetailSymbolGroupInstructions` data object that describes how to construct a symbol group.

The method `pfcDetailSymbolGroup.ParentGroup` returns the parent symbol group to which a given symbol group belongs.

The method `pfcDetailSymbolGroup.ParentDefinition` returns the symbol definition of a given symbol group.

The method `pfcDetailSymbolGroup.ListChildren()` lists the subgroups of a given symbol group.

The method `pfcDetailSymbolDefItem.ListSubgroups()` lists the subgroups of a given symbol group stored in the symbol definition at the indicated level.

The method `pfcDetailSymbolDefItem.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 `pfcSymbolGroupFilter` 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:

- **pfcDetailSymbolGroup.Delete()**
- **pfcDetailSymbolGroup.Modify()**
- **pfcDetailSymbolDefItem.CreateSubgroup()**
- **pfcDetailSymbolDefItem.SetSubgroupLevelExclusive()**
- **pfcDetailSymbolDefItem.SetSubgroupLevelIndependent()**

The method `pfcDetailSymbolGroup.Delete()` deletes the specified symbol group from the symbol definition. This method does not delete the entities contained in the group.

The method `pfcDetailSymbolGroup.Modify()` modifies the specified symbol group based on the `pfcDetailSymbolGroupInstructions` data object that contains information about the modifications that can be made to the symbol group.

The method `pfcDetailSymbolDefItem.CreateSubgroup()` creates a new subgroup in the symbol definition at the indicated level below the parent group.

The method `pfcDetailSymbolDefItem.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 Web.Link is represented by the class `pfcAttachment`. 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:

- **pfcAttachment.GetType()**

The method `pfcDetail.Attachment.GetType` returns the `pfcAttachmentType` 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 class `pfcFreeAttachment`. It is a child of the `pfcAttachment` class.

Properties Introduced:

- **pfcFreeAttachment.AttachmentPoint**
- **pfcFreeAttachment.View**

The property `pfcFreeAttachment.AttachmentPoint` 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.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.

Parametric Attachment

The `ATTACH_PARAMETRIC` detail attachment type is represented by the class `pfcParametricAttachment`. It is a child of the `pfcAttachment` class.

Property Introduced:

- **pfcParametricAttachment.AttachedGeometry**

The property `pfcDetail.ParametricAttachment.GetAttachedGeometry` returns the `pfcSelection` object representing the item to which the detail attachment is attached. This includes the drawing view in which the attachment is made.

Offset Attachment

The `ATTACH_OFFSET` detail attachment type is represented by the class `pfcOffsetAttachment`. It is a child of the `pfcAttachment` class.

Properties Introduced:

- **pfcOffsetAttachment.AttachedGeometry**
- **pfcOffsetAttachment.AttachmentPoint**

The property `pfcOffsetAttachment.AttachedGeometry` returns the `pfcSelection` 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 property `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 class `pfcUnsupportedAttachment`. It is a child of the `pfcAttachment` class.

property Introduced:

- **pfcUnsupportedAttachment.AttachmentPoint**

The property `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.

Solid

Most of the objects and methods in Web.Link are used with solid models (parts and assemblies). Because solid objects inherit from the interface `pfcModel`, you can use any of the `pfcModel` methods on any `pfcSolid`, `pfcPart`, or `pfcAssembly` object.

Getting a Solid Object

Methods and Properties Introduced:

- **pfcBaseSession.CreatePart()**
- **pfcBaseSession.CreateAssembly()**
- **pfcComponentPath.Root**
- **pfcComponentPath.Leaf**
- **pfcMFG.GetSolid()**

The methods `pfcBaseSession.CreatePart()` and `pfcBaseSession.CreateAssembly()` create new solid models with the names you specify.

The properties `pfcComponentPath.Root` and `pfcComponentPath.Leaf` 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.CreateUDFGroup()` on the storage solid.

Solid Information

Properties Introduced:

- **pfcSolid.RelativeAccuracy**
- **pfcSolid.AbsoluteAccuracy**

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 and Properties Introduced:

- **pfcSolid.Regenerate()**
- **pfcRegenInstructions.Create()**
- **pfcRegenInstructions.AllowFixUI**
- **pfcRegenInstructions.ForceRegen**
- **pfcRegenInstructions.FromFeat**
- **pfcRegenInstructions.RefreshModelTree**
- **pfcRegenInstructions.ResumeExcludedComponents**
- **pfcRegenInstructions.UpdateAssemblyOnly**
- **pfcRegenInstructions.UpdateInstances**
- **pfcSolid.GeomOutline**
- **pfcSolid.EvalOutline()**
- **pfcSolid.IsSkeleton**

- **pfcSolid.ListGroups()**

The method `pfcSolid.Regenerate()` causes the solid model to regenerate according to the instructions provided in the form of the `pfcRegenInstructions` 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, Web.Link does not support regeneration in this mode. The method `pfcSolid.Solid.Regenerate` throws an exception `pfcXToolkitBadContext`, if 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 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 Creo Parametric session.

The `pfcRegenInstructions` 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 property `pfcRegenInstructions.AllowFixUI` to modify this parameter.

- *ForceRegen*—Creo Parametric

  Use the property `pfcRegenInstructions.ForceRegen` to modify this parameter.

- *FromFeat*—Not currently used. This parameter is reserved for future use.

  Use the property `pfcRegenInstructions.FromFeat` to modify this parameter.

- *RefreshModelTree*—Creo Parametric **Model Tree**

  Use the property `pfcRegenInstructions.RefreshModelTree` to modify this parameter.

- *ResumeExcludedComponents*—Creo Parametric

  Use the property `pfcRegenInstructions.ResumeExcludedComponents` 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 property `pfcRegenInstructions.UpdateAssemblyOnly` 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 property `pfcRegenInstructions.UpdateInstances` to modify this parameter.

The property `pfcSolid.GeomOutline` returns the three-dimensional bounding box for the specified solid. The method `pfcSolid.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 property `pfcSolid.IsSkeleton` 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 Web.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 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:
  - Creo Parametric Default—`Inch lbm Second`. This is the default system followed by Creo Parametric.
  - `FPS—Foot Pound Second`
  - `IPS—Inch Pound Second`
  - `mmNS—Millimeter Newton Second`

In Web.Link, the system of units followed by the model is represented by the interface `pfcUnits.UnitSystem`.

Accessing Individual Units

Methods and Properties Introduced:

- **pfcSolid.ListUnits()**
- **pfcSolid.GetUnit()**
- **pfcUnit.Name**
- **pfcUnit.Expression**
- **pfcUnit.Type**
- **pfcUnit.IsStandard**
- **pfcUnit.ReferenceUnit**
- **pfcUnit.ConversionFactor**
- **pfcUnitConversionFactor.Offset**
- **pfcUnitConversionFactor.Scale**

The method `pfcSolid.ListUnits()` returns the list of units available to the specified model.

The method `pfcSolid.GetUnit()` retrieves the unit, based on its name or expression for the specified model in the form of the `pfcUnit` object.

The property `pfcUnit.Name` returns the name of the unit.

The property `pfcUnit.Expression` 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 property `pfcUnit.Type` returns the type of quantity represented by the unit in terms of the `pfcUnitType` 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 property `pfcUnit.IsStandard` 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 property `pfcUnit.ReferenceUnit` returns a reference unit (one of the available system units) in terms of the `pfcUnit` object.

The property `pfcUnit.ConversionFactor` identifies the relation of the unit to its reference unit in terms of the `pfcUnitConversionFactor` 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

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 properties `pfcUnitConversionFactor.Offset` and `pfcUnits.UnitConversionFactor.GetScale` to retrieve the unit conversion factors listed above.

Modifying Individual Units

Methods and Properties Introduced:

- **pfcUnit.Modify()**
- **pfcUnit.Delete()**

The method `pfcUnit.Modify()` modifies the definition of a unit by applying a new conversion factor specified by the `pfcUnitConversionFactor` object and a reference unit.

The method `pfcUnits.Unit.Delete` deletes the unit.

Note

You can delete only custom units and not standard units.

Creating a New Unit

Methods Introduced:

- **pfcSolid.CreateCustomUnit()**
- **pfcUnitConversionFactor.Create()**

The method `pfcSolid.CreateCustomUnit()` creates a custom unit based on the specified name, the conversion factor given by the `pfcUnitConversionFactor` object, and a reference unit.

The method `pfcUnits.pfcUnits.UnitConversionFactor_Create` creates the `pfcUnitConversionFactor` object containing the unit conversion factors.

Accessing Systems of Units

Methods and Properties Introduced:

- **pfcSolid.ListUnitSystems()**
- **pfcSolid.GetPrincipalUnits()**
- **pfcUnitSystem.GetUnit()**
- **pfcUnitSystem.Name**
- **pfcUnitSystem.Type**
- **pfcUnitSystem.IsStandard**

The method `pfcSolid.ListUnitSystems()` returns the list of unit systems available to the specified model.

The method `pfcSolid.GetPrincipalUnits()` returns the system of units assigned to the specified model in the form of the `pfcUnitSystem` object.

The method `pfcUnitSystem.GetUnit()` retrieves the unit of a particular type used by the unit system.

The property `pfcUnitSystem.Name` returns the name of the unit system.

The property `pfcUnitSystem.Type` returns the type of the unit system in the form of the `pfcUnitSystemType` 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.

The property `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

Method Introduced:

- **pfcUnitSystem.Delete()**

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.

Creating a New System of Units

Method Introduced:

- **pfcSolid.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 `pfcUnitSystemType` object, and the types of units specified by the `pfcUnits` sequence to use for each of the base measurement types (length, force or mass, and temperature).

Conversion to a New Unit System

Methods and Properties Introduced:

- **pfcSolid.SetPrincipalUnits()**
- **pfcUnitConversionOptions.Create()**
- **pfcUnitConversionOptions.DimensionOption**
- **pfcUnitConversionOptions.IgnoreParamUnits**

The method `pfcSolid.SetPrincipalUnits()` changes the principal system of units assigned to the solid model based on the the unit conversion options specified by the `pfcUnitConversionOptions` object. The method

`pfcUnitConversionOptions.Create()` creates the `pfcUnitConversionOptions` 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 property `pfcUnitConversionOptions.DimensionOption` 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 property `pfcUnits.UnitConversionOptions.SetIgnoreParamUnits` to modify this attribute.

Mass Properties

Method Introduced:

- **pfcSolid.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 an object containing 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.js` located at `<creo_weblink_loadpoint>/weblinkexamples/jscript` retrieves a `MassProperty` object from a specified solid model. The solid's mass, volume, and center of gravity point are then printed.

Annotations

Methods and Properties Introduced:

- **pfcNote.Lines**
- **pfcNote.GetText()**
- **pfcNote.URL**
- **pfcNote.Display()**
- **pfcNote.Delete()**
- **pfcNote.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 property `pfcNote.Lines` returns the text contained in the 3D model note.

The method `pfcNote.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 property `pfcNote.URL` returns the URL stored in the 3D model note.

The method `pfcNote.Display()` forces the display of the model note.

The method `pfcNote.Delete()` deletes a model note.

The method `pfcNote.Note.GetOwner` returns the solid model owner of the note.

Cross Sections

Methods Introduced:

- **pfcSolid.ListCrossSections()**
- **pfcSolid.GetCrossSection()**
- **pfcXSection.GetName()**
- **pfcXSection.SetName()**
- **pfcXSection.GetXSecType()**
- **pfcXSection.Delete()**
- **pfcXSection.Display()**
- **pfcXSection.Regenerate()**

The method `pfcSolid.ListCrossSections()` returns a sequence of cross section objects represented by the Xsection interface. The method `pfcSolid.GetCrossSection()` searches for a cross section given its name.

The method `pfcXSection.GetName()` returns the name of the cross section in Creo Parametric. The method `pfcXSection.SetName()` modifies the cross section name.

The method `pfcXSection.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.Delete()` deletes a cross section.

The method `pfcXSection.Display()` forces a display of the cross section in the window.

The method `pfcXSection.XSection.Regenerate` regenerates a cross section.

Materials

Web.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 and Properties Introduced:

- **pfcMaterial.Save()**
- **pfcMaterial.Delete()**
- **pfcPart.CurrentMaterial**
- **pfcPart.ListMaterials()**
- **pfcPart.CreateMaterial()**
- **pfcPart.RetrieveMaterial()**

The method `pfcMaterial.Save()` writes to a material file that can be imported into any Creo Parametric part.

The method `pfcMaterial.Delete()` removes material from the part.

The property `pfcPart.CurrentMaterial` returns and sets the material assigned to the part.

Note

By default, while assigning a material to a sheetmetal part, the property `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 property `pfcPart.Part.SetCurrentMaterial` may change the model display, if the new material has a default appearance assigned to it.

The property may also change the family table, if the parameter `PTC_MATERIAL_NAME` is a part of the family table.

The method `pfcPart.ListMaterials()` returns a list of the materials available in the part.

The method `pfcPart.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

and properties Introduced:

* **pfcMaterial.StructuralMaterialType**
* **pfcMaterial.ThermalMaterialType**
* **pfcMaterial.SubType**
* **pfcMaterial.PermittedSubTypes**

The property `pfcMaterial.StructuralMaterialType` sets 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.

The property `pfcMaterial.ThermalMaterialType` sets 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.

The property `pfcMaterial.SubType` sets the subtype for the `MTL_ISOTROPIC` material type.

Use the property `pfcPart.Material.GetPermittedSubTypes` to retrieve a list of the permitted string values for the material subtype.

Accessing Material Properties

The methods and properties listed in this section enable you to access material properties.

Methods and Properties Introduced:

* **pfcMaterialProperty.Create()**

- **pfcMaterial.GetPropertyValue()**
- **pfcMaterial.SetPropertyValue()**
- **pfcMaterial.SetPropertyUnits()**
- **pfcMaterial.RemoveProperty()**
- **pfcMaterial.Description**
- **pfcMaterial.FatigueType**
- **pfcMaterial.PermittedFatigueTypes**
- **pfcMaterial.FatigueMaterialType**
- **pfcMaterial.PermittedFatigueMaterialTypes**
- **pfcMaterial.FatigueMaterialFinish**
- **pfcMaterial.PermittedFatigueMaterialFinishes**
- **pfcMaterial.FailureCriterion**
- **pfcMaterial.PermittedFailureCriteria**
- **pfcMaterial.Hardness**
- **pfcMaterial.HardnessType**
- **pfcMaterial.Condition**
- **pfcMaterial.BendTable**
- **pfcMaterial.CrossHatchFile**
- **pfcMaterial.MaterialModel**
- **pfcMaterial.PermittedMaterialModels**
- **pfcMaterial.ModelDefByTests**

The method `pfcMaterialProperty.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 `pfcMaterial.GetPropertyValue()` returns the value and the units of the material property.

Use the method `pfcMaterial.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 `pfcMaterial.SetPropertyUnits()` to set the units of the material property.

Use the method `pfcMaterial.RemoveProperty()` to remove the material property.

Material properties that are non-numeric can be accessed using the following properties.

The property `pfcMaterial.Description` setsthe description string for the material.

The property `pfcMaterial.FatigueType` and sets the valid fatigue type for the material.

Use the property `pfcMaterial.PermittedFatigueTypes` to get a list of the permitted string values for the fatigue type.

The property `pfcMaterial.FatigueMaterialType` sets the class of material when determining the effect of the fatigue.

Use the property `pfcMaterial.PermittedFatigueMaterialTypes` to retrieve a list of the permitted string values for the fatigue material type.

The property `pfcMaterial.FatigueMaterialFinish` sets the type of surface finish for the fatigue material.

Use the property `pfcMaterial.PermittedFatigueMaterialFinishes` to retrieve a list of permitted string values for the fatigue material finish.

The property `pfcMaterial.FailureCriterion` setsthe 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 property `pfcMaterial.PermittedFailureCriteria` to retrieve a list of permitted string values for the material failure criterion.

The property `pfcMaterial.Hardness` sets the hardness for the specified material.

The property `pfcMaterial.HardnessType` sets the hardness type for the specified material.

The property `pfcMaterial.Condition` sets the condition for the specified material .

The property `pfcMaterial.BendTable` and `pfcPart.Material.SetBendTable` return and sets the bend table for the specified material .

The property `pfcMaterial.CrossHatchFile` sets the file containing the crosshatch pattern for the specified material .

The property `pfcMaterial.MaterialModel` sets the type of hyperelastic isotropic material model .

Use the property `pfcMaterial.PermittedMaterialModels` to retrieve a list of the permitted string values for the material model.

The property `pfcPart.Material.GetModelDefByTests` determines whether the hyperelastic isotropic material model has been defined 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 `pfcMaterial` is a child of `pfcParameterOwner`, which provides access to user-defined parameters and properties of materials through the methods in that interface.

Windows and Views

Web.Link provides access to Creo Parametric windows and saved views. This section describes the methods that provide this access.

Windows

This section describes the Web.Link methods that access `Window` objects. The topics are as follows:

- [Getting a Window Object](#)
- [Window Operations](#)

Getting a Window Object

Methods and Property Introduced:

- **pfcBaseSession.CurrentWindow**
- **pfcBaseSession.CreateModelWindow()**
- **pfcModel.Display()**
- **pfcBaseSession.ListWindows()**
- **pfcBaseSession.GetWindow()**
- **pfcBaseSession.OpenFile()**
- **pfcBaseSession.GetModelWindow()**

The property `pfcBaseSession.CurrentWindow` provides access to the current active window in Creo Parametric.

The method `pfcBaseSession.CreateModelWindow()` creates a new window that contains the model that was passed as an argument.

Note

You must call the method `pfcModel.Display()` for the model geometry to be displayed in the window.

Use the method `pfcBaseSession.ListWindows()` to get a list of all the current windows in session.

The method `pfcBaseSession.GetWindow()` gets the handle to a window given its integer identifier.

The method `pfcBaseSession.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 and Properties Introduced:

- **pfcWindow.Height**
- **pfcWindow.Width**
- **pfcWindow.XPos**
- **pfcWindow.YPos**
- **pfcWindow.GraphicsAreaHeight**
- **pfcWindow.GraphicsAreaWidth**
- **pfcWindow.Clear()**
- **pfcWindow.Repaint()**
- **pfcWindow.Refresh()**
- **pfcWindow.Close()**

- **pfcWindow.Activate()**
- **pfcWindow.GetId()**
- **pfcBaseSession.FlushCurrentWindow()**

The properties `pfcWindow.Height`, `pfcWindow.Width`, `pfcWindow.XPos`, and `pfcWindow.YPos` 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 properties `pfcWindow.GraphicsAreaHeight` and `pfcWindow.GraphicsAreaWidth` retrieve the height and width of the 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.Clear()` removes geometry from the window.

Both `pfcWindow.Repaint()` and `pfcWindow.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.Close()` to close the window. If the current window is the original window created when Creo Parametric started, this method clears the window. Otherwise, it removes the window from the screen.

The method `pfcWindow.Activate()` activates a window. This function is available only in the asynchronous mode.

The method `pfcWindow.GetId()` retrieves the ID of the 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.GetURL()**
- **pfcWindow.SetURL()**
- **pfcWindow.GetBrowserSize()**
- **pfcWindow.SetBrowserSize()**

The methods `pfcWindow.GetURL()` and `pfcWindow.SetURL()` enables you to find and change the URL displayed in the embedded browser in the Creo Parametric window.

The methods `pfcWindow.GetBrowserSize()` and `pfcWindow.Window.SetBrowserSize` enables you to find and change the size of the embedded browser in the 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 Web.Link methods that access `pfcView` objects. The topics are as follows:

- [Getting a View Object](#)
- [View Operations](#)

Getting a View Object

Methods Introduced:

- **pfcViewOwner.RetrieveView()**
- **pfcViewOwner.GetView()**
- **pfcViewOwner.ListViews()**
- **pfcViewOwner.GetCurrentView()**

Any solid model inherits from the interface `pfcViewOwner`. This will enable you to use these methods on any solid object.

The method `pfcViewOwner.RetrieveView()` sets the current view to the orientation previously saved with a specified name.

Use the method `pfcViewOwner.GetView()` to get a handle to a named view without making any modifications.

The method `pfcViewOwner.ListViews()` returns a list of all the views previously saved in the model.

The method `pfcView.ViewOwner.GetCurrentView` 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 and Properties Introduced:

- **pfcView.Name**
- **pfcView.IsCurrent**
- **pfcView.Reset()**
- **pfcViewOwner.SaveView()**

To get the name of a view given its identifier, use the property `pfcView.Name`.

The property `pfcView.IsCurrent` determines if the View object represents the current view.

The `pfcView.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 Creo Parametric and accessible from Web.Link and how to transform from one coordinate system to another.

Coordinate Systems

Creo Parametric and Web.Link use the following coordinate systems:

- Solid Coordinate System
- Screen Coordinate System
- Window Coordinate System
- Drawing Coordinate System
- Drawing View Coordinate System
- Assembly Coordinate System
- Datum Coordinate System
- Section Coordinate System

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 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 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 Creo Parametric user.

Solid coordinates are used by Web.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 Creo Parametric window. 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 nominal pixel counts. The bottom, left corner of the default window is at (0, 0) and the top, right corner is at (1000, 864).

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, except it is not affected by zoom and pan. When an object is first displayed in a window, or the option **View**, **Pan/Zoom**, **Reset** is used, the screen and window coordinates are the same.

Window coordinates are needed only if you take account of zoom and pan. For example, you can find out whether a point on the solid is visible in the window or you can 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 Web.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 Web.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 and Properties Introduced:

- **pfcTransform3D.Invert()**

- **pfcTransform3D.TransformPoint()**
- **pfcTransform3D.TransformVector()**
- **pfcTransform3D.Matrix**
- **pfcTransform3D.GetOrigin()**
- **pfcTransform3D.GetXAxis()**
- **pfcTransform3D.GetYAxis()**
- **pfcTransform3D.GetZAxis()**

All coordinate systems are treated in Web.Link as if they were three-dimensional. Therefore, a point in any of the coordinate systems is always represented by the `pfcPoint3D` class:

Vectors store the same data but are represented for clarity by the `pfcVector3D` 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 `IpfcTransform3D` 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:

$$\begin{bmatrix} X' & Y' & Z' & 1 \end{bmatrix} = \begin{bmatrix} X & Y & Z & 1 \end{bmatrix} \begin{bmatrix} \dots & \dots & \dots & 0 \\ \dots & \dots & \dots & 0 \\ \dots & \dots & \dots & 0 \\ X_s & Y_s & Z_s & 1 \end{bmatrix}$$

The utility method `pfcTransform3D.Invert()` inverts a transformation matrix so that it can be used to transform points in the opposite direction.

Web.Link provides two utilities for performing coordinate transformations. The method `pfcTransform3D.TransformPoint()` transforms a three-dimensional point and `pfcTransform3D.TransformVector()` transforms a three-dimensional vector.

The following diagram summarizes the coordinate transformations needed when using Web.Link and specifies the Web.Link methods that provide the transformation matrix.
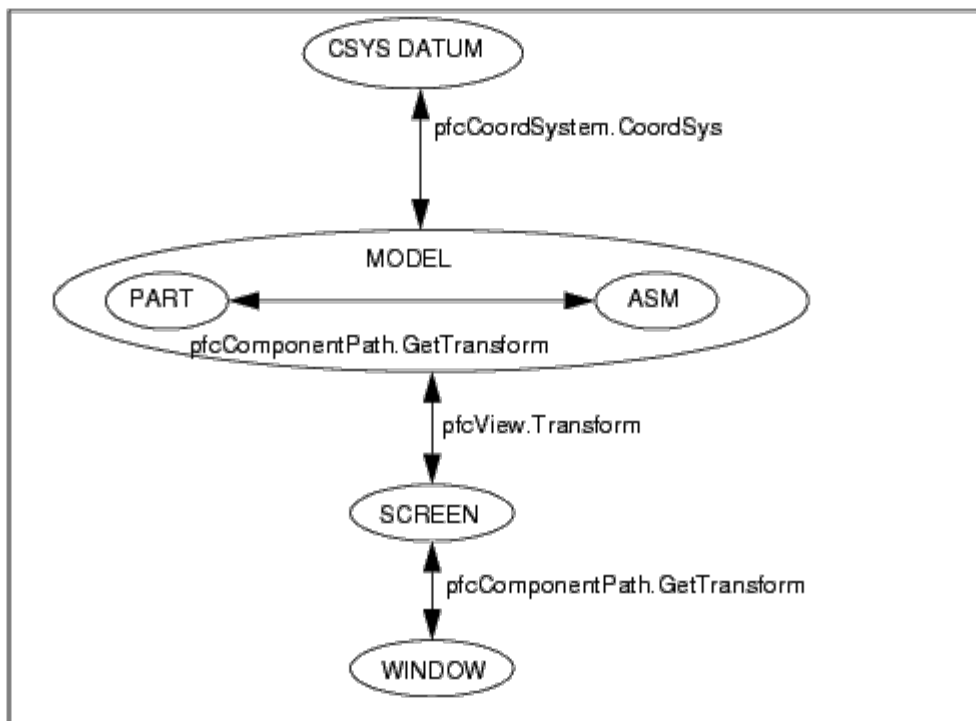
Transforming to Screen Coordinates

Methods and Properties Introduced:

- **pfcView.Transform**
- **pfcView.Rotate()**

The view matrix describes the transformation from solid to screen coordinates. The property `pfcView.Transform` provides the view matrix for the specified 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 `pfcTransform3D.Invert()`.

Transforming to Coordinate System Datum Coordinates

property Introduced:

- **pfcCoordSystem.CoordSys**

The property `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

Properties Introduced

- **pfcWindow.ScreenTransform**
- **pfcScreenTransform.PanX**
- **pfcScreenTransform.PanY**
- **pfcScreenTransform.Zoom**

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:

- **pfcComponentPath.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.

.

ModelItem

This section describes the Web.Link methods that enable you to access and manipulate `ModelItems`.

Solid Geometry Traversal

Solid models are made up of 11 distinct types of `pfcModelItem`, as follows:

- `pfcFeature`
- `pfcSurface`
- `pfcEdge`
- `pfcCurve` (datum curve)
- `pfcAxis` (datum axis)
- `pfcPoint` (datum point)
- `pfcQuilt` (datum quilt)
- `pfcLayer`
- `pfcNote`
- `pfcDimension`
- `pfcRefDimension`

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 and Properties Introduced:

- **pfcModelItemOwner.ListItems()**
- **pfcFeature.ListSubItems()**
- **pfcLayer.ListItems()**
- **pfcModelItemOwner.GetItemById()**
- **pfcModelItemOwner.GetItemByName()**
- **pfcFamColModelItem.RefItem**
- **pfcSelection.SelItem**

All models inherit from the class `pfcModelItemOwner`. The method `pfcModelItemOwner.ListItems()` returns a sequence of `pfcModelItems` contained in the model. You can specify which type of `pfcModelItem` to collect by passing in one of the enumerated `pfcModelItemType` values, or you can collect all `pfcModelItems` 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.ListSubItems()` and `pfcLayer.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 `pfcModelItemOwner.GetItemById()`. This methods enables you to access the model item by identifier.

To access specific model items, call the method `pfcModelItemOwner.GetItemByName()`. This methods enables you to access the model item by name.

The property `pfcFamColModelItem.RefItem` returns the dimension or feature used as a header for a family table.

The property `pfcSelect.Selection.GetSelItem` returns the item selected interactively by the user.

ModelItem Information

Methods and Properties Introduced:

- **pfcModelItem.GetName()**
- **pfcModelItem.SetName()**
- **pfcModelItem.Id**
- **pfcModelItem.Type**

Certain `pfcModelItems` also have a string name that can be changed at any time. The methods `GetName` and `SetName` access this name.

The property `GetId` returns the unique integer identifier for the `pfcModelItem`.

The `GetType` property returns an enumeration object that indicates the model item type of the specified `pfcModelItem`. See the section Solid Geometry Traversal for the list of possible model item types.

Duplicating ModelItems

Methods and Properties Introduced:

- **pfcBaseSession.AllowDuplicateModelItems()**

You can control the creation of ModelItems more than twice for the same 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, although 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 Web.Link, layers are instances of `pfcModelItem`. The following sections describe how to get layer objects and the operations you can perform on them.

Getting Layer Objects

Method Introduced:

- **pfcModel.CreateLayer()**

The method `pfcModel.Model.CreateLayer` returns a new layer with the name you specify.

See the section Getting ModelItem Objects for other methods that can return layer objects.

Layer Operations

Methods and Properties Introduced:

- **pfcLayer.Status**
- **pfcLayer.ListItems()**
- **pfcLayer.AddItem()**
- **pfcLayer.RemoveItem()**
- **pfcLayer.Delete()**

The property `pfcLayer.Status` enables you to access the display status of a layer. The corresponding enumeration class is `pfcDisplayStatus` and the possible values are `Normal`, `Displayed`, `Blank`, or `Hidden`.

Use the methods `pfcLayer.ListItems()`, `pfcLayer.AddItem()`, and `pfcLayer.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 `pfcXToolkitInvalidType`.

The method `pfcLayer.Layer.Delete` removes the layer (but not the items it contains) from the model.

Features

All Creo Parametric solid models are made up of features. This section describes how to program on the feature level using Web.Link.

Access to Features

Methods and Properties Introduced:

- **pfcFeature.ListChildren()**
- **pfcFeature.ListParents()**
- **pfcFeatureGroup.GroupLeader**
- **pfcFeaturePattern.PatternLeader**
- **pfcFeaturePattern.ListMembers()**
- **pfcSolid.ListFailedFeatures()**
- **pfcSolid.ListFeaturesByType()**
- **pfcSolid.GetFeatureById()**

The methods `pfcFeature.ListChildren()` and `pfcFeature.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 property `pfcFeatureGroup.GroupLeader`.

The property `pfcFeaturePattern.PatternLeader` and the method `pfcFeaturePattern.ListMembers()` return features that make up the specified feature pattern. See the section [Feature Groups and Patterns](#) for more information on feature patterns.

The method `pfcSolid.ListFailedFeatures()` returns a sequence that contains all the features that failed regeneration.

The method `pfcSolid.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 `pfcFeatureType` 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 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

Properties Introduced:

- **pfcFeature.FeatType**
- **pfcFeature.Status**
- **pfcFeature.IsVisible**
- **pfcFeature.IsReadonly**
- **pfcFeature.IsEmbedded**
- **pfcFeature.Number**
- **pfcFeature.FeatTypeName**
- **pfcFeature.FeatSubType**
- **pfcRoundFeat.IsAutoRoundMember**

The enumeration classes `pfcFeatureType` and `pfcFeatureStatus` provide information for a specified feature. The following properties specify this information:

- `pfcFeature.FeatType`—Returns the type of a feature.

- `pfcFeature.Status`—Returns whether the feature is suppressed, active, or failed regeneration.

The other properties that gather feature information include the following:

- `pfcFeature.IsVisible`—Identifies whether the specified feature will be visible on the screen.
- `pfcFeature.IsReadonly`—Identifies whether the specified feature can be modified.
- `pfcFeature.GetIsEmbedded`—Specifies whether the specified feature is an embedded datum.
- `pfcFeature.Number`—Returns the feature regeneration number. This method returns `void null` if the feature is suppressed.

The property `pfcFeature.FeatTypeName` returns a string representation of the feature type.

The property `pfcFeature.FeatSubType` returns a string representation of the feature subtype, for example, "Extrude" for a protrusion feature.

The property `pfcRoundFeat.RoundFeat.GetIsAutoRoundMember` determines whether the specified round feature is a member of an Auto Round feature.

Feature Operations

Methods and Properties Introduced:

- **pfcSolid.ExecuteFeatureOps()**
- **pfcFeature.CreateSuppressOp()**
- **pfcSuppressOperation.Clip**
- **pfcSuppressOperation.AllowGroupMembers**
- **pfcSuppressOperation.AllowChildGroupMembers**
- **pfcFeature.CreateDeleteOp()**
- **pfcDeleteOperation.Clip**
- **pfcDeleteOperation.AllowGroupMembers**
- **pfcDeleteOperation.AllowChildGroupMembers**
- **pfcDeleteOperation.KeepEmbeddedDatums**
- **pfcFeature.CreateResumeOp()**
- **pfcResumeOperation.WithParents**
- **pfcFeature.CreateReorderBeforeOp()**
- **pfcReorderBeforeOperation.BeforeFeat**
- **pfcFeature.CreateReorderAfterOp()**
- **pfcReorderAfterOperation.AfterFeat**

The method `pfcSolid.ExecuteFeatureOps()` causes a sequence of feature operations to run in order. Feature operations include suppressing, resuming, reordering, and deleting features. The optional `pfcRegenInstructions` 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 `pfcXToolkitBadContext`. 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 Creo Parametric session. Refer to the [Solid Operations](#) section in the [Solid](#) section 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 class `pfcFeature`. Each created operation must be passed as a member of the `pfcFeatureOperations` object to the method `pfcSolid.Solid.ExecuteFeatureOps`.

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 properties `pfcDeleteOperation.Clip` and `pfcSuppressOperation.Clip` 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 properties `pfcSuppressOperation.AllowGroupMembers` and `pfcDeleteOperation.AllowGroupMembers` 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 properties `pfcSuppressOperation.AllowChildGroupMembers` and `pfcDeleteOperation.AllowChildGroupMembers` 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 property `pfcDeleteOperation.KeepEmbeddedDatums` to modify this option.

* `WithParents`—Specifies whether to resume the parents of the selected feature.

  Use the property `pfcResumeOperation.WithParents` to modify this option.

* `BeforeFeat`—Specifies the feature before which you want to reorder the features.

  Use the property `pfcReorderBeforeOperation.BeforeFeat` to modify this option.

* `AfterFeat`—Specifies the feature after which you want to reorder the features.

  Use the property `pfcFeature.ReorderAfterOperation.SetAfterFeat` to modify this option.

Feature Groups and Patterns

Patterns are treated as features in Creo Parametric. A feature type, `FEATTYPE_PATTERN_HEAD`, is used for the pattern header feature.

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 and Properties Introduced:

* **pfcFeature.Group**
* **pfcFeature.Pattern**
* **pfcSolid.CreateLocalGroup()**
* **pfcFeatureGroup.Pattern**
* **pfcFeatureGroup.GroupLeader**
* **pfcFeaturePattern.PatternLeader**
* **pfcFeaturePattern.ListMembers()**
* **pfcFeaturePattern.Delete()**

The property `pfcFeature.Group` returns a handle to the local group that contains the specified feature.

To get the first feature in the specified group call the property `pfcFeatureGroup.GroupLeader`.

The property `pfcFeaturePattern.PatternLeader` and the method `pfcFeaturePattern.ListMembers()` return features that make up the specified feature pattern.

The properties `pfcFeature.Pattern` and `pfcFeatureGroup.Pattern` return the `FeaturePattern` object that contains the corresponding `Feature` or `FeatureGroup`. Use the method `pfcSolid.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`.

Feature Groups

Feature groups 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.

The results of the header feature are as follows:

- Models that contain groups will get one extra feature in the regeneration list, of type `pfcFeatureType.FEATTYPE_GROUP_HEAD`. This affects the feature numbers of all subsequent features, including those in the group.
- Each group automatically contains the header feature in the list of features returned from `pfcFeatureGroup.ListMembers()`.
- Each group automatically gets the group head feature as the leader. This is returned from `pfcFeatureGroup.GroupLeader`.
- Each group pattern contains a series of groups, and each group in the pattern will be similarly constructed.

User Defined Features

Groups in 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 and Properties Introduced:

- **pfcFeatureGroup.UDFName**
- **pfcFeatureGroup.UDFInstanceName**
- **pfcFeatureGroup.ListUDFDimensions()**
- **pfcUDFDimension.UDFDimensionName**

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 property `pfcFeatureGroup.UDFName` 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 property `pfcFeatureGroup.UDFInstanceName` supplies the instance name.

The method `pfcFeatureGroup.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 property `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.CreateUDFGroup()`.

Creating Features from UDFs

Method Introduced:

- **pfcSolid.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 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 Creo Parametric. PTC recommends that you read about UDF's in the Creo Parametric on-line help, and practice defining and using UDF's in Creo Parametric before you attempt to use this method.

When you create a UDF interactively, Creo Parametric prompts you for the information it needs to fix the properties of the resulting features. When you create a UDF from Web.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`, 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 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 Creo Parametric must know what direction the new feature will point to. When you create such a UDF interactively 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 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 Creo Parametric, Web.Link uses a special class that prepares and sets all the options and passes them to Creo Parametric.

Creating Interactively Defined UDFs

Method Introduced:

- **pfcUDFPromptCreateInstructions.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:

- **pfcUDFCustomCreateInstructions.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 Creo Parametric gives you when you create a UDF interactively. PTC recommends that you experiment with creating the UDF interactively using Creo Parametric before you write the Web.Link code to fill the structure.

Setting the Family Table Instance Name

Property Introduced:

- **pfcUDFCustomCreateInstructions.InstanceName**

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

Property Introduced:

- **pfcUDFCustomCreateInstructions.DependencyType**

The `pfcUDFDependencyType` 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

Properties Introduced:

- **pfcUDFCustomCreateInstructions.ScaleType**
- **pfcUDFCustomCreateInstructions.Scale**

The property *ScaleType* specifies the length units of the UDF in the form of the `pfcUDFScaleType` 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.

The property *Scale* specifies the scale factor. If the *ScaleType* is set to UDFSCALE_CUSTOM, the property `Scale` assigns the user defined scale factor. Otherwise, this attribute is ignored.

Setting the Appearance of the Non UDF Dimensions

Properties Introduced:

- **pfcUDFCustomCreateInstructions.DimDisplayType**

The `pfcUDFDimensionDisplayType` object sets the options in 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 and Properties Introduced:

- **pfcUDFCustomCreateInstructions.VariantValues**
- **pfcUDFVariantDimension.Create()**
- **pfcUDFVariantPatternParam.Create()**

`pfcUDFVariantValues` class represents an array of variable dimensions and pattern parameters.

`pfcUDFVariantDimension.Create()` is a static method creating a `pfcUDFVariantDimension`. 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 Web.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.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 `pfcFeatureGroup.ListUDFDimensions()` and `pfcUDFDimension.UDFDimensionName` to find out the name.

`pfcUDFCreate.pfcUDFCreate.UDFVariantPatternParam_Create` is a static method which creates a `pfcUDFVariantPatternParam`. It accepts the following parameters:

- *name*—The string name that the pattern parameter had when the UDF was originally defined
- *number*—The new value.

After the `pfcUDFVariantValues` object has been compiled, use `pfcUDFCustomCreateInstructions.VariantValues` to add the variable dimensions and parameters to the instructions.

Setting the User Defined References

Methods and Properties Introduced:

- **pfcUDFReference.Create()**
- **pfcUDFReference.IsExternal**
- **pfcUDFReference.ReferenceItem**
- **pfcUDFCustomCreateInstructions.References**

The method `pfcUDFReference.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.CreateUDFGroup()` will not recognize it and prompts the user for the reference in the usual way.

- *ReferenceItem*—Specifies the `pfcSelection` 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 property `pfcUDFReference.IsExternal`.

To set the item to be used for reference, use the property `pfcUDFReference.ReferenceItem`.

After the `UDFReferences` object has been set, use `pfcUDFCreate.UDFCustomCreateInstructions.SetReferences` to add the program-defined references.

Setting the Assembly Intersections

Methods and Properties Introduced:

- **pfcUDFAssemblyIntersection.Create()**
- **pfcUDFAssemblyIntersection.InstanceNames**
- **pfcUDFCustomCreateInstructions.Intersections**

`pfcUDFAssemblyIntersection.Create()` is a static method creating a `pfcUDFReference` object. It accepts the following parameters:

- *ComponentPath*—Is an `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.

`pfcUDFAssemblyIntersection.InstanceNames` sets an array of names for the new instances of parts created to represent the intersection geometry. This property accepts the following parameters:

- *instance names*—is a `com.ptc.cipjava.stringseq` type object representing the array of new instance names.

After the `pfcUDFAssemblyIntersections` object has been set, use `pfcUDFCreate.UDFCustomCreateInstructions.SetIntersections` to add the assembly intersections.

Setting Orientations

Properties Introduced:

- **pfcUDFCustomCreateInstructions.Orientations**

`pfcUDFOrientations` class represents an array of orientations that provide the answers to Creo Parametric prompts that use a flip arrow. Each term is a `pfcUDFOrientation` 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.

The order of orientations should correspond to the order in which Creo Parametric prompts for them when the UDF is created interactively. If you do not provide an orientation that Creo Parametric needs, it uses the default value `NO_FLIP`.

After the `pfcUDFOrientations` object has been set use `pfcUDFCreate.UDFCustomCreateInstructions.SetOrientations` to add the orientations.

Setting Quadrants

Property Introduced:

- **pfcUDFCustomCreateInstructions.Quadrants**

The property `pfcUDFCreate.UDFCustomCreateInstructions.SetQuadrants` sets an array of points, which provide the X, Y, and Z coordinates that correspond to the picks answering the Creo Parametric prompts for the feature positions. The order of quadrants should correspond to the order in which Creo Parametric prompts for them when the UDF is created interactively.

Setting the External References

Property Introduced:

- **pfcUDFCustomCreateInstructions.ExtReferences**

The property `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.js` located at `<creo_weblink_loadpoint>/weblinkexamples/jscript` 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..

Datum Features

This section describes the Web.Link methods and properties 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`data object.

Methods and Properties Introduced:

- **pfcDatumPlaneFeat.Flip**
- **pfcDatumPlaneFeat.Constraints**
- **pfcDatumPlaneConstraint.ConstraintType**
- **pfcDatumPlaneThroughConstraint.Create()**
- **pfcDatumPlaneThroughConstraint.ThroughRef**
- **pfcDatumPlaneNormalConstraint.Create()**
- **pfcDatumPlaneNormalConstraint.NormalRef**
- **pfcDatumPlaneParallelConstraint.Create()**
- **pfcDatumPlaneParallelConstraint.ParallelRef**
- **pfcDatumPlaneTangentConstraint.Create()**
- **pfcDatumPlaneTangentConstraint.TangentRef**
- **pfcDatumPlaneOffsetConstraint.Create()**
- **pfcDatumPlaneOffsetConstraint.OffsetRef**
- **pfcDatumPlaneOffsetConstraint.OffsetValue**
- **pfcDatumPlaneOffsetCoordSysConstraint.Create()**
- **pfcDatumPlaneOffsetCoordSysConstraint.CsysAxis**
- **pfcDatumPlaneAngleConstraint.Create()**
- **pfcDatumPlaneAngleConstraint.AngleRef**
- **pfcDatumPlaneAngleConstraint.AngleValue**
- **pfcDatumPlaneSectionConstraint.Create()**
- **pfcDatumPlaneSectionConstraint.SectionRef**
- **pfcDatumPlaneSectionConstraint.SectionIndex**
- **pfcDatumPlaneDefaultXConstraint.Create()**
- **pfcDatumPlaneDefaultYConstraint.Create()**
- **pfcDatumPlaneDefaultZConstraint.Create()**

The properties of the `pfcDatumPlaneFeat` object are described as follows:

- `Flip`—Specifies whether the datum plane was flipped during creation. Use the property `pfcDatumPlaneFeat.Flip` to determine if the datum plane was flipped during creation.

- `Constraints`—Specifies a collection of constraints (given by the `pfcDatumPlaneConstraint` object). The property `pfcDatumPlaneFeat.Constraints` obtains the collection of constraints defined for the datum plane.

Use the property `pfcDatumPlaneConstraint.ConstraintType` to obtain the type of constraint. The type of constraint is given by the `pfcDatumPlaneConstraintType` enumerated type. The available types are as follows:

- `DTMPLN_THRU`—Specifies the Through constraint. The `pfcDatumPlaneThroughConstraint` object specifies this constraint. Use the method `pfcDatumPlaneFeat.pfcDatumPlaneFeat.DatumPlaneThrough Constraint_Create` to create a new object. Use the property `pfcDatumPlaneFeat.DatumPlaneThroughConstraint. GetThroughRef` to get the reference selection handle for the Through constraint.

- `DTMPLN_NORM`—Specifies the Normal constraint. The `pfcDatumPlaneNormalConstraint` object specifies this constraint. Use the method `pfcDatumPlaneFeat.pfcDatumPlaneFeat.DatumPlaneNormal Constraint_Create` to create a new object. Use the property `pfcDatumPlaneFeat.DatumPlaneNormalConstraint .GetNormalRef` to get the reference selection handle for the Normal constraint.

- `DTMPLN_PRL`—Specifies the Parallel constraint.
  The `pfcDatumPlaneFeatDatumPlaneParallelConstraint` object specifies this constraint. Use the method `pfcDatumPlaneFeat.pfcDatumPlaneFeat`
  `DatumPlaneParallelConstraint_Create` to create a new object. Use the property `pfcDatumPlaneFeat.DatumPlaneParallelConstraint`
  `.GetParallelRef` to get the reference selection handle for the Parallel constraint.

- `DTMPLN_TANG`—Specifies the Tangent constraint. The `pfcDatumPlaneTangentConstraint` specifies this constraint.
  Use the method `pfcDatumPlaneFeat.pfcDatumPlaneFeat.DatumPlaneTangent`
  `Constraint_Create` to create a new object. Use the property `pfcDatumPlaneFeat.DatumPlaneTangentConstraint`
  `.GetTangentRef` to get the reference selection handle for the Tangent constraint.

- `DTMPLN_OFFS`—Specifies the Offset constraint. The `pfcDatumPlaneOffsetConstraint` object specifies this constraint. Use the method `pfcDatumPlaneFeat.pfcDatumPlaneOffsetConstraint`
  `_Create` to create a new object. Use the property `pfcDatumPlaneFeat.DatumPlaneOffsetConstraint`
  `.GetOffsetRef` to get the reference selection handle for the Offset constraint. Use the property `pfcDatumPlaneOffsetConstraint.OffsetValue` to get the offset value.

  An Offset constraint where the offset reference is a coordinate system is given by the `pfcDatumPlaneOffsetCoordSysConstraint` object. Use the method `pfcDatumPlaneFeat.pfcDatumPlaneFeat.`
  `DatumPlaneOffsetCoordSysConstraint_Create` to create a new object. Use the property `pfcDatumPlaneFeat.DatumPlaneOffsetCoordSysConstraint`
  `.GetCsysAxis` to get the reference coordinate axis.

- `DTMPLN_ANG`—Specifies the Angle constraint. The `pfcDatumPlaneAngleConstraint` object specifies this constraint.
  Use the method `pfcDatumPlaneFeat.pfcDatumPlaneAngleConstraint`
  `_Create` to create a new object. Use the property `pfcDatumPlaneFeat.DatumPlaneAngleConstraint`
  `.GetAngleRef` to get the reference selection handle for the Angle constraint. Use the property `pfcDatumPlaneFeat.DatumPlaneAngleConstraint`
  `.GetAngleValue` to get the angle value.

- `DTMPLN_SEC`—Specifies the Section constraint. The `pfc`
  `DatumPlaneSectionConstraint` object specifies this constraint. Use the method `pfcDatumPlaneFeat.pfcDatumPlaneSectionConstraint`
  `_Create` to create a new object. Use the property `pfcDatumPlaneFeat.DatumPlaneSectionConstraint.`
  `GetSectionRef` to get the reference selection for the Section constraint. Use the property `pfcDatumPlaneFeat.DatumPlaneSectionConstraint`
  `.GetSectionIndex` to get the section index.

- `DTMPLN_DEF_x`—Specifies the default RIGHT constraint for the datum plane. The `pfcDatumPlaneDefaultXConstraint` object specifies this constraint. Use the method `pfcDatumPlaneFeat.pfcDatumPlaneFeat`
  `.DatumPlaneDefaultXConstraint_Create` to create a new object.

- `DTMPLN_DEF_y`—Specifies the default TOP constraint for the datum plane. The `pfcDatumPlaneDefaultYConstraint` object specifies this constraint. Use the method `pfcDatumPlaneFeat.pfcDatumPlaneFeat`
  `.DatumPlaneDefaultYConstraint_Create` to create a new object.

- `DTMPLN_DEF_z`—Specifies the default FRONT constraint for the datum plane. The `pfcDatumPlaneDefaultZConstraint` object specifies this constraint Use the method `pfcDatumPlaneFeat.pfcDatumPlaneFeat`
  `.DatumPlaneDefaultZConstraint_Create` to create a new object.

Datum Axis Features

The properties of the Datum Axis feature are defined in the `pfcDatumAxisFeat` data object.

Methods Introduced:

- **pfcDatumAxisFeat.Constraints**
- **pfcDatumAxisConstraint.Create()**
- **pfcDatumAxisConstraint.ConstraintType**

- **pfcDatumAxisConstraint.ConstraintRef**
- **pfcDatumAxisFeat.DimConstraints**
- **pfcDatumAxisDimensionConstraint.Create()**
- **pfcDatumAxisDimensionConstraint.DimOffset**
- **pfcDatumAxisDimensionConstraint.DimRef**

The properties of the `pfcDatumAxisFeat` object are described as follows:

- `Constraints`—Specifies a collection of constraints (given by the `pfcDatumAxisConstraint` object). The property `pfcDatumAxisFeat.Constraints` obtains the collection of constraints applied to the Datum Axis feature.

  Use the method `pfcDatumAxisConstraint.Create()` to create a new `pfcDatumAxisConstraint` object. This object contains the following attributes:

  - `ConstraintType`—Specifies the type of constraint in terms of the `pfcDatumAxisConstraintType` 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 property `pfcDatumAxisConstraint.ConstraintType` to get the constraint type.

  - `ConstraintRef`—Specifies the reference selection for the constraint. Use the property `pfcDatumAxisConstraint.ConstraintRef` to get the reference selection handle.

- `DimConstraints`—Specifies a collection of dimension constraints (given by the `pfcDatumAxisDimensionConstraint` object). The property `pfcDatumAxisFeat.DimConstraints` obtains the collection of dimension constraints applied to the Datum Axis feature.

  Use the method `pfcDatumAxisFeat.pfcDatumAxisFeat.DatumAxisDimensionConstraint_Create` to create a new `pfcDatumAxisFeat.DatumAxisDimensionConstraint` object. This object contains the following attributes:

  - `DimOffset`—Specifies the offset value for the dimension constraint. Use the property `pfcDatumAxisFeat.DatumAxisDimensionConstraint.GetDimOffset` to get the offset value.

  - `DimRef`—Specifies the reference selection for the dimension constraint. Use the property `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.` data object.

Methods Introduced:

- **pfcDatumPointFeat.FeatName**
- **pfcDatumPointFeat.GetPoints()**
- **pfcGeneralDatumPoint.Name**
- **pfcDatumPointPlacementConstraint.Create()**
- **pfcGeneralDatumPoint.PlaceConstraints**
- **pfcDatumPointDimensionConstraint.Create()**
- **pfcGeneralDatumPoint.DimConstraints**
- **pfcDatumPointConstraint.ConstraintRef**
- **pfcDatumPointConstraint.ConstraintType**
- **pfcDatumPointConstraint.Value**

The properties of the `pfcDatumPointFeat` object are described as follows:

- `FeatName`—Specifies the name of the General Datum Point feature. Use the property `pfcDatumPointFeat.FeatName` to get the name.

- `GeneralDatumPoints`—Specifies a collection of general datum points (given by the `pfcGeneralDatumPoint` object). Use the method `pfcDatumPointFeat.GetPoints()` to obtain the collection of general datum points. The `pfcGeneralDatumPoint` object consists of the following attributes:

  - `Name`—Specifies the name of the general datum point. Use the property `pfcGeneralDatumPoint.Name` to get the name.

  - `PlaceConstraints`—Specifies a collection of placement constraints (given by the `pfcDatumPointPlacementConstraint` object). Use the method `pfcDatumPointFeat.pfcDatumPointFeat.DatumPointPlacement` `Constraint_Create` to create a new object. Use the property `pfcDatumPointFeat.GeneralDatumPoint.` `GetPlaceConstraints` to obtain the collection of placement constraints.

  - `DimConstraints`—Specifies a collection of dimension constraints (given by the `pfcDatumPointDimensionConstraint` object) Use the method `pfcDatumPointFeat.pfcDatumPointFeat.` `DatumPointDimensionConstraint_Create` to create a new object. Use the property `pfcDatumPointFeat.GeneralDatumPoint.` `GetDimConstraints` to obtain the collection of dimension constraints.

The constraints for a datum point are given by the `pfcDatumPoint` `Constraint` object. This object contains the following attributes:

- `ConstraintRef`—Specifies the reference selection for the datum point constraint. Use the property `pfcDatumPointConstraint.ConstraintRef` to get the reference selection handle.

- `ConstraintType`—Specifies the type of datum point constraint. in terms of the `pfcDatumPointFeat.DatumPointConstraintType` enumerated type. Use the property `pfcDatumPointConstraint.ConstraintType` to get the constraint type.

- `Value`—Specifies the constraint reference value with respect to the datum point. Use the property `pfcDatumPointFeat.DatumPointConstraint.GetValue` to get the value of the constraint reference with respect to the datum point.

The `pfcDatumPointPlacementConstraint` and `pfcDatumPointDimensionConstraint` objects inherit from the `pfcDatumPointConstraint` object. Use the methods of the `pfcDatumPointFeat.` object for the inherited objects.

Datum Coordinate System Features

The properties of the Datum Coordinate System feature are defined in the `pfcCoordSysFeat` object.

Methods and Properties Introduced:

- **pfcCoordSysFeat.OriginConstraints**
- **pfcDatumCsysOriginConstraint.Create()**
- **pfcDatumCsysOriginConstraint.OriginRef**
- **pfcCoordSysFeat.DimensionConstraints**
- **pfcDatumCsysDimensionConstraint.Create()**
- **pfcDatumCsysDimensionConstraint.DimRef**
- **pfcDatumCsysDimensionConstraint.DimValue**
- **pfcDatumCsysDimensionConstraint.DimConstraintType**
- **pfcCoordSysFeat.OrientationConstraints**
- **pfcDatumCsysOrientMoveConstraint.Create()**
- **pfcDatumCsysOrientMoveConstraint.OrientMoveConstraintType**
- **pfcDatumCsysOrientMoveConstraint.OrientMoveValue**
- **pfcCoordSysFeat.IsNormalToScreen**
- **pfcCoordSysFeat.OffsetType**
- **pfcCoordSysFeat.OnSurfaceType**
- **pfcCoordSysFeat.OrientByMethod**

The properties of the `pfcCoordSysFeat` object are described as follows:

- `OriginConstraints`—Specifies a collection of origin constraints (given by the `pfcDatumCsysOriginConstraint` object). Use the property `pfcCoordSysFeat.OriginConstraints` to obtain the collection of origin constraints for

the coordinate system. Use the method `pfcCoordSysFeat.pfcDatumCsysOriginConstraint` `_Create` to create a new `pfcDatumCsysOrigin` `Constraint` object. This object contains the following attribute:

- ○ `OriginRef`—Specifies the selection reference for the origin. Use the property `pfcDatumCsysOriginConstraint.OriginRef` to get the selection reference handle.

- `DimensionConstraints`—Specifies a collection of dimension constraints (given by the `pfcDatumCsysDimensionConstraint` object). Use the property `pfcCoordSysFeat.DimensionConstraints` to obtain the collection of dimension constraints for the coordinate system. Use the method `pfcCoordSysFeat.pfcCoordSysFeat.DatumCsysDimension` `Constraint_Create` to create a new `pfcDatumCsysDimensionConstraint` object. This object contains the following attributes:

  - ○ `DimRef`—Specifies the reference selection for the dimension constraint. Use the property `pfcDatumCsysDimensionConstraint.DimRef` to get the reference selection handle.

  - ○ `DimValue`—Specifies the value of the reference. Use the property `pfcDatumCsysDimensionConstraint.DimValue` to get the value.

  - ○ `DimConstraintType`—Specifies the type of dimension constraint in terms of the `pfcDatumCsysDimConstraintType` enumerated type. Use the property `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 `pfcDatumCsysOrientMoveConstraint` object). Use the property `pfcCoordSysFeat.OrientationConstraints` to obtain the collection of orientation constraints for the coordinate system. Use the method `pfcCoordSysFeat.pfcCoordSysFeat.DatumCsysOrient` `MoveConstraint_Create` to create a new `pfcDatumCsysOrient` `MoveConstraints` object. This object contains the following attributes:

  - ○ `OrientMoveConstraintType`—Specifies the type of orientation for the constraint. The orientation type is given by the `pfcDatumCsysOrientMoveConstraintType` enumerated type. Use the property `pfcCoordSysFeat.DatumCsysOrientMoveConstraint.` `GetOrientMoveConstraintType` to get the orientation type.

  - ○ `OrientMoveValue`—Specifies the reference value for the constraint. Use the property `pfcCoordSysFeat.DatumCsysOrientMoveConstraint.` `GetOrientMoveValue` to get the reference value.

- `IsNormalToScreen`—Specifies if the coordinate system is normal to screen. Use the property `pfcCoordSysFeat.IsNormalToScreen` to determine if the coordinate system is normal to screen.

- `OffsetType`—Specifies the offset type of the coordinate system in terms of the `pfcDatumCsysOffsetType` enumerated type. Use the property `pfcCoordSysFeat.OffsetType` 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 `pfcDatumCsysOffsetType` enumerated type. Use the property `pfcCoordSysFeat.OnSurfaceType` 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 pfcDatumCsysOrientByMethod enumerated type. Use the property 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.

Geometry Evaluation

This section describes geometry representation and discusses how to evaluate geometry using Web.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:

- **pfcModelItemOwner.ListItems()**
- **pfcSurface.ListContours()**
- **pfcContour.ListElements()**

To traverse the geometry, follow these steps:

1. Starting at the top-level model, use `pfcModelItemOwner.ListItems()` with an argument of `ModelItemType.ITEM_SURFACE`.
2. Use `pfcSurface.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 Web.Link. You can get edges through geometry traversal or get a list of edges using the methods presented in the section ModelItem.

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.
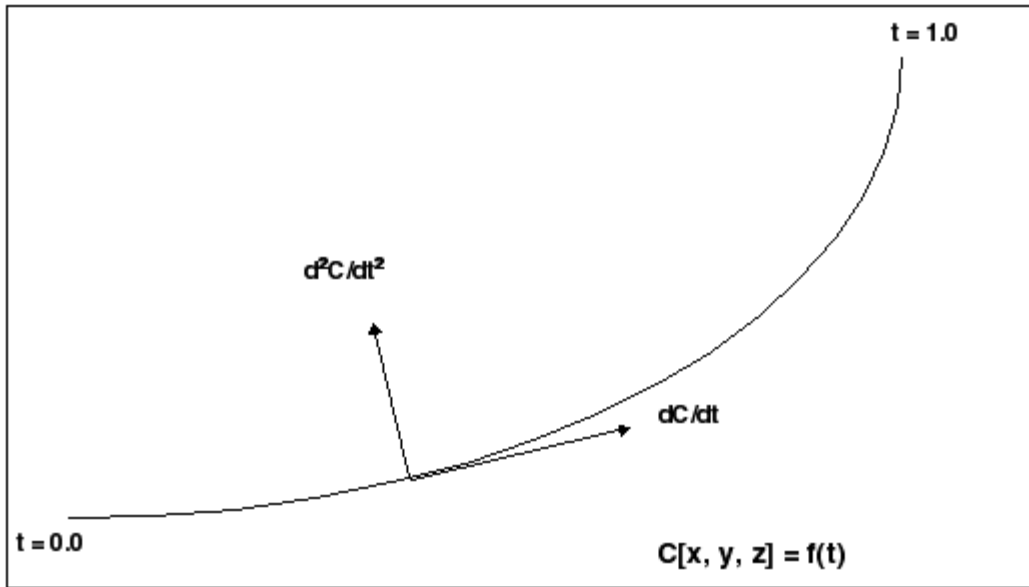
Curve and Edge Types

Solid edges and datum curves can be any of the following types:

- LINE—A straight line represented by the class `pfcLine`.
- ARC—A circular curve represented by the class `pfcArc`.
- SPLINE—A nonuniform cubic spline, represented by the class `pfcSpline`.
- B-SPLINE—A nonuniform rational B-spline curve or edge, represented by the class `pfcBSpline`.
- COMPOSITE CURVE—A combination of two or more curves, represented by the class `pfcCompositeCurve`. This is used for datum curves only.

See the section [Geometry Representations](#) for the parameterization of each curve type. To determine what type of curve a `pfcEdge` or `pfcCurve` object represents, use the Java `instanceof` operator.

Because each curve class inherits from `pfcGeomCurve`, you can use all the evaluation methods in `pfcGeomCurve` on any edge or curve.

The following curve types are not used in solid geometry and are reserved for future expansion:

- CIRCLE (`pfcCircle`)
- ELLIPSE (`pfcEllipse`)
- POLYGON (`pfcPolygon`)
- ARROW (`pfcArrow`)
- TEXT (`pfcText`)

Evaluation of Curves and Edges

Methods Introduced:

- **pfcGeomCurve.Eval3DData()**
- **pfcGeomCurve.EvalFromLength()**
- **pfcGeomCurve.EvalParameter()**
- **pfcGeomCurve.EvalLength()**
- **pfcGeomCurve.EvalLengthBetween()**

The methods in `pfcGeomCurve` provide information about any curve or edge.

The method `pfcGeomCurve.Eval3DData()` returns a `pfcCurveXYZData` object with information on the point represented by the input parameter `t`. The method `pfcGeomCurve.EvalFromLength()` returns a similar object with information on the point that is a specified distance from the starting point.

The method `pfcGeomCurve.EvalParameter()` returns the `t` parameter that represents the input `pfcPoint3D` object.

Both `pfcGeomCurve.EvalLength()` and `pfcGeometry.GeomCurve.EvalLengthBetween` return numerical values for the length of the curve or edge.

Solid Edge Geometry

Methods and Properties Introduced:

- **pfcEdge.Surface1**
- **pfcEdge.Surface2**
- **pfcEdge.Edge1**
- **pfcEdge.Edge2**
- **pfcEdge.EvalUV()**
- **pfcEdge.GetDirection()**

Note

The methods in the interface `pfcEdge` provide information only for solid or surface edges.

The properties `pfcEdge.Surface1` and `pfcEdge.Surface2` return the surfaces bounded by this edge. The properties `pfcEdge.Edge1` and `pfcEdge.Edge2` return the next edges in the two contours that contain this edge.

The method `pfcEdge.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:

- **pfcGeomCurve.GetCurveDescriptor()**
- **pfcGeomCurve.GetNURBSRepresentation()**

Note

To get geometric information for an edge, access the `pfcCurveDescriptor` object for one edge using `pfcGeomCurve.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 and Properties Introduced:

- **pfcSurface.ListContours()**
- **pfcContour.InternalTraversal**
- **pfcContour.FindContainingContour()**
- **pfcContour.EvalArea()**
- **pfcContour.EvalOutline()**
- **pfcContour.VerifyUV()**

Contours are a series of edges that completely bound a surface. A contour is not a `pfcModelItem`. You cannot get contours using the methods that get different types of `pfcModelItem`. Use the method `pfcSurface.ListContours()` to get contours from their containing surfaces.

The property `pfcContour.InternalTraversal` returns a `pfcContourTraversal` enumerated type that identifies whether a given contour is on the outside or inside of a containing surface.

Use the method `pfcContour.FindContainingContour()` to find the contour that entirely encloses the specified contour.

The method `pfcContour.EvalArea()` provides the area enclosed by the contour.

The method `pfcContour.EvalOutline()` returns the points that make up the bounding rectangle of the contour.

Use the method `pfcGeometry.Contour.VerifyUV` to determine whether the given `pfcUVParams` argument lies inside the contour, on the boundary, or outside the contour.

Surfaces

Using Web.Link you access datum and solid surfaces in the same way.

UV Parameterization

A surface in 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.
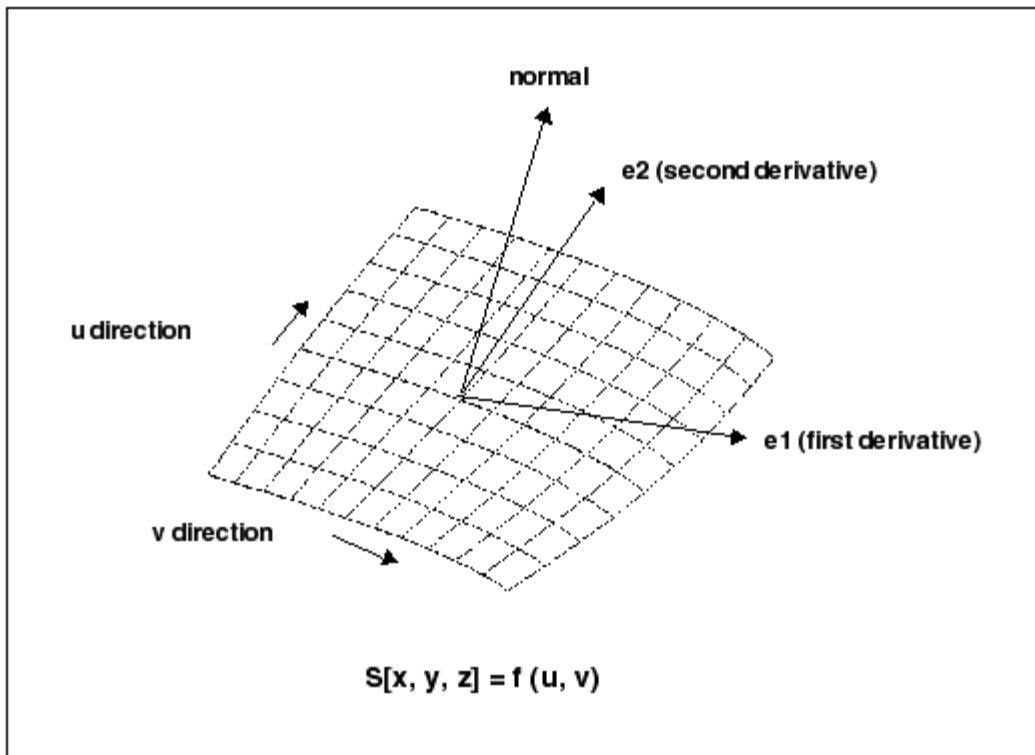
Surface Types

Surfaces within Creo Parametric can be any of the following types:

- PLANE—A planar surface represented by the class `pfcPlane`.
- CYLINDER—A cylindrical surface represented by the class `IpfcCylinder`.
- CONE—A conic surface region represented by the class `pfcCone`.
- TORUS—A toroidal surface region represented by the class `pfcTorus`.
- REVOLVED SURFACE—Generated by revolving a curve about an axis. This is represented by the class `pfcRevSurface`.
- RULED SURFACE—Generated by interpolating linearly between two curve entities. This is represented by the class `pfcRuledSurface`.

- TABULATED CYLINDER—Generated by extruding a curve linearly. This is represented by the class `pfcTabulatedCylinder`.
- QUILT—A combination of two or more surfaces. This is represented by the class `pfcQuilt`.

   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 `pfcCoonsPatch`.
- 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 `pfcFilletedSurface`.
- 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 `pfcSplineSurface`.
- 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 `pfcNURBSSurface`.
- 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 `pfcCylindricalSplineSurface`.

To determine which type of surface a `pfcSurface` object represents, access the surface type using `pfcGeometry.Geometry.GetSurfaceType`.

Surface Information

Methods Introduced:

- **pfcSurface.GetSurfaceType()**
- **pfcSurface.GetXYZExtents()**
- **pfcSurface.GetUVExtents()**
- **pfcSurface.GetOrientation()**

Evaluation of Surfaces

Surface methods allow you to use multiple surface information to calculate, evaluate, determine, and examine surface functions and problems.

Methods and Properties Introduced:

- **pfcSurface.OwnerQuilt**
- **pfcSurface.EvalClosestPoint()**
- **pfcSurface.EvalClosestPointOnSurface()**
- **pfcSurface.Eval3DData()**
- **pfcSurface.EvalParameters()**
- **pfcSurface.EvalArea()**
- **pfcSurface.EvalDiameter()**
- **pfcSurface.EvalPrincipalCurv()**
- **pfcSurface.VerifyUV()**
- **pfcSurface.EvalMaximum()**
- **pfcSurface.EvalMinimum()**
- **pfcSurface.ListSameSurfaces()**

The property `pfcSurface.OwnerQuilt` returns the `Quilt` object that contains the datum surface.

The method `pfcSurface.EvalClosestPoint()` projects a three-dimensional point onto the surface. Use the method `pfcSurface.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 `pfcSurface.Eval3DData()` returns a `pfcSurfXYZData` object that contains information about the

surface at the specified *u* and *v* parameters. The method `pfcSurface.EvalParameters()` returns the *u* and *v* parameters that correspond to the specified three-dimensional point.

The method `pfcSurface.EvalArea()` returns the area of the surface, whereas `pfcSurface.EvalDiameter()` returns the diameter of the surface. If the diameter varies the optional `pfcUVParams` argument identifies where the diameter should be evaluated.

The method `pfcSurface.EvalPrincipalCurv()` returns a `pfcCurvatureData` object with information regarding the curvature of the surface at the specified *u* and *v* parameters.

Use the method `pfcSurface.VerifyUV()` to determine whether the `pfcUVParams` are actually within the boundary of the surface.

The methods `pfcSurface.EvalMaximum()` and `pfcSurface.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:

- **pfcSurface.GetSurfaceDescriptor()**
- **pfcSurface.GetNURBSRepresentation()**

The method `pfcSurface.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 `pfcModelItems` to get one of these geometry objects. Refer to the section [ModelItem](#) for additional information.

Evaluation of ModelItems

Properties Introduced:

- **pfcAxis.Surf**
- **pfcCoordSystem.CoordSys**
- **pfcPoint.Point**

The property `pfcAxis.Surf` returns the revolved surface that uses the axis.

The property `pfcCoordSystem.CoordSys` returns the `pfcTransform3D` object (which includes the origin and x-, y-, and z- axes) that defines the coordinate system.

The property `pfcGeometry.Point.GetPoint` returns the xyz coordinates of the datum point.

Interference

Creo Parametric assemblies can contain interferences between components when constraint by certain rules defined by the user. The `pfcInterference` moduleallows 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 and Properties Introduced:

- **MpfcInterference.CreateGlobalEvaluator()**
- **pfcGlobalEvaluator.ComputeGlobalInterference()**
- **pfcGlobalEvaluator.Assem**
- **pfcGlobalEvaluator.Assem**

- **pfcGlobalInterference.Volume**
- **pfcGlobalInterference.SelParts**

To compute all the interferences within an Assembly one has to call `MpfcInterference.CreateGlobalEvaluator()` with a `pfcAssembly` object as an argument. This call returns a `pfcGlobalEvaluator` object.

The property `pfcGlobalEvaluator.Assem` accesses the assembly to be evaluated.

The method `pfcGlobalEvaluator.ComputeGlobalInterference()` determines the set of all the interferences within the assembly.

This method will return a sequence of `pfcGlobalInterference` 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 `pfcGlobalInterference.Volume` respectively.

Analyzing Interference Information

Methods and Properties Introduced:

- **pfcSelectionPair.Create()**
- **MpfcInterference.CreateSelectionEvaluator()**
- **pfcSelectionEvaluator.Selections**
- **pfcSelectionEvaluator.ComputeInterference()**
- **pfcSelectionEvaluator.ComputeClearance()**
- **pfcSelectionEvaluator.ComputeNearestCriticalDistance()**

The method `pfcSelectionPair.Create()` creates a `pfcSelectionPair` object using two `pfcSelection` objects as arguments.

A return from this method will serve as an argument to `MpfcInterference.CreateSelectionEvaluator()`, which will provide a way to determine the interference data between the two selections.

`pfcSelectionEvaluator.Selections` will extract and set the object to be evaluated.

`pfcSelectionEvaluator.ComputeInterference()` determines the interfering information about the provided selections. This method will return the `pfcInterferenceVolume` object or null if the selections do no interfere.

`pfcSelectionEvaluator.ComputeClearance()` computes the clearance data for the two selection. This method returns a `pfcClearanceData` 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 `pfcCriticalDistanceData` object, which is used to determine and set critical points, surface parameters, and critical distance between points.

Analyzing Interference Volume

Methods and Properties Introduced:

- **pfcInterferenceVolume.ComputeVolume()**
- **pfcInterferenceVolume.Highlight()**
- **pfcInterferenceVolume.Boundaries**

The method `pfcInterferenceVolume.ComputeVolume()` will calculate a value for interfering volume.

The method `pfcInterferenceVolume.Highlight()` will highlight the interfering volume with the color provided in the argument to the function.

The property `pfcInterference.InterferenceVolume.GetBoundaries` will return a set of boundary surface descriptors for the interference volume.

Example Code

The sample code in the file `pfcInterferenceExamples.js` located at `<creo_weblink_loadpoint>/weblinkexamples/jscript` 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.

This method allows a user to evaluate the assembly for a presence of any interferences. Upon finding one, this method will highlight the interfering surfaces, compute and highlight the interference volume.

Dimensions and Parameters

This section describes the Web.Link methods and classes that affect dimensions and parameters.

Overview

Dimensions and parameters in Creo Parametric have similar characteristics but also have significant differences. In Web.Link, the similarities between dimensions and parameters are contained in the `pfcBaseParameter` class. This class 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 `pfcDimension` inherits from the class `pfcModelItem`, and can be assigned tolerances, whereas parameters are not `pfcModelItems` and cannot have tolerances.

The ParamValue Object

Both parameters and dimension objects contain an object of type `pfcParamValue`. 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 `pfcParamValue` object there are different methods used to access each value type and some methods will not be applicable for some `pfcParamValue` objects. If you try to use an incorrect method an exception will be thrown.

Accessing a ParamValue Object

Methods and Property Introduced:

- **MpfcModelItem.CreateIntParamValue()**
- **MpfcModelItem.CreateDoubleParamValue()**
- **MpfcModelItem.CreateStringParamValue()**
- **MpfcModelItem.CreateBoolParamValue()**
- **MpfcModelItem.CreateNoteParamValue()**
- **pfcBaseParameter.Value**

The `pfcModelItem` utility class contains methods for creating each type of `pfcParamValue` object. Once you have established the value type in the object, you can change it. The property `pfcModelItem.BaseParameter.GetValue` returns the `pfcParamValue` associated with a particular parameter or dimension.

A `NotepfcParamValue` 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

Properties Introduced:

- **pfcParamValue.discr**
- **pfcParamValue.IntValue**
- **pfcParamValue.DoubleValue**
- **pfcParamValue.StringValue**
- **pfcParamValue.BoolValue**
- **pfcParamValue.NoteId**

The property `pfcParamValue.discr` returns a enumeration object that identifies the type of value contained in the `pfcParamValue` object. Use this information with the specified properties to access the value. If you use an incorrect property an exception of type `pfcXBadGetParamValue` will be thrown.

Parameter Objects

The following sections describe the Web.Link methods that access parameters. The topics are as follows:

- [Creating and Accessing Parameters](#)
- [Parameter Selection Options](#)

Creating and Accessing Parameters

Methods and Property Introduced:

- **pfcParameterOwner.CreateParam()**
- **pfcParameterOwner.CreateParamWithUnits()**
- **pfcParameterOwner.GetParam()**
- **pfcParameterOwner.ListParams()**
- **pfcParameterOwner.SelectParam()**
- **pfcParameterOwner.SelectParameters()**
- **pfcFamColParam.RefParam**

In Web.Link, models, features, surfaces, and edges inherit from the `pfcParameterOwner` class, because each of the objects can be assigned parameters in Creo Parametric.

The method `pfcParameterOwner.GetParam()` gets a parameter given its name.

The method `pfcParameterOwner.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 `pfcParameterOwner.CreateParam()`.

To create a new parameter with a name, a specific value, and units, call the method `pfcParameterOwner.CreateParamWithUnits()`.

The method `pfcParameterOwner.SelectParam()` allows you to select a parameter from the Creo Parametric user interface. The top model from which the parameters are selected must be displayed in the current window.

The method `pfcParameterOwner.SelectParameters()` allows you to interactively select parameters from the Creo Parametric **Parameter** dialog box based on the parameter selection options specified by the `pfcParameterSelectionOptions` object. The top model from which the parameters are selected must be displayed in the current window. Refer to the section [Parameter Selection Options](#) for more information.

The property `pfcFamily.FamColParam.GetRefParam` returns the reference parameter from the parameter column in a family table.

Parameter Selection Options

Parameter selection options in Web.Link are represented by the `pfcParameterSelectionOptions` class.

Methods and Properties Introduced:

- **pfcParameterSelectionOptions.Create()**
- **pfcParameterSelectionOptions.AllowContextSelection**
- **pfcParameterSelectionOptions.Contexts**
- **pfcParameterSelectionOptions.AllowMultipleSelections**
- **pfcParameterSelectionOptions.SelectButtonLabel**

The method `pfcParameterSelectionOptions.Create()` creates a new instance of the `pfcParameterSelectionOptions` object that is used by the method `pfcParameterOwner.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 property `pfcModelItem.ParameteSelectionOptions.SetAllow ContextSelection` to modify the value of this attribute.

- `Contexts`—The permitted parameter selection contexts in the form of the `pfcParameterSelectionContexts` object. Use the property `pfcParameterSelectionOptions.Contexts` 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 property `pfcModelItem.ParameterSelectionOptions.SetAllow MultipleSelections` to modify this attribute.

- `SelectButtonLabel`—The visible label for the select button in the dialog box.

  Use the property `pfcModelItem.ParameterSelectionOptions.SetSelectButtonLabel` to set the label. If not set, the default label in the language of the active Creo Parametric session is displayed.

Parameter Information

Methods and Properties Introduced:

- **pfcBaseParameter.Value**
- **pfcParameter.GetScaledValue()**
- **pfcParameter.SetScaledValue()**
- **pfcParameter.Units**
- **pfcBaseParameter.IsDesignated**
- **pfcBaseParameter.IsModified**
- **pfcBaseParameter.ResetFromBackup()**
- **pfcParameter.Description**
- **pfcParameter.GetRestriction()**
- **pfcParameter.GetDriverType()**
- **pfcParameter.Reorder()**
- **pfcParameter.Delete()**
- **pfcNamedModelItem.Name**

Parameters inherit methods from the `pfcBaseParameter`, `pfcParameter` and `pfcNamedModelItem` classes.

The property `pfcBaseParameter.Value` returns the value of the parameter or dimension.

The method `pfcParameter.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 `pfcParameter.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 `pfcParameter.Units` returns the units assigned to the parameter.

You can access the designation status of the parameter using the property `pfcBaseParameter.IsDesignated` .

The property `pfcBaseParameter.IsModified` and the method `pfcBaseParameter.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 property `pfcParameter.Description` returns the parameter description, or null, if no description is assigned.

The property `pfcModelItem.Parameter.SetDescription` assigns the parameter description.

The property `pfcParameter.GetRestriction()` identifies if the parameter's value is restricted to a certain range or enumeration. It returns the `pfcParameterRestriction` object. Refer to the section [Parameter Restrictions](#) for more information.

The property `pfcParameter.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 `pfcBaseParameter.IsRelationDriven` for a parameter object type.

The method `pfcParameter.Reorder()` reorders the given parameter to come immediately after the indicated parameter in the **Parameter** dialog box and information files generated by Creo Parametric.

The method `pfcParameter.Delete()` permanently removes a specified parameter.

The property `pfcModelItem.NamedModelItem.GetName` accesses the name of the specified parameter.

Parameter Restrictions

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 Web.Link, but not modify the permitted values or range of values. Parameter restrictions in Web.Link are represented by the class `pfcParameterRestriction`.

Method Introduced:

* **pfcParameterRestriction.Type**

The method `pfcModelItem.ParameterRestriction.GetType` returns the `pfcRestrictionType` 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 class `pfcParameterEnumeration`. It is a child of the `pfcParameterRestriction` class.

Property Introduced:

* **pfcParameterEnumeration.PermittedValues**

The property `pfcModelItem.ParameterEnumeration.GetPermittedValues` returns a list of permitted parameter values allowed by this restriction in the form of a sequence of the `pfcParamValue` objects.

Range Restriction

The `PARAMSELECT_RANGE` type of parameter restriction is represented by the interface `pfcParameterRange`. It is a child of the `pfcParameterRestriction` interface.

Properties Introduced:

* **pfcParameterRange.Maximum**
* **pfcParameterRange.Minimum**
* **pfcParameterLimit.Type**
* **pfcParameterLimit.Value**

The property `pfcParameterRange.Maximum` returns the maximum value limit for the parameter in the form of the `pfcParameterLimit` object.

The property `pfcParameterRange.Minimum` returns the minimum value limit for the parameter in the form of the `pfcParameterLimit` object.

The property `pfcParameterLimit.Type` returns the `pfcParameterLimitType` 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 property `pfcModelItem.ParameterLimit.GetValue` returns the boundary value of the parameter limit in the form of the `pfcParamValue` object.

Example Code: Updating Model Parameters

The sample code in the file `pfcParameterExamples.js` located at `<creo_weblink_loadpoint>/weblinkexamples/jscript` contains a single static utility method. This method creates or updates model parameters based on the name-value pairs in the URL page. A utility method parses the String returned via the URL into int, double, or boolean values if possible.

Dimension Objects

Dimension objects include standard 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 Creo Parametric model items. See the section <u>Getting ModelItem Objects</u> for methods that can return `pfcDimension` and `pfcRefDimension` objects.

Dimension Information

Methods and Properties Introduced:

* **pfcBaseParameter.Value**
* **pfcBaseDimension.DimValue**
* **pfcBaseParameter.IsDesignated**
* **pfcBaseParameter.IsModified**
* **pfcBaseParameter.ResetFromBackup()**
* **pfcBaseParameter.IsRelationDriven**
* **pfcBaseDimension.DimType**
* **pfcBaseDimension.Symbol**
* **pfcBaseDimension.Texts**

All the `pfcBaseParameter` methods are accessible to Dimensions as well as Parameters. See the section <u>Parameter Objects</u> for brief descriptions.

Note

You cannot set the value or designation status of reference dimension objects.

The property `pfcBaseDimension.DimValue` accesses the dimension value as a double. This property provides a shortcut for accessing the dimensions' values without using a `ParamValue` object.

The `pfcBaseParameter.IsRelationDriven` property identifies whether the part or assembly relations control a

dimension.

The property `pfcBaseDimension.DimType` returns an enumeration object that identifies whether a dimension is linear, radial, angular, or diametrical.

The property `pfcBaseDimension.Symbol` returns the dimension or reference dimension symbol (that is, "d#" or "rd#").

The property `pfcDimension.BaseDimension.GetTexts` and `pfcDimension.BaseDimension.SetTexts` methods allows access to the text strings that precede or follow the dimension value.

Dimension Tolerances

Methods and Properties Introduced:

- **pfcDimension.Tolerance**
- **pfcDimTolPlusMinus.Create()**
- **pfcDimTolSymmetric.Create()**
- **pfcDimTolLimits.Create()**
- **pfcDimTolSymSuperscript.Create()**
- **pfcDimTolISODIN.Create()**

Only true dimension objects can have geometric tolerances.

The property `pfcDimension.Tolerance` enables you to access the dimension tolerance. The object types for the dimension tolerance are:

- `pfcDimTolLimits`—Displays dimension tolerances as upper and lower limits.

  Note

  This format is not available when only the tolerance value for a dimension is displayed.

- `pfcDimTolPlusMinus`—Displays dimensions as nominal with plus-minus tolerances. The positive and negative values are independent.

- `pfcDimTolSymmetric`—Displays dimensions as nominal with a single value for both the positive and the negative tolerance.

- `pfcDimTolSymSuperscript`—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.

- `pfcDimTolISODIN`—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 Creo Parametric.

To determine whether a given tolerance is plus/minus, symmetric, limits, or superscript use TBD.

Example Code: Setting Tolerances to a Specified Range

The sample code in the file `pfcDimensionExamples.js` located at `<creo_weblink_loadpoint>/weblinkexamples/jscript` 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.

Relations

This section describes how to access relations on all models and model items in Creo Parametric using the methods provided in Web.Link.

Accessing Relations

In Web.Link, the set of relations on any model or model item is represented by the `pfcRelationOwner` class. Models, features, surfaces, and edges inherit from this interface, because each object can be assigned relations in Creo Parametric.

Methods and Properties Introduced:

- **pfcRelationOwner.RegenerateRelations()**
- **pfcRelationOwner.DeleteRelations()**
- **pfcRelationOwner.Relations**
- **pfcRelationOwner.EvaluateExpression()**

The method `pfcRelationOwner.RegenerateRelations()` regenerates the relations assigned to the owner item. It also determines whether the specified relation set is valid.

The method `pfcRelationOwner.DeleteRelations()` deletes all the relations assigned to the owner item.

The property `pfcRelationOwner.Relations` returns the list of initial relations assigned to the owner item as a sequence of strings.

The method `pfcModelItem.RelationOwner.EvaluateExpression` evaluates the given relations-based expression, and returns the resulting value in the form of the `pfcParamValue` object. Refer to the section The ParamValue Object in the section Dimensions and Parameters for more information on this object.

Example 1: Adding Relations between Parameters in a Solid Model

The sample code in the file `pfcRelationExamples.js` located at `<creo_weblink_loadpoint>/weblinkexamples/jscript` demonstrates how to add relations between parameters in a solid model.

Accessing Post Regeneration Relations

Method and Property Introduced:

- **pfcModel.PostRegenerationRelations**
- **pfcModel.RegeneratePostRegenerationRelations()**
- **pfcModel.DeletePostRegenerationRelations()**

The property `pfcModel.PostRegenerationRelations` 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 `pfcRelationOwner.RegenerateRelations()` *and* `pfcRelationOwner.DeleteRelations()`.

You can regenerate the relation sets post-regeneration in a model using the method `pfcModel.RegeneratePostRegenerationRelations()`.

To delete all the post-regeneration relations in the specified model, call the method `pfcModel.Model.DeletePostRegenerationRelations`.

Assemblies and Components

This section describes the Web.Link functions that access the functions of a 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 `pfcAssembly` is an instance of `pfcSolid`. The `pfcAssembly` object can therefore be used as input to any of the `pfcSolid` and `pfcModel` 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`, 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 Web.Link functions for assemblies are those that operate on the components of an assembly. The object `pfcComponentFeat`, which is an instance of `pfcFeature` 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 `pfcFeature` 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 `pfcComponentPath`, which is used as the input to Web.Link assembly functions.

The following figure shows an assembly hierarchy with two examples of the contents of a `pfcComponentPath` object.
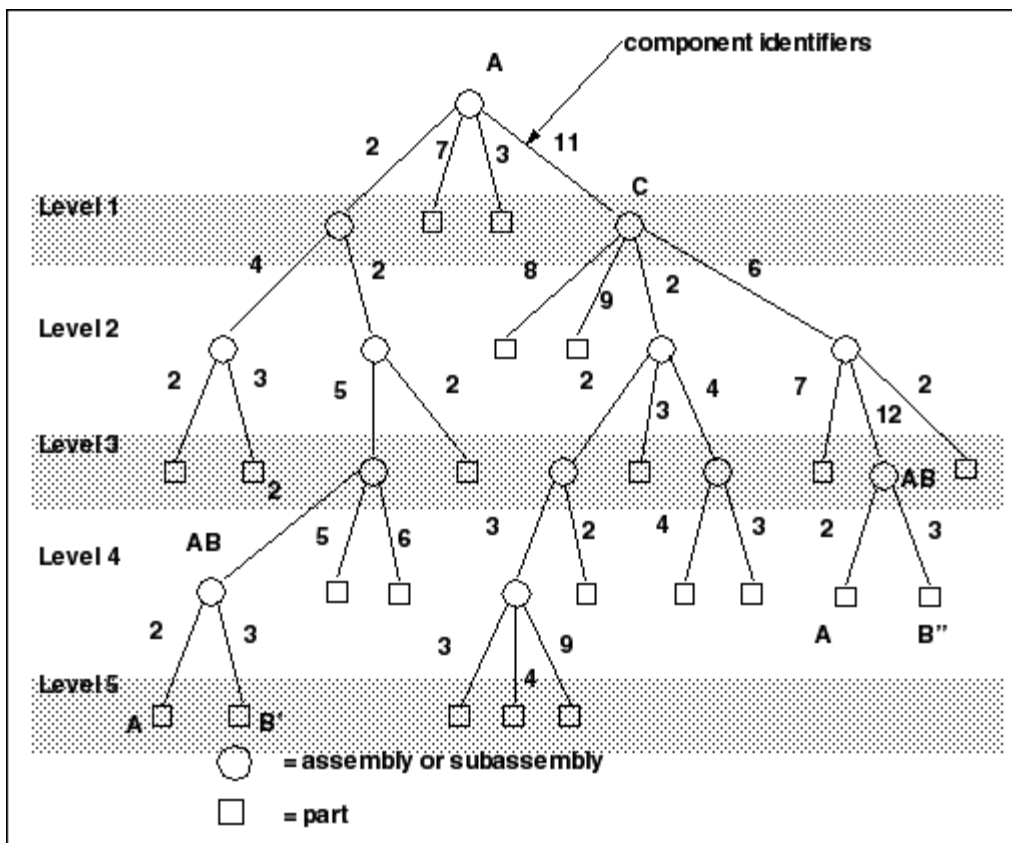
In the assembly shown in the figure, subassembly C is component identifier 11 within assembly A, Part B is component identifier 3 within assembly AB, and so on. The subassembly AB occurs twice. To refer to the two occurrences of part B, use the following:

```
(?)Component B'                 Component B"
ComponentIds.Item(0) = 2    ComponentIds.Item(1) = 11
ComponentIds.Item(1) = 2    ComponentIds.Item(2) = 6
ComponentIds.Item(2) = 5    ComponentIds.Item(3) = 12
ComponentIds.Item(3) = 2    ComponentIds.Item(4) = 3
ComponentIds.Item(4) = 3
```

The object `pfcComponentPath` is one of the main portions of the `pfcSelection` object.

Assembly Components

Methods and Properties Introduced:

- **pfcComponentFeat.IsBulkitem**
- **pfcComponentFeat.IsSubstitute**
- **pfcComponentFeat.CompType**
- **pfcComponentFeat.ModelDescr**
- **pfcComponentFeat.IsPlaced**
- **pfcComponentFeat.IsPackaged**
- **pfcComponentFeat.IsUnderconstrained**
- **pfcComponentFeat.IsFrozen**
- **pfcComponentFeat.Position**
- **pfcComponentFeat.CopyTemplateContents()**
- **pfcComponentFeat.CreateReplaceOp()**

The property `pfcComponentFeat.IsBulkitem` 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 property `pfcComponentFeat.IsSubstitute` 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 property `pfcComponentFeat.CompType` enables you to set the type of the assembly component. The component type identifies the purpose of the component in a manufacturing assembly.

The property `pfcComponentFeat.ModelDescr` returns the model descriptor of the component part or subassembly.

Note

From Pro/ENGINEER Wildfire 4.0 onwards, the property `pfcComponentFeat.ComponentFeat.GetModelDescr` throws an exception `pfcXtoolkitCantOpen` if called on an assembly component whose immediate generic is not in session. Handle this exception and typecast the assembly component as `pfcSolid`, which in turn can be typecast as `pfcFamilyMember`, and use the method `pfcFamilyMember.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 property `pfcComponentFeat.IsPlaced` 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 property `pfcComponentFeat.IsPackaged` to determine if the specified component is packaged.

The property `pfcComponentFeat.IsUnderconstrained` determines if the specified component is underconstrained, that is, it possesses some constraints but is not fully constrained.

The property `pfcComponentFeat.IsFrozen` 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 property `pfcComponentFeat.Position` 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 property modifies the assembly component data but does not regenerate the assembly component. To regenerate the component, use the method `pfcComponentFeat.Regenerate()`.

The method `pfcComponentFeat.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.ExecuteFeatureOps()`.

Example Code: Replacing Instances

The sample code in the file `pfcComponentFeatExamples.js` located at `<creo_weblink_loadpoint>/weblinkexamples/jscript` 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.Regenerate()**

The method `pfcComponentFeat.ComponentFeat.Regenerate` regenerates an assembly component. The method regenerates the assembly component just as in an interactive Creo Parametric session.

Creating a Component Path

Methods Introduced

- **MpfcAssembly.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 and Properties Introduced:

- **pfcComponentPath.Root**
- **pfcComponentPath.ComponentIds**
- **pfcComponentPath.Leaf**
- **pfcComponentPath.GetTransform()**
- **pfcComponentPath.SetTransform()**
- **pfcComponentPath.GetIsVisible()**

The property `pfcComponentPath.Root` returns the assembly at the head of the component path object.

The property `pfcComponentPath.ComponentIds` returns the sequence of ids which is the path to the particular component.

The property `pfcComponentPath.Leaf` returns the solid model at the end of the component path.

The method `pfcComponentPath.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 `pfcComponentPath.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.AssembleComponent()**
- **pfcAssembly.AssembleByCopy()**
- **pfcComponentFeat.GetConstraints()**
- **pfcComponentFeat.SetConstraints()**

The method `pfcAssembly.AssembleComponent()` adds a specified component model to the assembly at the specified initial position. The position is specified in the format defined by the class `pfcTransform3D`. 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.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.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.Regenerate()`.

Constraint Attributes

Methods and Properties Introduced:

- **pfcConstraintAttributes.Create()**
- **pfcConstraintAttributes.Force**
- **pfcConstraintAttributes.Ignore**

The method `pfcConstraintAttributes.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.

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. 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 and Properties Introduced:

- **pfcComponentConstraint.Create()**
- **pfcComponentConstraint.Type**
- **pfcComponentConstraint.AssemblyReference**
- **pfcComponentConstraint.AssemblyDatumSide**
- **pfcComponentConstraint.ComponentReference**
- **pfcComponentConstraint.ComponentDatumSide**
- **pfcComponentConstraint.Offset**
- **pfcComponentConstraint.Attributes**
- **pfcComponentConstraint.UserDefinedData**

The method `pfcComponentConstraint.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 properties listed above to access the parameters of the component constraint object.

Redefining and Rerouting Assembly Components

These functions enable you to reroute previously assembled components, just as in an interactive Creo Parametric session.

Methods Introduced:

- **pfcComponentFeat.RedefineThroughUI ()**
- **pfcComponentFeat.MoveThroughUI ()**

The method `pfcComponentFeat.RedefineThroughUI()` must be used in interactive Web.Link applications. This method displays the Creo Parametric **Constraint** dialog box. This enables the end user to redefine the constraints interactively. The control returns to Web.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 Web.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.js` located at `<creo_weblink_loadpoint>/weblinkexamples/jscript` 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.js` located at `<creo_weblink_loadpoint>/weblinkexamples/jscript` 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 and Properties Introduced:

- **pfcAssembly.IsExploded**
- **pfcAssembly.Explode()**
- **pfcAssembly.UnExplode()**
- **pfcAssembly.GetActiveExplodedState()**
- **pfcAssembly.GetDefaultExplodedState()**
- **pfcExplodedState.Activate()**

The methods `pfcAssembly.Explode()` and `pfcAssembly.UnExplode()` enable you to determine and change the explode status of the assembly object.

The property `pfcAssembly.IsExploded` reports whether the specified assembly is currently exploded.

The method `pfcAssembly.GetActiveExplodedState()` returns the current active explode state.

The method `pfcAssembly.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 and Properties Introduced:

- **pfcAssembly.AssembleSkeleton()**
- **pfcAssembly.AssembleSkeletonByCopy()**
- **pfcAssembly.GetSkeleton()**
- **pfcAssembly.DeleteSkeleton()**
- **pfcSolid.IsSkeleton**

The method `pfcAssembly.AssembleSkeleton()` adds an existing skeleton model to the specified assembly.

The method `pfcAssembly.GetSkeleton()` returns the skeleton model of the specified assembly.

The method `pfcAssembly.DeleteSkeleton()` deletes a skeleton model component from the specified assembly.

The method `pfcAssembly.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 property `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.

Family Tables

This section describes how to use Web.Link classes and methods to access and manipulate family table information.

Working with Family Tables

Web.Link provides several methods for accessing family table information. Because every model inherits from the class `pfcFamilyMember`, every model can have a family table associated with it.

Accessing Instances

Methods and Properties Introduced:

- **pfcFamilyMember.Parent**
- **pfcFamilyMember.GetImmediateGenericInfo()**
- **pfcFamilyMember.GetTopGenericInfo()**
- **pfcFamilyTableRow.CreateInstance()**
- **pfcFamilyMember.ListRows()**
- **pfcFamilyMember.GetRow()**
- **pfcFamilyMember.RemoveRow()**
- **pfcFamilyTableRow.InstanceName**
- **pfcFamilyTableRow.IsLocked**

To get the generic model for an instance, call the property `pfcFamilyMember.Parent`.

From Pro/ENGINEER Wildfire 4.0 onwards, the behavior of the property `pfcFamily.FamilyMember.GetParent` has changed as a result of performance improvement in family table retrieval mechanism. When you now call the property `pfcFamily.FamilyMember.GetParent`, it throws an exception `pfcXToolkitCantOpen`, 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 `pfcFamilyMember.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 `pfcFamilyMember.GetTopGenericInfo()`.

Similarly, the method `pfcFamilyTableRow.CreateInstance()` returns an instance model created from the information stored in the `pfcFamilyTableRow` object.

The method `pfcFamilyMember.ListRows()` returns a sequence of all rows in the family table, whereas `pfcFamilyMember.GetRow()` gets the row object with the name you specify.

Use the method `pfcFamilyMember.RemoveRow()` to permanently delete the row from the family table.

The property `pfcFamilyTableRow.InstanceName` returns the name that corresponds to the invoking row object.

To control whether the instance can be changed or removed, call the property `pfcFamily.FamilyTableRow.GetIsLocked`.

Accessing Columns

Methods and Properties Introduced:

- **pfcFamilyMember.ListColumns()**
- **pfcFamilyMember.GetColumn()**
- **pfcFamilyMember.RemoveColumn()**
- **pfcFamilyTableColumn.Symbol**
- **pfcFamilyTableColumn.Type**
- **pfcFamColModelItem.RefItem**

pfcFamColParam.RefParam

The method `pfcFamilyMember.ListColumns()` returns a sequence of all columns in the family table.

The method `pfcFamilyMember.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 `pfcFamilyMember.RemoveColumn()`.

The property `pfcFamilyTableColumn.Symbol` returns the string symbol at the top of the column, such as `D4` or `F5`.

The property `pfcFamilyTableColumn.Type` returns an enumerated value indicating the type of parameter governed by the column in the family table.

The property `pfcFamColModelItem.RefItem` returns the `pfcModelItem` (`Feature` or `Dimension`) controlled by the column, whereas `pfcFamily.FamColParam.GetRefParam` returns the `Parameter` controlled by the column.

Accessing Cell Information

Methods and Properties Introduced:

- **pfcFamilyMember.GetCell()**
- **pfcFamilyMember.GetCellIsDefault()**
- **pfcFamilyMember.SetCell()**
- **pfcParamValue.StringValue**
- **pfcParamValue.IntValue**
- **pfcParamValue.DoubleValue**
- **pfcParamValue.BoolValue**

The method `pfcFamilyMember.GetCell()` returns a string `pfcParamValue` that corresponds to the cell at the intersection of the row and column arguments. Use the method `pfcFamilyMember.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 `pfcFamilyMember.SetCell()` assigns a value to a column in a particular family table instance.

The `pfcParamValue.StringValue`, `pfcParamValue.IntValue`, `pfcParamValue.DoubleValue`, and `pfcModelItem.ParamValue.GetBoolValue` properties are used to get the different types of parameter values.

Creating Family Table Instances

Methods Introduced:

- **pfcFamilyMember.AddRow()**
- **MpfcModelItem.CreateStringParamValue()**
- **MpfcModelItem.CreateIntParamValue()**
- **MpfcModelItem.CreateDoubleParamValue()**
- **MpfcModelItem.CreateBoolParamValue()**

Use the method `pfcFamilyMember.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:

- **pfcFamilyMember.CreateDimensionColumn()**
- **pfcFamilyMember.CreateParamColumn()**
- **pfcFamilyMember.CreateFeatureColumn()**
- **pfcFamilyMember.CreateComponentColumn()**
- **pfcFamilyMember.CreateCompModelColumn()**

- **pfcFamilyMember.CreateGroupColumn()**
- **pfcFamilyMember.CreateMergePartColumn()**
- **pfcFamilyMember.CreateColumn()**
- **pfcFamilyMember.AddColumn()**
- **MpfcModelItem.CreateStringParamValue()**

The above methods initialize a column based on the input argument. These methods assign the proper symbol to the column header.

The method `pfcFamilyMember.CreateColumn()` creates a new column given a properly defined symbol and column type. The results of this call should be passed to the method `pfcFamilyMember.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.js` located at `<creo_weblink_loadpoint>/weblinkexamples/jscript` 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 column in the family table. All the dimensions, parameters, features, and components could be added to the family table using a similar method.

Interface

This section describes various methods of importing and exporting files in Web.Link.

Exporting Files and 2D Models

Method Introduced:

- **pfcModel.Export()**

The method `pfcModel.Model.Export` exports model data to a file. The exported files are placed in the current Creo Parametric working directory. The input parameters are:

- *filename*—Output file name including extensions
- *exportdata*—The `pfcExportInstructions` object that controls the export operation. The type of data that is exported is given by the `pfcExportType` object.

There are four general categories of files to which you can export models:

- File types whose instructions inherit from `pfcGeomExportInstructions`.

  These instructions export files that contain precise geometric information used by other CAD systems.
- File types whose instructions inherit from `pfcCoordSysExportInstructions`.

  These instructions export files that contain coordinate information describing faceted, solid models (without datums and surfaces).
- File types whose instructions inherit from `pfcFeatIdExportInstructions`.

  These instructions export information about a specific feature.
- General file types that inherit only from `pfcExportInstructions`.

  These instructions provide conversions to file types such as BOM (bill of materials).

For information on exporting to a specific format, see the Web.Link APIWizard and online help for the Creo Parametric interface.

Export Instructions

Methods Introduced:

- **pfcRelationExportInstructions.Create()**
- **pfcModelInfoExportInstructions.Create()**
- **pfcProgramExportInstructions.Create()**
- **pfcIGESFileExportInstructions.Create()**
- **pfcDXFExportInstructions.Create()**
- **pfcRenderExportInstructions.Create()**
- **pfcSTLASCIIExportInstructions.Create()**
- **pfcSTLBinaryExportInstructions.Create()**
- **pfcBOMExportInstructions.Create()**
- **pfcDWGSetupExportInstructions.Create()**
- **pfcFeatInfoExportInstructions.Create()**
- **pfcMFGFeatCLExportInstructions.Create()**
- **pfcMFGOperCLExportInstructions.Create()**
- **pfcMaterialExportInstructions.Create()**
- **pfcCGMFILEExportInstructions.Create()**
- **pfcInventorExportInstructions.Create()**
- **pfcFIATExportInstructions.Create()**
- **pfcConnectorParamExportInstructions.Create()**
- **pfcCableParamsFileInstructions.Create()**
- **pfcCATIAFacetsExportInstructions.Create()**

- **pfcVRMLModelExportInstructions.Create()**
- **pfcSTEP2DExportInstructions.Create()**
- **pfcMedusaExportInstructions.Create()**
- **pfcCADDSExportInstructions.Create()**
- **pfcSliceExportData.Create()**
- **pfcNEUTRALFileExportInstructions.Create()**
- **pfcProductViewExportInstructions.Create()**
- **pfcBaseSession.ExportDirectVRML()**

Export Instructions Table

Class	Used to Export
pfcRelationExportInstructions	A list of the relations and parameters in a part or assembly
pfcModelInfoExportInstructions	Information about a model, including units information, features, and children
pfcProgramExportInstructions	A program file for a part or assembly that can be edited to change the model
pfcIGESExportInstructions	A drawing in IGES format
pfcDXFExportInstructions	A drawing in DXF format
pfcRenderExportInstructions	A part or assembly in RENDER format
pfcSTLASCIIExportInstructions	A part or assembly to an ASCII STL file
pfcSTLBinaryExportInstructions	A part or assembly in a binary STL file
pfcBOMExportInstructions	A BOM for an assembly
pfcDWGSetupExportInstructions	A drawing setup file
pfcFeatInfoExportInstructions	Information about one feature in a part or assembly
pfcMfgFeatCLExportInstructions	A cutter location (CL) file for one NC sequence in a manufacturing assembly
pfcMfgOperClExportInstructions	A cutter location (CL) file for all the NC sequences in a manufacturing assembly
pfcMaterialExportInstructions	A material from a part
pfcCGMFILEExportInstructions	A drawing in CGM format
pfcInventorExportInstructions	A part or assembly in Inventor format
pfcFIATExportInstructions	A part or assembly in FIAT format
pfcConnectorParamExportInstructions	The parameters of a connector to a text file
pfcCableParamsFileInstructions	Cable parameters from an assembly
pfcCATIAFacetsExportInstructions	A part or assembly in CATIA format (as a faceted model)
pfcVRMLModelExportInstructions	A part or assembly in VRML format
pfcSTEP2DExportInstructions	A two-dimensional STEP format file
pfcMedusaExportInstructions	A drawing in MEDUSA file
pfcCADDSExportInstructions	A CADDS5 solid model
pfcNEUTRALFileExportInstructions	A Creo Parametric part to neutral format

`pfcProductViewExportInstructions`	A part, assembly, or drawing in Creo View format
`pfcExport.SliceExportData`	A slice export format

Note

The New Instruction Classes replace the following Deprecated Classes:

Deprecated Classes	New Instruction Classes
`pfcSTEPExportInstructions`	`pfcSTEP3DExportInstructions`
`pfcVDAExportInstructions`	`pfcVDA3DExportInstructions`
`pfcIGES3DExportInstructions`	`pfcIGES3DNewExportInstructions`

Exporting Drawing Sheets

The options required to export multiple sheets of a drawing are given by the `pfcExport2DOption` object.

Methods Introduced:

- **pfcExport2DOption.Create()**
- **pfcExport2DOption.ExportSheetOption**
- **pfcExport2DOption.ModelSpaceSheet**
- **pfcExport2DOption.Sheets**

The method `pfcModel.pfcModel.Export2DOptions_Create` creates a new instance of the `pfcExport2DOption` object. This object contains the following options:

- *ExportSheetOption*—Specifies the option for exporting multiple drawing sheets. Use the property `pfcExport2DOption.ExportSheetOption` to set the option for exporting multiple drawing sheets. The options are given by the `pfcExport2DSheetOption` 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 property `pfcExport2DOption.ModelSpaceSheet` 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 property `pfcModel.Export2DOption.SetSheets` to set this option.

Exporting to Faceted Formats

The methods described in this section support the export of Creo Parametric drawings and solid models to faceted formats like `CATIA CGR`.

Methods Introduced:

- **pfcTriangulationInstructions.AngleControl**
- **pfcTriangulationInstructions.ChordHeight**
- **pfcTriangulationInstructions.StepSize**
- **pfcTriangulationInstructions.FacetControlOptions**

The property `pfcTriangulationInstructions.AngleControl` 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 property `pfcTriangulationInstructions.ChordHeight` and `pfcExport.TriangulationInstructions.SetChordHeight` to get and set the chord height for the exported facet drawings and models.

The methods `pfcTriangulationInstructions.StepSize` 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 property `pfcCoordSysExportInstructions.StepSize` 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 property `pfcExport.TriangulationInstructions.GetFacetControlOptions` control the facet export options using bit flags. You can set the bit flags using the pfc`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 Creo Parametric user interface are used during the export operation.

- `FACET_CHORD_HEIGHT_DEFAULT`—Uses the default value set in the Creo Parametric user interface for the chord height.

- `FACET_ANGLE_CONTROL_DEFAULT`—Uses the default value set in the Creo Parametric user interface for the angle control.

- `FACET_STEP_SIZE_DEFAULT`—Uses the default value set in the 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.

- `FacetControlFlag_nil`—Nil option.

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:

- **pfcCoordSysExportInstructions.CsysName**
- **pfcCoordSysExportInstructions.Quality**
- **pfcCoordSysExportInstructions.MaxChordHeight**
- **pfcCoordSysExportInstructions.AngleControl**
- **pfcCoordSysExportInstructions.GetSliceExportData()**
- **pfcCoordSysExportInstructions.SetSliceExportData()**
- **pfcCoordSysExportInstructions.StepSize**
- **pfcCoordSysExportInstructions.FacetControlOptions**

The property `pfcCoordSysExportInstructions.CsysName` 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 property `pfcCoordSysExportInstructions.Quality` can be used instead of

`pfcCoordSysExportInstructions.MaxChordHeight` and `pfcCoordSysExportInstructions.AngleControl` . 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 property `pfcModel.CoordSysExportInstructions.GetMaxChordHeight` 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 property `pfcModel.CoordSysExportInstructions.GetAngleControl` 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 `pfcCoordSysExportInstructions.GetSliceExportData()` and `pfcCoordSysExportInstructions.SetSliceExportData()` get and set the `pfcSliceExportData` 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 property `pfcSliceExportData.CompIds` to work with the component IDs.

The property `pfcCoordSysExportInstructions.StepSize` 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 property `pfcModel.CoordSysExportInstructions.GetFacetControlOptions` control the facet export options using bit flags. You can set the bit flags using the `pfcFacetControlFlag` object. For more information on the bit flag values, please refer to the section <u>Exporting to Faceted Formats</u>.

Exporting to PDF and U3D

The methods described in this section support the export of 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 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 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:

- **pfcPDFExportInstructions.Create()**
- **pfcPDFExportInstructions.FilePath**
- **pfcPDFExportInstructions.Options**
- **pfcPDFExportInstructions.ProfilePath**
- **pfcPDFOption.Create()**
- **pfcPDFOption.OptionType**
- **pfcPDFOption.OptionValue**

The method `pfcPDFExportInstructions.Create()` creates a new instance of the `pfcPDFExportInstructions` data object that describes how to export 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 property `pfcPDFExportInstructions.FilePath` to set the name of the output file.

- *Options*—Specifies a collection of PDF export options of the type `pfcPDFOption`. Create a new instance of this object using the method `pfcPDFOption.Create()`. This object contains the following attributes:

  - *OptionType*—Specifies the type of option in terms of the `pfcPDFOptionType` enumerated class. Set this option using the property `pfcPDFOption.OptionType`.

  - *OptionValue*—Specifies the value of the option in terms of the `pfcArgValue` object. Set this option using the property `pfcExport.PDFOption.SetOptionValue`.

  Use the property `pfcPDFExportInstructions.Options` to set the collection of PDF export options.

- *ProfilePath*—Specifies the export profile path. Use the property `pfcPDFExportInstructions.ProfilePath` to set the profile path. When you set the profile path, the PDF export options set in the data object `pfcPDFExportInstructions` data object are ignored when the method `pfcModel.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` enumerated 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 `pfcPDFFontStrokeMode` enumerated 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 `pfcPDFColorDepth` enumerated 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 `pfcPDFHiddenLineMode` enumerated 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 `pfcPDFLayerMode` enumerated 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 `pfcPDFParameterMode` enumerated 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 pfcPDFRestrictOperationsMode enumerated 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 pfcPDFPrintingMode enumerated 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 pfcPDFRestrictOperationsMode enumerated class. The default value is true.
- PDFOPT_PENTABLE—If true, uses the standard 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 pfcPDFLinecap enumerated 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 `pfcPDFLinejoin` enumerated 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 Creo Parametric drawing that are to be exported to PDF. It is given by the `pfcPrintSheets` 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 `pfcPDFExportMode` enumerated 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 `pfcPDFOptionType` enumerated 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 `pfcPDFU3DLightingMode` enumerated class.
- `PDFOPT_RENDER_STYLE_DEFAULT`—Enables you to set the default rendering style used while exporting 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 `pfcPDFU3DRenderMode` enumerated class.
- `PDFOPT_SIZE`—Allows you to specify the page size of the exported PDF file. The values for this option are given by the `pfcPlotPaperSize` enumerated 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 `pfcSheetOrientation` enumerated 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 pfcPDFSelectedViewMode enumerated 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 pfcPDFSaveMode enumerated 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 pfcPDFLayerMode 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

Web.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.Export().

Export Instructions

Methods and Properties Introduced:

- **pfcExport3DInstructions.Configuration**
- **pfcExport3DInstructions.ReferenceSystem**
- **pfcExport3DInstructions.Geometry**
- **pfcExport3DInstructions.IncludedEntities**
- **pfcExport3DInstructions.LayerOptions**
- **pfcGeometryFlags.Create()**
- **pfcInclusionFlags.Create()**
- **pfcLayerExportOptions.Create()**
- **pfcSTEP3DExportInstructions.Create()**
- **pfcVDA3DExportInstructions.Create()**
- **pfcIGES3DNewExportInstructions.Create()**
- **pfcCATIAModel3DExportInstructions.Create()**
- **pfcACIS3DExportInstructions.Create()**
- **pfcCatiaPart3DExportInstructions.Create()**
- **pfcCatiaProduct3DExportInstructions.Create()**
- **pfcCatiaCGR3DExportInstructions.Create()**
- **pfcDXF3DExportInstructions.Create()**
- **pfcDWG3DExportInstructions.Create()**

- **pfcJT3DExportInstructions.Create()**
- **pfcParaSolid3DExportInstructions.Create()**
- **pfcUG3DExportInstructions.Create()**
- **pfcTriangulationInstructions.Create()**

The class `pfcExport3DInstructions` contains data to export a part or an assembly to a specifed 3D format. The fields of this class are:

- `Configuration`—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.

- `ReferenceSystem`—The reference coordinate system used for export. If this value is null, the system uses the default coordinate system.

- `Geometry`—The object describing the type of geometry to export. The `pfcGeometryFlags.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.

- `IncludedEntities`—The object returned by the method `pfcInclusionFlags.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.

- `LayerOptions`—The instructions object returned by the method `pfcLayerExportOptions.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

Class	Used to Export
pfcSTEP3DExportInstructions	A part or assembly in STEP format
pfcVDA3DExportInstructions	A part or assembly in VDA format
pfcIGES3DNewExportInstructions	A part or assembly in IGES format
pfcCATIAModel3DExportInstructions	A part or assembly in CATIA MODEL format

pfcACIS3DExportInstructions	A part or assembly in `ACIS` format
pfcCatiaPart3DExportInstructions	A part or assembly in `CATIA PART` format
pfcCatiaProduct3DExportInstructions	A part or assembly in `CATIA PRODUCT` format
pfcCatiaCGR3DExportInstructions	A part or assembly in `CATIA CGR` format
pfcJT3DExportInstructions	A part or assembly in `JT` format
pfcParaSolid3DExportInstructions	A part or assembly in `PARASOLID` format
pfcUG3DExportInstructions	A part or assembly in `UG` format
pfcDWG3DExportInstructions	A part or assembly in `DWG` format
pfcDXF3DExportInstructions	A part or assembly in `DXF` format
pfcTriangulationInstructions	A part or assembly in faceted format

Export Utilities

Methods Introduced:

- **pfcBaseSession.IsConfigurationSupported()**
- **pfcBaseSession.IsGeometryRepSupported()**

The method `pfcBaseSession.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 `pfcModel.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.ExportShrinkwrap()**

You can export the specified solid model as a shrinkwrap model using the method `pfcSolid.Solid.ExportShrinkwrap`. This method takes the `pfcShrinkwrapExportInstructions` object as an

argument.

Use the appropriate class given in the following table to create the required type of shrinkwrap. All the classes 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 `pfcShrinkwrapExportInstructions` or `pfcShrinkwrapModelExportInstructions`.

Type of Shrinkwrap Model	Class to Use
Surface Subset	`pfcShrinkwrapSurfaceSubset Instructions`
Faceted Part	`pfcShrinkwrapFacetedPart Instructions`
Faceted VRML	`pfcShrinkwrapFacetedVRML Instructions`
Faceted STL	`pfcShrinkwrapFacetedSTLInstructions`
Merged Solid	`ShrinkwrapMergedSolidInstructions`

Setting Shrinkwrap Options

The class `pfcShrinkwrapModelExportInstructions` 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.

Properties Introduced:

- **pfcShrinkwrapModelExportInstructions.Method**
- **pfcShrinkwrapModelExportInstructions.Quality**
- **pfcShrinkwrapModelExportInstructions.AutoHoleFilling**
- **pfcShrinkwrapModelExportInstructions.IgnoreSkeleton**
- **pfcShrinkwrapModelExportInstructions.IgnoreQuilts**
- **pfcShrinkwrapModelExportInstructions.AssignMassProperties**
- **pfcShrinkwrapModelExportInstructions.IgnoreSmallSurfaces**
- **pfcShrinkwrapModelExportInstructions.SmallSurfPercentage**
- **pfcShrinkwrapModelExportInstructions.DatumReferences**

The property `pfcShrinkwrapModelExportInstructions.Method` 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 property `pfcShrinkwrapModelExportInstructions.Quality` 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. The default value is 1.

The property `pfcShrinkwrapModelExportInstructions.AutoHoleFilling` sets a flag that forces 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 property `pfcShrinkwrap.ShrinkwrapModelExportInstructions.Get IgnoreSkeleton` determines whether the skeleton model geometry must be included in the shrinkwrap model.

The property `pfcShrinkwrap.ShrinkwrapModelExportInstructions.Get IgnoreQuilts` and determines whether external quilts must be included in the shrinkwrap model.

The property `pfcShrinkwrap.ShrinkwrapModelExportInstructions.SetAssign MassProperties` assigns 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 property `pfcShrinkwrap.ShrinkwrapModelExportInstructions`
`.SetIgnore`
`SmallSurfaces` sets a flag that forces 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 property `pfcShrinkwrap.ShrinkwrapModelExportInstructions`
`.GetSmall`
`SurfPercentage`.

The `pfcShrinkwrap.ShrinkwrapModelExportInstructions.SetDatum`
`References` specifies and selects the datum planes, points, curves, axes, and coordinate system references to be included in the shrinkwrap model.

Surface Subset Options

Methods and Properties Introduced:

- **pfcShrinkwrapSurfaceSubsetInstructions.Create()**
- **pfcShrinkwrapSurfaceSubsetInstructions.AdditionalSurfaces**
- **pfcShrinkwrapSurfaceSubsetInstructions.OutputModel**

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 property `pfcShrinkwrap.ShrinkwrapSurfaceSubsetInstructions.`
`SetAdditionalSurfaces` selects individual surfaces to be included in the shrinkwrap model.

The property `pfcShrinkwrap.ShrinkwrapSurfaceSubsetInstructions.`
`SetOutputModel` sets the template model.

Faceted Solid Options

The `pfcShrinkwrapFacetedFormatInstructions` class consists of the following types:

- `SWFACETED_PART`—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 class.

Properties Intoduced:

- **pfcShrinkwrapFacetedFormatInstructions.Format**
- **pfcShrinkwrapFacetedFormatInstructions.FramesFile**

The property `pfcShrinkwrapFacetedFormatInstructions.Format` returns the the output file format of the shrinkwrap model.

The property `pfcShrinkwrap.ShrinkwrapFacetedFormatInstructions.SetFramesFile` enables 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 and Properties Introduced:

- **pfcShrinkwrapFacetedPartInstructions.Create()**
- **pfcShrinkwrapFacetedPartInstructions.Lightweight**

The static method `pfcShrinkwrapFacetedPartInstructions.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 Creo Parametric part.

The property `pfcShrinkwrap.ShrinkwrapFacetedPartInstructions.`

`SetLight`
`weight` specifies if the Creo Parametric part is exported as a light weight faceted geometry.

VRML Export Options

Methods and Properties Introduced:

- **pfcShrinkwrapVRMLInstructions.Create()**
- **pfcShrinkwrapVRMLInstructions.OutputFile**

The static method `pfcShrinkwrapVRMLInstructions.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 property `pfcShrinkwrap.ShrinkwrapVRMLInstructions.SetOutputFile` specifies the name of the output file to be created.

STL Export Options

Methods and Properties Introduced:

- **pfcShrinkwrapVRMLInstructions.Create()**
- **pfcShrinkwrapVRMLInstructions.OutputFile**

The static method `pfcShrinkwrapVRMLInstructions.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 property `pfcShrinkwrapSTLInstructions.OutputFile` specifies the name of the output file to be created.

Merged Solid Options

Methods and Properties Introduced:

- **pfcShrinkwrapMergedSolidInstructions.Create()**
- **pfcShrinkwrapMergedSolidInstructions.AdditionalComponents**

The static method `pfcShrinkwrapMergedSolidInstructions.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 property `pfcShrinkwrap.ShrinkwrapMergedSolidInstructions.GetAdditional` `Components` specifies individual components of the assembly to be merged into the shrinkwrap model.

Importing Files

Method Introduced:

- **pfcModel.Import()**

The method `pfcModel.Model.Import` reads a file into Creo Parametric. The format must be the same as it would be if these files were created by Creo Parametric. The parameters are:

- *FilePath*—Absolute path of the file to be imported along with its extension.
- *ImportData*—The `pfcImportInstructions` object that controls the import operation.

Import Instructions

Methods Introduced:

- **pfcRelationImportInstructions.Create()**
- **pfcIGESSectionImportInstructions.Create()**
- **pfcProgramImportInstructions.Create()**
- **pfcConfigImportInstructions.Create()**
- **pfcDWGSetupImportInstructions.Create()**
- **pfcSpoolImportInstructions.Create()**
- **pfcConnectorParamsImportInstructions.Create()**
- **pfcASSEMTreeCFGImportInstructions.Create()**
- **pfcWireListImportInstructions.Create()**

- **pfcCableParamsImportInstructions.Create()**
- **pfcSTEPImport2DInstructions.Create()**
- **pfcIGESImport2DInstructions.Create()**
- **pfcDXFImport2DInstructions.Create()**
- **pfcDWGImport2DInstructions.Create()**

The methods described in this section create an instructions data object to import a file of a specified type into Creo Parametric. The details are as shown in the table below:

Class	Used to Import
pfcRelationImportInstructions	A list of relations and parameters in a part or assembly.
pfcIGESSectionImportInstructions	A section model in IGES format.
pfcProgramImportInstructions	A program file for a part or assembly that can be edited to change the model.
pfcConfigImportInstructions	Configuration instructions.
pfcDWGSetupImportInstructions	A drawing s/u file.
pfcSpoolImportInstructions	Spool instructions.
pfcConnectorParamsImportInstructions	Connector parameter instructions.
pfcASSEMTreeCFGImportInstructions	Assembly tree CFG instructions.
pfcWireListImportInstructions	Wirelist instructions.
pfcCableParamsImportInstructions	Cable parameters from an assembly.
pfcSTEPImport2DInstructions	A part or assembly in STEP format.
pfcIGESImport2DInstructions	A part or assembly in IGES format.
pfcDXFImport2DInstructions	A drawing in DXF format.
pfcDWGImport2DInstructions	A drawing in DWG format.

Note

- *The method pfcModel.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](#).*

Importing 2D Models

Method Introduced:

- **pfcBaseSession.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 `pfcImport2DInstructions` object that controls the import operation.

  The class `pfcImport2DInstructions` 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:

- **pfcBaseSession.GetImportSourceType()**
- **pfcBaseSession.ImportNewModel()**

For some input formats, the method `pfcBaseSession.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`. 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
- pfcModelType—Specifies the type of the model. It can be a part, assembly or drawing.
- *NewModelName*—Specifies a name for the imported model.
- pfcLayerImportFilter—Specifies the layer filter. This parameter is optional.

Plotting Files

From Pro/ENGINEER Wildfire 5.0 onwards, the pfcPlotInstructions object containing the instructions for plotting files has been deprecated. All the methods listed below for creating and accessing the instruction attributes in pfcPlotInstructions have also been deprecated. Use the new interface type pfcPrinterInstructions and its methods described in the next section.

Methods and Properties Deprecated:

- **pfcPlotInstructions.Create()**
- **pfcPlotInstructions.PlotterName**
- **pfcPlotInstructions.OutputQuality**
- **pfcPlotInstructions.UserScale**
- **pfcPlotInstructions.PenSlew**
- **pfcPlotInstructions.PenVelocityX**
- **pfcPlotInstructions.PenVelocityY**
- **pfcPlotInstructions.SegmentedOutput**
- **pfcPlotInstructions.LabelPlot**
- **pfcPlotInstructions.SeparatePlotFiles**
- **pfcPlotInstructions.PaperSize**
- **pfcPlotInstructions.PageRangeChoice**
- **pfcPlotInstructions.PaperSizeX**
- **pfcPlotInstructions.FirstPage**
- **pfcPlotInstructions.LastPage**

Printing Files

The printer instructions for printing a file are defined in pfcPrinterInstructions data object.

Methods Introduced:

- **pfcPrinterInstructions.Create()**
- **pfcPrinterInstructions.PrinterOption**
- **pfcPrinterInstructions.PlacementOption**
- **pfcPrinterInstructions.ModelOption**
- **pfcPrinterInstructions.WindowId**

The method pfcPrinterInstructions.Create() creates a new instance of the pfcPrinterInstructions object. The object contains the following instruction attributes:

- *PrinterOption*—Specifies the printer settings for printing a file in terms of the pfcPrintPrinterOption object. Set this attribute using the property pfcPrinterInstructions.PrinterOption.
- *PlacementOption*—Specifies the placement options for printing purpose in terms of the pfcPrintMdlOption object. Set this attribute using the property pfcPrinterInstructions.PlacementOption.
- *ModelOption*—Specifies the model options for printing purpose in terms of the pfcPrintPlacementOption object. Set this attribute using the property pfcPrinterInstructions.ModelOption.
- *WindowId*—Specifies the current window identifier. Set this attribute using the property pfcExport.PrinterInstructions.SetWindowId.

Printer Options

The printer settings for printing a file are defined in the pfcPrintPrinterOption object.

Methods Introduced:

- **pfcPrintPrinterOption.Create()**
- **pfcBaseSession.GetPrintPrinterOptions()**
- **pfcPrintPrinterOption.DeleteAfter**
- **pfcPrintPrinterOption.FileName**
- **pfcPrintPrinterOption.PaperSize**
- **pfcPrintSize.Create()**
- **pfcPrintSize.Height**
- **pfcPrintSize.Width**
- **pfcPrintSize.PaperSize**
- **pfcPrintPrinterOption.PenTable**
- **pfcPrintPrinterOption.PrintCommand**
- **pfcPrintPrinterOption.PrinterType**
- **pfcPrintPrinterOption.Quantity**
- **pfcPrintPrinterOption.RollMedia**
- **pfcPrintPrinterOption.RotatePlot**
- **pfcPrintPrinterOption.SaveMethod**
- **pfcPrintPrinterOption.SaveToFile**
- **pfcPrintPrinterOption.SendToPrinter**
- **pfcPrintPrinterOption.Slew**
- **pfcPrintPrinterOption.SwHandshake**
- **pfcPrintPrinterOption.UseTtf**

The method pfcPrintPrinterOption.Create() creates a new instance of the pfcPrintPrinterOption object.

The method pfcBaseSession.GetPrintPrinterOptions() retrieves the printer settings.

The pfcPrintPrinterOption 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 property pfcPrintPrinterOption.DeleteAfter to assign this option.
- *FileName*—Specifies the name of the file to be printed. Use the property pfcPrintPrinterOption.FileName to set the name.

    Note

    If the method pfcModel.Export() is called for pfcExportType 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 pfcPrintSize object. The property pfcPrintPrinterOption.PaperSize assigns the *PaperSize* option. Use the method pfcPrintSize.Create() to create a new instance of the pfcPrintSize object. This object contains the following options:

    - *Height*—Specifies the height of paper. Use the property pfcPrintSize.Height to set the paper height.
    - *Width*—Specifies the width of paper. Use the property pfcPrintSize.Width 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 property pfcPrintSize.PaperSize to set the paper size.
- *PenTable*—Specifies the file containing the pen table. Use the property pfcPrintPrinterOption.PenTable to set this option.
- *PrintCommand*—Specifies the command to be used for printing. Use the property pfcPrintPrinterOption.PrintCommand to set the command.
- *PrinterType*—Specifies the printer type. Use the property pfcPrintPrinterOption.PrinterType to assign the type.
- *Quantity*—Specifies the number of copies to be printed. Use the property pfcPrintPrinterOption.Quantity 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 property `pfcPrintPrinterOption.RollMedia` to assign this option.
- *RotatePlot*—Determines if the plot is rotated by 90 degrees. Set it to true to rotate the plot. Use the property `pfcPrintPrinterOption.RotatePlot` to set this option.
- *SaveMethod*—Specifies the save method in terms of the `pfcPrintSaveMethod` enumerated class. Use the property `pfcPrintPrinterOption.SaveMethod` 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 property `pfcPrintPrinterOption.SaveToFile` 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 property `pfcPrintPrinterOption.SendToPrinter` to set this option.
- *Slew*—Specifies the speed of the pen in centimeters per second in X and Y direction. Use the property `pfcPrintPrinterOption.Slew` 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 property `pfcPrintPrinterOption.SwHandshake` 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 property `pfcExport.PrintPrinterOption.SetUseTtf` to set this option.

Placement Options

The placement options for printing purpose are defined in the `pfcPrintPlacementOption` object.

Methods Introduced:

- **pfcPrintPlacementOption.Create()**
- **pfcBaseSession.GetPrintPlacementOptions()**
- **pfcPrintPlacementOption.BottomOffset**
- **pfcPrintPlacementOption.ClipPlot**
- **pfcPrintPlacementOption.KeepPanzoom**
- **pfcPrintPlacementOption.LabelHeight**
- **pfcPrintPlacementOption.PlaceLabel**
- **pfcPrintPlacementOption.Scale**
- **pfcPrintPlacementOption.ShiftAllCorner**
- **pfcPrintPlacementOption.SideOffset**
- **pfcPrintPlacementOption.X1ClipPosition**
- **pfcPrintPlacementOption.X2ClipPosition**
- **pfcPrintPlacementOption.Y1ClipPosition**
- **pfcPrintPlacementOption.Y2ClipPosition**

The method `pfcPrintPlacementOption.Create()` creates a new instance of the `pfcPrintPlacementOption` object.

The method `pfcBaseSession.GetPrintPlacementOptions()` retrieves the placement options.

The `pfcPrintPlacementOption` object contains the following options:

- *BottomOffset*—Specifies the offset from the lower-left corner of the plot. Use the property `pfcPrintPlacementOption.BottomOffset` 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 property `pfcPrintPlacementOption.ClipPlot` 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 property `pfcPrintPlacementOption.KeepPanzoom` to set this option.

- *LabelHeight*—Specifies the height of the label in inches. Use the property `pfcPrintPlacementOption.LabelHeight` to set this option.

- *PlaceLabel*—Specifies whether you want to place the label on the plot. Use the property `pfcPrintPlacementOption.PlaceLabel` to set this option.

- *Scale*—Specifies the scale used for the plot. Use the property `pfcPrintPlacementOption.Scale` 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 property `pfcPrintPlacementOption.ShiftAllCorner` to set this option.

- *SideOffset*—Specifies the offset from the sides. Use the property `pfcPrintPlacementOption.SideOffset` to set this option.

- *X1ClipPosition*—Specifies the first X parameter for defining the clip position. Use the property `pfcPrintPlacementOption.X1ClipPosition` to set this option.

- *X2ClipPosition*—Specifies the second X parameter for defining the clip position. Use the property `pfcPrintPlacementOption.X2ClipPosition` to set this option.

- *Y1ClipPosition*—Specifies the first Y parameter for defining the clip position. Use the property `pfcPrintPlacementOption.Y1ClipPosition` to set this option.

- *Y2ClipPosition*—Specifies the second Y parameter for defining the clip position. Use the property `pfcExport.PrintPlacementOption.SetY2ClipPosition` to set this option.

Model Options

The model options for printing purpose are defined in the `pfcPrintMdlOption` object.

Methods Introduced:

- **pfcPrintMdlOption.Create()**
- **pfcBaseSession.GetPrintMdlOptions()**
- **pfcPrintMdlOption.DrawFormat**
- **pfcPrintMdlOption.FirstPage**
- **pfcPrintMdlOption.LastPage**
- **pfcPrintMdlOption.LayerName**
- **pfcPrintMdlOption.LayerOnly**
- **pfcPrintMdlOption.Mdl**
- **pfcPrintMdlOption.Quality**
- **pfcPrintMdlOption.Segmented**
- **pfcPrintMdlOption.Sheets**
- **pfcPrintMdlOption.UseDrawingSize**
- **pfcPrintMdlOption.UseSolidScale**

The method `pfcPrintMdlOption.Create()` creates a new instance of the `pfcExport.PrintMdlOption` object.

The method `pfcBaseSession.GetPrintMdlOptions()` retrieves the model options.

The `pfcPrintMdlOption` object contains the following options:

- *DrawFormat*—Displays the drawing format used for printing. Use the property `pfcPrintMdlOption.DrawFormat` to set this option.

- *FirstPage*—Specifies the first page number. Use the property `pfcPrintMdlOption.FirstPage` to set this option.

- *LastPage*—Specifies the last page number. Use the property `pfcPrintMdlOption.LastPage` to set this option.

- *LayerName*—Specifies the name of the layer. Use the property `pfcPrintMdlOption.LayerName` to set the name.

- *LayerOnly*—Prints the specified layer only. Set this option to `true` to print the specified layer. Use the property `pfcPrintMdlOption.LayerOnly` to set this option.

- *Mdl*—Specifies the model to be printed. Use the property `pfcPrintMdlOption.Mdl` 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 property `pfcPrintMdlOption.Quality` 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 property

`pfcPrintMdlOption.Segmented` to set this option.

- *Sheets*—Specifies the sheets that need to be printed in terms of the `pfcPrintSheets` class. Use the property `pfcPrintMdlOption.Sheets` 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 property `pfcPrintMdlOption.UseDrawingSize` 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 property `pfcExport.PrintMdlOption.SetUseSolidScale` to set this option.

Plotter Configuration File (PCF) Options

The printing options for PCF file are defined in the `pfcPrinterPCFOptions` object.

Methods Introduced:

- **pfcPrinterPCFOptions.Create()**
- **pfcPrinterPCFOptions.PrinterOption**
- **pfcPrinterPCFOptions.PlacementOption**
- **pfcPrinterPCFOptions.ModelOption**

The method `pfcPrinterPCFOptions.Create()` creates a new instance of the `pfcPrinterPCFOptions` object.

The `pfcPrinterPCFOptions` object contains the following options:

- *PrinterOption*—Specifies the printer settings for printing a file in terms of the `pfcPrintPrinterOption` object. Set this attribute using the property `pfcPrinterPCFOptions.PrinterOption`.

- *PlacementOption*—Specifies the placement options for printing purpose in terms of the `pfcPrintMdlOption` object. Set this attribute using the property `pfcPrinterPCFOptions.PlacementOption`.

- *ModelOption*—Specifies the model options for printing purpose in terms of the `pfcPrintPlacementOption` object. Set this attribute using the property `pfcExport.PrinterPCFOptions.SetModelOption`.

Solid Operations

Method Introduced:

- **pfcSolid.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 `pfcIntfDataSource` object. The type of source data that can be imported is given by the `pfcIntfType` 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 `pfcImportFeatAttr` 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.js` located at `<creo_weblink_loadpoint>/weblinkexamples/jscript` 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.ExportRasterImage()

The method `pfcWindow.Window.ExportRasterImage` outputs a standard Creo Parametric raster output file.

Simplified Representations

Web.Link gives programmatic access to all the simplified representation functionality of Creo Parametric. Create simplified representations either permanently or on the fly and save, retrieve, or modify them by adding or deleting items.

Overview

Using Web.Link, you can create and manipulate assembly simplified representations just as you can using Creo Parametric interactively.

Note

Web.Link supports simplified representation of assemblies only, not parts.

Simplified representations are identified by the `pfcSimRep` class. This class is a child of `pfcModelItem`, 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 `pfcSimpRepInstructions` which contains several data objects and fields, including:

* `String`—The name of the simplified representation

* `pfcSimpRepAction`—The rule that controls the default treatment of items in the simplified representation.

* `pfcSimpRepItem`—An array of assembly components and the actions applied to them in the simplified representation.

A `pfcSimpRepItem` is identified by the assembly component path to that item. Each `pfcSimpRepItem` has it's own `pfcSimpRepAction` assigned to it. `pfcSimpRepAction` is a visible data object that includes a field of type `pfcSimpRepActionType`. You can use the property `pfcSimpRepAction.Action()` to set the actions. To delete an existing item, you must set the action as NULL.

`pfcSimpActionType` 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 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:

* **pfcBaseSession.RetrieveAssemSimpRep()**
* **pfcBaseSession.RetrieveGeomSimpRep()**
* **pfcBaseSession.RetrieveGraphicsSimpRep()**
* **pfcBaseSession.RetrieveSymbolicSimpRep()**
* **pfcRetrieveExistingSimpRepInstructions.Create()**

You can retrieve a named simplified representation from a model using the method

`pfcBaseSession.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. 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 `pfcBaseSession.RetrieveGeomSimpRep()`, `pfcBaseSession.RetrieveGraphicsSimpRep()`, and `pfcBaseSession.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:

- **pfcCreateNewSimpRepInstructions.Create()**
- **pfcSolid.CreateSimpRep()**
- **pfcSolid.DeleteSimpRep()**

To create a simplified representation, you must allocate and fill a `pfcSimpRepInstructions` object by calling the method `pfcCreateNewSimpRepInstructions.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.CreateSimpRep()`. This method returns the `pfcSimpRep` object for the new representation.

The method `pfcSolid.Solid.DeleteSimpRep` deletes a simplified representation from its model owner. The method requires only the `pfcSimpRep` object as input.

Extracting Information About Simplified Representations

Methods and Properties Introduced:

- **pfcSimpRep.GetInstructions()**
- **pfcSimpRepInstructions.DefaultAction**
- **pfcCreateNewSimpRepInstructions.NewSimpName**
- **pfcSimpRepInstructions.IsTemporary**
- **pfcSimpRepInstructions.Items**

Given the object to a simplified representation, `pfcSimpRep.GetInstructions()` fills out the `pfcSimpRepInstructions` object.

The `pfcSimpRepInstructions.DefaultAction`, `pfcCreateNewSimpRepInstructions.NewSimpName`, and `pfcSimpRepInstructions.IsTemporary` properties return the associated values contained in the `pfcSimpRepInstructions`object.

The property `pfcSimpRep.SimpRepInstructions.GetItems` returns all the items that make up the simplified representation.

Modifying Simplified Representations

Methods and Properties Introduced:

- **pfcSimpRep.GetInstructions()**
- **pfcSimpRep.SetInstructions()**
- **pfcSimpRepInstructions.DefaultAction**
- **pfcCreateNewSimpRepInstructions.NewSimpName**

- **pfcSimpRepInstructions.IsTemporary**

Using Web.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 `pfcSimpRepInstructions` object.

To modify an existing simplified representation retrieve it and then get the `pfcSimpRepInstructions` object by calling `pfcSimpRep.GetInstructions()`. If you created the representation programmatically within the same application, the `pfcSimpRepInstructions` object is already available. Once you have modified the data object, reassign it to the corresponding simplified representation by calling the method `pfcSimpRep.SetInstructions()`.

Adding Items to and Deleting Items from a Simplified Representation

Methods and Properties Introduced:

- **pfcSimpRepInstructions.Items**
- **pfcSimpRepItem.Create()**
- **pfcSimpRep.SetInstructions()**
- **pfcSimpRepReverse.Create()**
- **pfcSimpRepInclude.Create()**
- **pfcSimpRepExclude.Create()**
- **pfcSimpRepSubstitute.Create()**
- **pfcSimpRepGeom.Create()**
- **pfcSimpRepGraphics.Create()**

You can add and delete items from the list of components in a simplified representation using Web.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 `pfcSimpRepActionType` to `SIMPREP_EXCLUDE`.

How to Add Items

1. Get the `pfcSimpRepInstructions` 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 `pfcSimpRepItem` object for the item by calling the method `pfcSimpRepItem.Create()`.
4. Add the item to the `pfcSimpRepItem` sequence. Put the new `pfcSimpRepInstructions` using `pfcSimpRepInstructions.Items`.
5. Reassign the `pfcSimpRepInstructions` object to the corresponding `pfcSimpRep` object by calling `pfcSimpRep.SetInstructions()`

How to Remove Items

Follow the procedure above, except remove the unwanted `pfcSimpRepItem` from the sequence.

Simplified Representation Utilities

Methods Introduced:

- **pfcModelItemOwner.ListItems()**
- **pfcModelItemOwner.GetItemById()**
- **pfcSolid.GetSimpRep()**
- **pfcSolid.SelectSimpRep()**
- **pfcSolid.ActivateSimpRep()**
- **pfcSolid.GetActiveSimpRep()**

This section describes the utility methods that relate to simplified representations.

The method `pfcModelItemOwner.ListItems()` can list all of the simplified representations in a Solid.

The method `pfcModelItemOwner.GetItemById()` initializes a `pfcSimpRep.SimpRep` object. It takes an integer id.

Note

Web.Link supports simplified representation of Assemblies only, not Parts.

The method `pfcSolid.GetSimpRep()` initializes a `pfcSimpRep` 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.SelectSimpRep()` creates a 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.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`.

Task Based Application Libraries

Applications created using different Creo Parametric API products are interoperable. These products use Creo Parametric as the medium of interaction, eliminating the task of writing native-platform specific interactions between different programming languages.

Application interoperability allows Web.Link applications to call into Creo Parametric TOOLKIT from areas not covered in the native interface. It allows you to put an HTML front end on legacy Creo Parametric TOOLKIT applications.

Managing Application Arguments

Web.Link passes application data to and from tasks in other applications as members of a sequence of `pfcArgument` 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)
- `pfcSelection` (a selection of an item in a Creo Parametric session)
- `pfcTransform3D` (a coordinate system transformation matrix)

Methods and Properties Introduced:

- **MpfcArgument.CreateIntArgValue()**
- **MpfcArgument.CreateDoubleArgValue()**
- **MpfcArgument.CreateBoolArgValue()**
- **MpfcArgument.CreateASCIIStringArgValue()**
- **MpfcArgument.CreateStringArgValue()**
- **MpfcArgument.CreateSelectionArgValue()**
- **MpfcArgument.CreateTransformArgValue()**
- **pfcArgValue.discr**
- **pfcArgValue.IntValue**
- **pfcArgValue.DoubleValue**
- **pfcArgValue.BoolValue**
- **pfcArgValue.ASCIIStringValue**
- **pfcArgValue.StringValue**
- **pfcArgValue.SelectionValue**
- **pfcArgValue.TransformValue**

The class `pfc ArgValue` contains one of the seven types of values. Web.Link provides different methods to create each of the seven types of argument values.

The property `pfcArgValue.discr` 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 and Properties Introduced:

- **pfcArgument.Create()**
- **pfcArgument.Label**
- **pfcArgument.Value**

The method `pfcArgument.Create()` creates a new argument. Provide a name and value as the input arguments of this method.

The property `pfcArgument.Label` returns the label of the argument.

The property `pfcArgument.Argument.GetValue` returns the value of the argument.

Launching a Creo Parametric TOOLKIT DLL

The methods described in this section enable a Web.Link user to register and launch a Creo Parametric TOOLKIT DLL from a Web.Link application. The ability to launch and control a Creo Parametric TOOLKIT application enables the following:

- Reuse of existing Creo Parametric TOOLKIT code with Web.Link applications.
- ATB operations.

Methods and Properties Introduced:

- **pfcBaseSession.LoadProToolkitDll()**
- **pfcBaseSession.LoadProToolkitLegacyDll()**
- **pfcBaseSession.GetProToolkitDll()**
- **pfcDll.ExecuteFunction()**
- **pfcDll.Id**
- **pfcDll.IsActive()**
- **pfcDll.Unload()**

Use the method `pfcBaseSession.LoadProToolkitDll()` to register and start a 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 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 Creo Parametric TOOLKIT DLL.

In order to register and start a legacy Pro/TOOLKIT DLL that is not Unicode-compliant, use the method `pfcBaseSession.LoadProToolkitLegacyDll()`. This method conveys to 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 Web.Link application to load a pre-Wildfire 4.0 Pro/TOOLKIT DLL.

Use the method `pfcBaseSession.GetProToolkitDll()` to obtain a 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 `pfcDll.Id` returns the DLL identifier string.

Use the method `pfcDll.ExecuteFunction()` to call a properly designated function in the Creo Parametric TOOLKIT DLL library. The input parameters of this method are:

- *FunctionName*—Name of the function in the Creo Parametric TOOLKIT DLL application.
- *InputArguments*—Input arguments to be passed to the library function.

The method returns an object of class `pfcFunctionReturn`. This interface contains data returned by a 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 `pfcDll.IsActive()` determines whether a 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 Creo Parametric TOOLKIT DLL previously loaded by the method `pfcSession.BaseSession.LoadProToolkitDll` and the application's `user_terminate()` function is called.

Launching Tasks from J-Link Task Libraries

The methods described in this section allow you to launch tasks from a predefined task library.

Methods Introduced:

- **pfcBaseSession.StartJLinkApplication()**
- **pfcJLinkApplication.ExecuteTask()**
- **pfcJLinkApplication.IsActive()**
- **pfcJLinkApplication.Stop()**

Use the method `pfcBaseSession.StartJLinkApplication()` to start a 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 application.
- *ClassName*—Specifies the name of the Java class that contains the application's start and stop method. This should be a fully qualified Java package and class name.
- *StartMethod*—Specifies the start method of the application.
- *StopMethod*—Specifies the stop method of the application.
- *AdditionalClassPath*—Specifies the locations of packages and classes that must be loaded when starting this 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 Creo Parametric.

Upon starting the application, the static `start()` method is invoked. The method returns a `pfcJLink.JLinkApplication` referring to the application.

The method `pfcJLinkApplication.ExecuteTask()` calls a registered task method in a 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`.

The method outputs an array of output arguments.

The method `pfcJLinkApplication.IsActive()` returns a `True` value if the application specified by the `pfcJLinkApplication` object is active.

The method `pfcJLink.JLinkApplication.Stop` stops the application specified by the `pfcJLinkApplication` object. This method activates the application's static `Stop()` method.

Graphics

This section covers Web.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 class `pfcMouseStatus`.

The enumerated values are defined in `pfcMouseButton` and are as follows:

- `MOUSE_BTN_LEFT`
- `MOUSE_BTN_RIGHT`
- `MOUSE_BTN_MIDDLE`
- `MOUSE_BTN_LEFT_DOUBLECLICK`

Methods Introduced:

- **pfcSession.UIGetNextMousePick()**
- **pfcSession.UIGetCurrentMouseStatus()**

The method `pfcSession.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.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 `pfcOutline3D`.

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 Creo Parametric current window and use the color and linestyle set by calls to `pfcBaseSession.SetStdColorFromRGB()` and `pfcBaseSession.SetLineStyle()`. The methods draw the graphics in the Creo Parametric graphics color. The default graphics color is white.

The methods in this section are called using the class `pfcDisplay`. This class is extended by the `pfcBaseSession` class. This architecture allows you to call all these methods on any `pfcSession` object.

Methods Introduced:

- **pfcDisplay.SetPenPosition()**
- **pfcDisplay.DrawLine()**
- **pfcDisplay.DrawPolyline()**
- **pfcDisplay.DrawCircle()**

- **pfcDisplay.DrawArc2D()**
- **pfcDisplay.DrawPolygon2D()**

The method `pfcDisplay.SetPenPosition()` sets the point at which you want to start drawing a line. The function `pfcDisplay.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.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.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.DrawArc2D()` draws an arc in screen coordinates.

Controlling Graphics Display

Properties Introduced:

- **pfcDisplay.CurrentGraphicsColor**
- **pfcDisplay.CurrentGraphicsMode**

The property `pfcDisplay.CurrentGraphicsColor` returns the Creo Parametric standard color used to display graphics. The Creo Parametric default is `COLOR_DRAWING` (white).

The property `pfcDisplay.Display.GetCurrentGraphicsMode` returns the mode used to draw graphics:

- `DRAW_GRAPHICS_NORMAL`— Creo Parametric draws graphics in the required color in each invocation.

- `DRAW_GRAPHICS_COMPLEMENT`—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.

Example Code: Creating Graphics On Screen

The sample code in the file `pfcDisplayExamples.js` located at `<creo_weblink_loadpoint>/weblinkexamples/jscript` 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.

Displaying Text in the Graphics Window

Method Introduced:

- **pfcDisplay.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.

Creo Parametric and therefore are not redrawn when you select **View, Repaint**. To notify the Creo Parametric of these objects, create them inside the `OnDisplay()` method of the Display Listener.

Controlling Text Attributes

Properties Introduced:

- **pfcDisplay.TextHeight**
- **pfcDisplay.WidthFactor**
- **pfcDisplay.RotationAngle**

- **pfcDisplay.SlantAngle**

These properties control the attributes of text added by calls to `pfcDisplay.DrawText2D()`.

You can access 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

and Properties Introduced:

- **pfcDisplay.DefaultFont**
- **pfcDisplay.CurrentFont**
- **pfcDisplay.GetFontById()**
- **pfcDisplay.GetFontByName()**

The property `pfcDisplay.DefaultFont` returns the default Creo Parametric text font. The text fonts are identified in Creo Parametric by names and by integer identifiers. To find a specific font, use the methods `pfcDisplay.GetFontById()` or `pfcDisplay.Display.GetFontByName`.

External Data

This chapter explains using External Data in Web.Link.

External Data

This chapter describes how to store and retrieve external data. External data enables a Web.Link application to store its own data in a Creo Parametric database in such a way that it is invisible to the Creo Parametric user. This method is different from other means of storage accessible through the Creo Parametric user interface.

Introduction to External Data

External data provides a way for the Creo Parametric application to store its own private information about a 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 Creo Parametric; the application has complete control over form and content.

The external data for a specific 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 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:

Web.Link Type	Data
pfcExternalDataType.EXTDATA_INTEGER	integer
pfcExternalDataType.EXTDATA_DOUBLE	double
pfcExternalDataType.EXTDATA_STRING	string

The Web.Link interfaces used to access external data in Creo Parametric are:

Web.Link Type	Data Type
pfcExternalDataAccess	This is the top level object and is created when attempting to access external data.
pfcExternalDataClass	This is a class of external data and is identified by a unique name.
pfcExternalDataSlot	This is a container for one item of data. Each slot is stored in a class.
pfcExternalData	This is a compact data structure that contains either an integer, double or string value.

Compatibility with Creo Parametric TOOLKIT

Web.Link and Creo Parametric TOOLKIT share external data in the same manner. Web.Link external data is accessible by Creo Parametric TOOLKIT and the reverse is also true. However, an error will result if Web.Link attempts to access external data previously stored by Creo Parametric TOOLKIT as a stream.

Accessing External Data

Methods Introduced:

- **pfcModel.AccessExternalData()**
- **pfcModel.TerminateExternalData()**
- **pfcExternalDataAccess.IsValid()**

The method `pfcModel.AccessExternalData()` prepares Creo Parametric to read external data from the model file. It returns the `pfcExternalDataAccess` 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.TerminateExternalData()` stops 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 `pfcExternalDataAccess` object can be used to read and write data.

Storing External Data

Methods and Properties Introduced:

- **pfcExternalDataAccess.CreateClass()**
- **pfcExternalDataClass.CreateSlot()**
- **pfcExternalDataSlot.Value**

The first step in storing external data in a new class and slot is to set up a class using the method `pfcExternalDataAccess.CreateClass()`, which provides the class name. The method outputs `pfcExternalDataClass`, used by the application to reference the class.

The next step is to use `pfcExternalDataClass.CreateSlot()` to create an empty data slot and input a slot name. The method outputs a `pfcExternalDataSlot` object to identify the new slot.

Note

Slot names cannot begin with a number.

The property `pfcExternal.ExternalDataSlot.SetValue` specifies the data type of a slot and writes an item of that type to the slot. The input is a `pfcExternalData` object that you can create by calling any one of the methods in the next section.

Initializing Data Objects

Methods Introduced:

- **MpfcExternal.CreateIntExternalData()**
- **MpfcExternal.CreateDoubleExternalData()**
- **MpfcExternal.CreateStringExternalData()**

These methods initialize a `pfcExternalData` object with the appropriate data inputs.

Retrieving External Data

Methods and Properties Introduced:

- **pfcExternalDataAccess.LoadAll()**
- **pfcExternalDataAccess.ListClasses()**
- **pfcExternalDataClass.ListSlots()**
- **pfcExternalData.discr**
- **pfcExternalData.IntegerValue**
- **pfcExternalData.DoubleValue**
- **pfcExternalData.StringValue**

For improved performance, external data is not loaded automatically into memory with the model. When the

model is in session, call the method `pfcExternalDataAccess.LoadAll()` to retrieve all the external data for the specified model from the 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 `pfcExternalDataAccess.ListClasses()` returns a sequence of `pfcExternalDataClasses` registered in the model. The method `pfcExternalDataClass.ListSlots()` provide a sequence of `pfcExternalDataSlots` existing for each class.

To find out a data type of a `pfcExternalData`, call `pfcExternalData.discr` and then call one of these propertiesto get the data, depending on the data type:

- `pfcExternalData.IntegerValue`

- `pfcExternalData.DoubleValue`

- `pfcExternal.ExternalData.GetStringValue`

Exceptions

Most exceptions thrown by external data methods in Web.Link extend `pfcXExternalDataError`, which is a subclass of `pfcXToolkitError`.

An additional exception thrown by external data methods is `pfcXBadExternalData`. This exception signals an error accessing data. For example, external data access might have been terminated or the model might contain stream data from Creo Parametric TOOLKIT.

The following table lists these exceptions.

Exception	Cause
`pfcXExternalDataInvalidObject`	Generated when a model or class is invalid.
`pfcXExternalDataClassOrSlotExists`	Generated when creating a class or slot and the proposed class or slot already exists.
`pfcXExternalDataNamesTooLong`	Generated when a class or slot name is too long.
`pfcXExternalDataSlotNotFound`	Generated when a specified class or slot does not exist.
`pfcXExternalDataEmptySlot`	Generated when the slot you are attempting to read is empty.
`pfcXExternalDataInvalidSlotName`	Generated when a specified slot name is invalid.
`pfcXBadGetExternalData`	Generated when you try to access an incorrect data type in a `pfcExternalData` object.

Windchill Connectivity APIs

Creo Parametric has the capability to be directly connected to Windchill solutions, including Windchill Foundation, ProjectLink, and PDMLink servers. This access allows users to manage and control the product data seamlessly from within Creo Parametric.

This section lists Web.Link APIs that support Windchill servers and server operations in a connected Creo Parametric session.

Introduction

The methods introduced in this section provide support for the basic Windchill server operations from within Creo Parametric. With these methods, operations such as registering a Windchill server, managing workspaces, and check in or check out of objects will be possible via Web.Link. The capabilities of these APIs are similar to the operations available from within the Creo Parametric Creo Parametric client, with some restrictions.

Non-Interactive Mode Operations

Some of the APIs specified in this section operate only in batch mode and cannot be used in the normal Creo Parametric interactive mode. This restriction is mainly centered around the Web.Link registered servers, that is, servers registered by Web.Link are not available in the Creo Parametric Server Registry or in other locations in the Creo Parametric user interface such as the Folder Navigator and embedded browser. If a Web.Link customization requires the user to have interactive access to the server, the server must be registered via the normal Creo Parametric techniques, that is, either by entry in the Server Registry or by automatic registration of a previously registered server.

All of these APIs are supported from a non-interactive, that is, batch mode application or asynchronous application.

Accessing a Windchill Server from a Creo Parametric Session

Creo Parametric allows you to register Windchill servers as a connection between the Windchill database and Creo Parametric. Although the represented Windchill database can be from Windchill Foundation, Windchill ProjectLink, or Windchill PDMLink all types of databases are represented in the same way.

You can use the following identifiers when referring to Windchill servers in Web.Link:

*   Codebase URL—This is the root portion of the URL that is used to connect to a 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 Windchill server, you must establish a connection by registering the server in Creo Parametric. The methods described in this section allow you to connect to a Windchill server and access information related to the server.

Methods and Properties Introduced:

*   **pfcBaseSession.AuthenticateBrowser()**
*   **pfcBaseSession.GetServerLocation()**
*   **pfcServerLocation.Class**
*   **pfcServerLocation.Location**
*   **pfcServerLocation.Version**
*   **pfcServerLocation.ListContexts()**
*   **pfcServerLocation.CollectWorkspaces()**

Use the method `pfcBaseSession.AuthenticateBrowser()` to set the authentication context using a valid username and password. A successful call to this method allows the 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 Windchill servers, provided that you register the appropriate servers or servers immediately after setting the context.

The property `pfcServerLocation.Location` specifies 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 property `pfcServerLocation.Class` specifies the class of the server or server location. The values are:

- `Windchill`—Denotes either a Windchill Classic PDM server or a Windchill PDMLink server.

- `ProjectLink`—Denotes Windchill ProjectLink type of servers.

The property `pfcServerLocation.Version` specifies the version of Windchill that is configured on the server or server location, for example, "7.0" or "8.0." This method accepts the server codebase URL as the input.

Note

`pfcServer.ServerLocation.GetVersion` works only for Windchill servers and throws the `pfcExceptions.XToolkitUnsupported` exception, if the server is not a Windchill server.

The method `pfcServerLocation.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

The methods described in this section are restricted to the non-interactive mode only. Refer to the section, Non-Interactive Mode Operations, for more information.

Methods Introduced:

- **pfcBaseSession.RegisterServer()**
- **pfcServer.Activate()**
- **pfcServer.Unregister()**

The method `pfcBaseSession.RegisterServer()` registers the specified server with the codebase URL. A successful call to `pfcBaseSession.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.Activate()` sets the specified server as the active server in the 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

Properties Introduced:

- **pfcServer.IsActive**
- **pfcServer.Alias**
- **pfcServer.Context**

The property `pfcServer.IsActive` specifies if the server is active.

The property `pfcServer.Alias` returns the alias of a server if you specify the codebase URL.

The property `pfcServer.Server.GetContext` returns the active context of the active server.

Information on Servers in Session

Methods Introduced:

- **pfcBaseSession.GetActiveServer()**
- **pfcBaseSession.GetServerByAlias()**

- **pfcBaseSession.GetServerByUrl()**
- **pfcBaseSession.ListServers()**

The method `pfcBaseSession.GetActiveServer()` returns returns the active server handle.

The method `pfcBaseSession.GetServerByAlias()` returns the handle to the server matching the given server alias, if it exists in session.

The method `pfcBaseSession.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 and Properties Introduced:

- **pfcWorkspaceDefinition.Create()**
- **pfcWorkspaceDefinition.WorkspaceName**
- **pfcWorkspaceDefinition.WorkspaceContext**

The class `pfcWorkspaceDefinition` contains the name and context of the workspace. The method `pfcServerLocation.CollectWorkspaces()` returns an array of workspace data. Workspace data is also required for the method `pfcServer.CreateWorkspace()` to create a workspace with a given name and a specific context.

The method `pfcWorkspaceDefinition.Create()` creates a new workspace definition object suitable for use when creating a new workspace on the server.

The property `pfcWorkspaceDefinition.WorkspaceName` retrieves the name of the workspace.

The property `pfcServer.WorkspaceDefinition.GetWorkspaceContext` retrieves the context of the workspace.

Creating and Modifying the Workspace

Methods and Properties Introduced:

- **pfcServer.CreateWorkspace()**
- **pfcServer.ActiveWorkspace**
- **pfcServerLocation.DeleteWorkspace()**

All methods and properties described in this section, except `pfcServer.ActiveWorkspace`, are permitted only in the non-interactive mode. Refer to the section, [Non-Interactive Mode Operations](), for more information.

The method `pfcServer.CreateWorkspace()` creates and activates a new workspace.

The property `pfcServer.Server.GetActiveWorkspace` retrieves the name of the active workspace.

The method `pfcServer.ServerLocation.DeleteWorkspace` deletes the specified workspace. 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.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 Windchill server with an existing workspace:

1. Set the appropriate authentication context using the method `pfcBaseSession.AuthenticateBrowser()` with a valid username and password.
2. Look up the list of workspaces using the method `pfcServerLocation.CollectWorkspaces()`. If you already know the name of the workspace on the server, then ignore this step.
3. Register the workspace using the method `pfcBaseSession.RegisterServer()` with an existing workspace name on the server.
4. Activate the server using the method `pfcServer.Activate()`.

To Register a Server with a New Workspace

Perform the following steps to register a Windchill server with a new workspace:

1. Perform steps 1 to 4 in the preceding section to register the Windchill server with an existing workspace.
2. Use the method `pfcServerLocation.ListContexts()` to choose the required context for the server.
3. Create a new workspace with the required context using the method `pfcServer.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 Creo Parametric versions, for example, .1 or .2, and so on.*

Server Operations

After registering the Windchill server with Creo Parametric, you can start accessing the data on the Windchill servers. The Creo Parametric interaction with Windchill servers leverages the following locations:

- Commonspace (Shared folders)

- Workspace (Server-side workspace)
- Workspace local cache (Client-side workspace)
- 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.
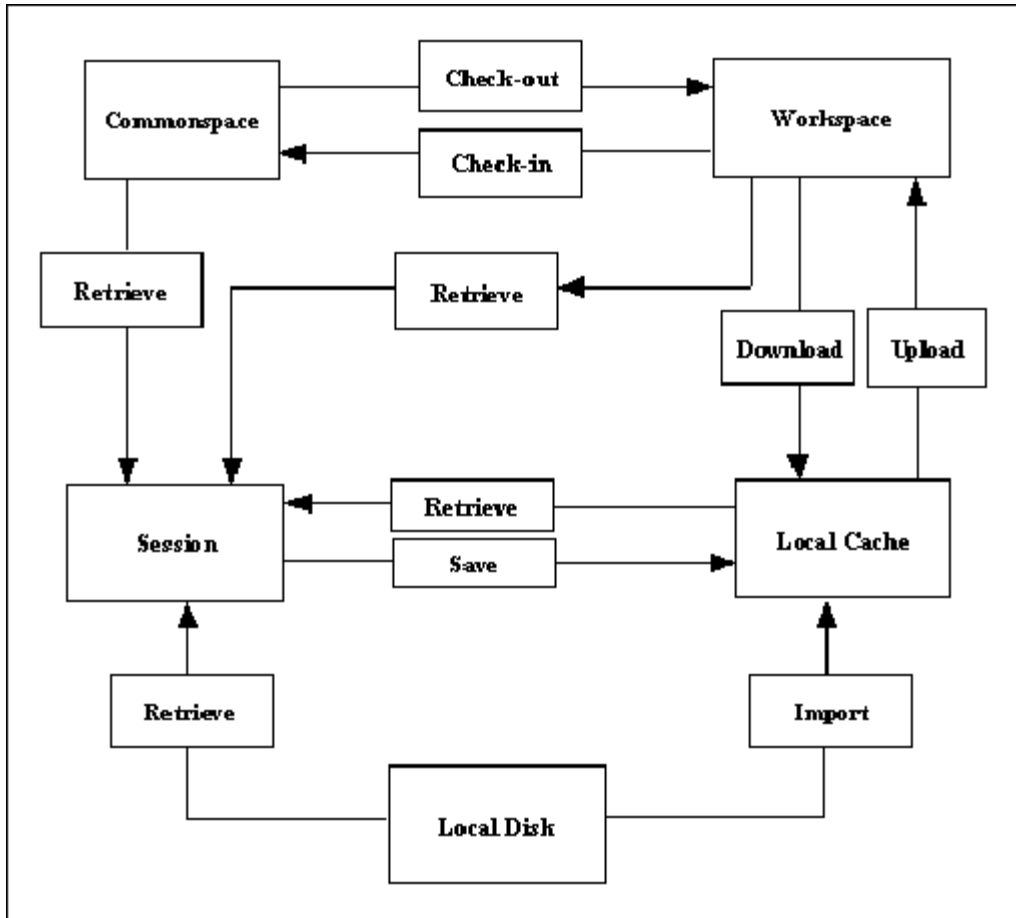
Save

Methods and Property Introduced:

- **pfcModel.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 Creo Parametric files and any other dependencies from the local workspace cache to the server-side workspace.

Methods Introduced:

- **pfcServer.UploadObjects()**
- **pfcServer.UploadObjectsWithOptions()**
- **pfcUploadOptions.Create()**

The method `pfcServer.UploadObjects()` uploads the object to the workspace. The object to be uploaded must be present in the current Creo Parametric session. You must save the object to the workspace using `pfcModel.Save()pfcBaseSession.ImportToCurrentWS()` before attempting to upload it.

The method `pfcServer.UploadObjectsWithOptions()` uploads objects to the workspace using the options specified in the `pfcUploadOptions` class. 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 `pfcUploadOptions` 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 Windchill database.

Methods and Properties Introduced:

- **pfcServer.CheckinObjects()**
- **pfcCheckinOptions.Create()**
- **pfcUploadBaseOptions.DefaultFolder**
- **pfcUploadBaseOptions.NonDefaultFolderAssignments**
- **pfcUploadBaseOptions.AutoresolveOption**
- **pfcCheckinOptions.BaselineName**
- **pfcCheckinOptions.BaselineNumber**
- **pfcCheckinOptions.BaselineLocation**
- **pfcCheckinOptions.BaselineLifecycle**
- **pfcCheckinOptions.KeepCheckedout**

The method `pfcServer.CheckinObjects()` checks in an object into the database. The object to be checked in must be present in the current Creo Parametric session. Changes made to the object are not included unless you save the object to the workspace using the method `pfcModel.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 Creo Parametric. For more details on **Auto Check-In**, refer to the online help for Creo Parametric.

Use the method `pfcCheckinOptions.Create()` to create a new `pfcCheckinOptions` 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 `pfcServerAutoresolveOption` enumerated type, 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 Windchill server and automatically checked out again for further modification.

Retrieval

Standard Web.Link provides several methods that are capable of retrieving models. When using these methods with Windchill servers, remember that these methods do not check out the object to allow modifications.

Methods Introduced:

- **pfcBaseSession.RetrieveModel()**
- **pfcBaseSession.RetrieveModelWithOpts()**
- **pfcBaseSession.OpenFile()**

The methods `pfcBaseSession.RetrieveModel()`, `pfcBaseSession.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 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 Web.Link, both operations are covered by the same set of methods.

Methods and Properties Introduced:

- **pfcServer.CheckoutObjects()**
- **pfcServer.CheckoutMultipleObjects()**
- **pfcCheckoutOptions.Create()**
- **pfcCheckoutOptions.Dependency**
- **pfcCheckoutOptions.SelectedIncludes**
- **pfcCheckoutOptions.IncludeInstances**
- **pfcCheckoutOptions.Version**
- **pfcCheckoutOptions.Download**
- **pfcCheckoutOptions.Readonly**

The method `pfcServer.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 Creo Parametric checkout rules apply. Use the method `pfcCheckoutOptions.Create()` to create a new `pfcCheckoutOptions` object.

Use the method `pfcServer.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 `pfcServerDependency` enumerated type are as follows:

  - `SERVER_DEPENDENCY_ALL`—All objects that are dependent on the selected object are checked out.

- ○ SERVER_DEPENDENCY_REQUIRED—All models required to successfully retrieve the originally selected model from the CAD application are selected for checkout.
- ○ SERVER_DEPENDENCY_NONE—None of the dependent objects are checked out.

- *IncludeInstances*—Specifies the rule for including instances from the family table during checkout. The type of instances given by the pfcServerIncludeInstances enumerated type 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 **pfc Server.Checkout Objects**	**pfcCheckoutOptions.SetDownload**	**pfcCheckoutOptions.SetReadonly**	Result
true	true	NA	Object is checked out and its content is downloaded.
true	true	NA	Object is checked out but content is not downloaded.
false	NA	true	Object is downloaded without checkout.
false	NA	false	Not supported

Undo Checkout

Method Introduced:

- **pfcServer.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 Creo Parametric session.

Import and Export

Web.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 and Properties Introduced:

- **pfcBaseSession.ExportFromCurrentWS()**
- **pfcBaseSession.ImportToCurrentWS()**
- **pfcWSImportExportMessage.Description**

- **pfcWSImportExportMessage.FileName**
- **pfcWSImportExportMessage.MessageType**
- **pfcWSImportExportMessage.Resolution**
- **pfcWSImportExportMessage.Succeeded**
- **pfcBaseSession.SetWSExportOptions()**
- **pfcWSExportOptions.Create()**
- **pfcWSExportOptions.IncludeSecondaryContent**

The method `pfcBaseSession.ExportFromCurrentWS()` exports the specified objects from the current workspace to a disk in a linked session of Creo Parametric.

The method `pfcBaseSession.ImportToCurrentWS()` imports the specified objects from a disk to the current workspace in a linked session of 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 `pfcWSImportExportMessages` object. Use the APIs listed above to access the contents of a message. The message specified by the `pfcWSImportExportMessage` 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 `pfcWSImportExportMessageType` enumerated type. 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 `pfcBaseSession.SetWSExportOptions()` sets the export options used while exporting the objects from a workspace in the form of the `pfcWSExportOptions` object. Create this object using the method `pfcWSExportOptions.Create()`. The export options are as follows:

- *Include Secondary Content*—Indicates whether or not to include secondary content while exporting the primary Creo Parametric model files. Use the property `pfcSession.WSExportOptions.SetIncludeSecondaryContent` to set this option.

File Copy

Web.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:

- **pfcBaseSession.CopyFileToWS()**
- **pfcBaseSession.CopyFileFromWS()**

Use the method `pfcBaseSession.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 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 Creo Parametric models, PTC recommends that you use methods `pfcBaseSession.ImportToCurrentWS()` and `pfcBaseSession.ExportFromCurrentWS()`, respectively, to perform the import or export operation. Methods that copy individual files do not traverse 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 Creo Parametric designated parameters, dependencies, and family table information to a PDM system while `pfcSession.BaseSession.ExportFromCurrentWS` can update exported 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-Creo Parametric files.

Server Object Status

Methods Introduced:

- **pfcServer.IsObjectCheckedOut()**
- **pfcServer.IsObjectModified()**

The methods described in this section verify the current status of the object in the workspace. The method `pfcServer.IsObjectCheckedOut()` specifies whether the object is checked out for modification.

The method `pfcServer.Server.IsObjectModified` specifies whether the object has been modified since checkout. This method returns the value `false` if newly created objects have not been uploaded.

Delete Objects

Method Introduced:

- **pfcServer.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

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.CheckinObjects()`
Checkout an object	`pfcServer.CheckoutObjects()`
Undo checkout of an object	`pfcServer.UndoCheckout()`
Upload object	`pfcServer.UploadObjects()`
Download object	`pfcServer.Server.CheckoutObjects` (with download as `true`)
Delete workspace	`pfcServer.ServerLocation.Delete Workspace`
Remove object	`pfcServer.RemoveObjects()`

These APIs throw a common exception `XToolkitCheckoutConflict` if an error is encountered during server operations. The exception description will include the details of the error condition. This description is similar to the description displayed by the 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.GetAliasedUrl()**
- **pfcBaseSession.GetModelNameFromAliasedUrl()**
- **pfcBaseSession.GetAliasFromAliasedUrl()**
- **pfcBaseSession.GetUrlFromAliasedUrl()**

The method `pfcServer.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. 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 Creo Parametric format. For example, `my_text.txt`, `prodev.dat`, `intf_file.stp`, and so on.

The method `pfcBaseSession.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 `pfcBaseSession.GetAliasFromAliasedUrl()` returns the server alias from aliased URL.

Application Conversion

This section describes how to convert Pro/Web.Link applications from Release 2001 or earlier to Wildfire embedded browser applications.

Converting Pro/Web.Link Applications from Previous Releases

To convert Pro/Web.Link applications from Release 2001 or earlier to Wildfire embedded browser applications:

1. Remove the prior Pro/Web.Link header.

   ```
   <EMBED TYPE = application/x-proconnect NAME = "mypwc" WIDTH=2 HEIGHT=2>
   <APPLET CODE = WebLink.class NAME = "pwl" WIDTH=2 HEIGHT=2> </APPLET>
   ```

2. Add a new header to the start of the script based on embedded JavaScript.

   ```
   <SCRIPT>
   function pfcCreate (className)
   {
     if (navigator.appName.indexOf ("Microsoft") != -1)
         return (new ActiveXObject ("pfc." + className));
     else
         alert ("Only IE is supported in this release");
   }

   try
   {
     document.pwl = pfcCreate("MpfcCOMGlobal").GetScript();
     document.pwlc = document.pwl.GetPWLConstants();
     document.pwlf = document.pwl.GetPWLFeatureConstants();
   }
   catch (x)
   {
   if (x.description == "pfcXNotConnectedToProE")
   {
   // The configuration option 'web_enable_javascript' was not set, or the page was loaded into a non-embedded
   browser window
   }
   else if (x.description == "Automation Server can't create object")
   {
   // The browser security does not allow embedded JavaScript
   }
   }
   ...
   </SCRIPT>
   ```

3. Replace all usages of arrays in the current Pro/Web.Link functions and return values. Search your web page for "[" and replace as follows:

   - If the array usage is populating an array for the purposes of passing it to a Pro/Web.Link method, replace the `<array>[<index>] = <value>` syntax with `<array>.Set (<index>, <value>)`.

   - If the array usage is extracting the contents of an array returned from a Pro/Web.Link method, replace the `<value> = <array>[<index>]` syntax with `<value> = <array>.Item (<index>)`.

     ### Note

     Your application can use JavaScript style arrays, but inputs and outputs accessing Pro/Web.Link methods must not be contained in standard arrays.

4. Replace all uses of named Pro/Web.Link constants in the following order:

   - Replace the string `pwl.PWL_FEAT_` with `pwlf.PWL_FEAT_`. The feature related constants are stored in the top-level object returned by `GetPWLFeatureConstants()`.

   - Replace the string `pwl.PWL` with `pwlc.PWL`. All non-feature related Pro/Web.Link constants are stored in the top-level object returned by `GetPWLConstants()`.

   ### Note

   - *The `document.pwl` object supports an `eval()` method which converts a plain string representation of a Web.Link constant to an integer value. For example, `document.pwl.eval ("PWL_FEATURE")` returns 3. The `eval()`*

method cannot be used to resolve and call functions at runtime.

- *The embedded browser version of Pro/Web.Link does not write a JavaScript trail file. Actions performed on buttons and components in web pages in the embedded browser are written into the standard Creo Parametric trail file.*

Working with Converted Functions

After converting the Pro/Web.Link applications from previous releases to the embedded browser, you can use the new "PFC" style interfaces available to Web.Link. In most situations, it will not be required to convert PWL sections of Pro/Web.Link code to PFC as many PWL methods return the required PFC object along with the other information.

For example, the method `pwlAssemblyComponentsGet` in the embedded browser returns:

- `int NumMdls`—The number of components found

- `string[] MdlNameExt`—The names and types of the components

- `int[] ComponentID`—The identifier of each component

- `pfcComponentFeats.ComponentFeatObjects`—Sequence of 'PFC' style component features

To examine the components' constraints, extract the PFC component feature objects from the last member of the return value, without rewriting other code using the original return parameters.

Summary of Technical Changes

This section describes the critical and miscellaneous technical changes in Creo Parametric 2.0 and Web.Link. It also lists the new and superseded functions for this release.

New Functions

This section describes new functions for Web.Link for Creo Parametric 2.0.

Drawing Sheet Information

New Function	Description
`pfcSheetOwner.GetSheet Info`	Returns information about the drawing sheet.

Export 3D Instructions

New Function	Description

`pfcDXF3DExportInstructions. Create()`
 	
`pfcDWG3DExportInstructions. Create()`
	Creates the instructions objects that are used to export a model to DWG and DXF formats.

Export to PDF

New Function	Description
`pfcPDFExportInstructions. ProfilePath`	Provides read and write access to the profile path in PDF export.

Facet Control Options

New Function	Description

`pfcCoordSys ExportInstructions. FacetControlOptions`
 	
`pfcTriangulation Instructions. FacetControlOptions`
	Provides read and write access to the flags that control the facet export options.

Post Regeneration Relations

New Function	Description

`pfcModel.RegeneratePost RegenerationRelations`
 	
`pfcModel.DeletePost RegenerationRelations`
	Regenerates and deletes the post regeneration relations in a model.

Step Size for Exported Files

New Function	Description
`pfcCoordSysExport Instructions.StepSize`	Provides read and write access to the step size of the exported files.

pfcTriangulation Instructions.StepSize	

Superseded Functions

This section describes the superseded functions for Web.Link for Creo Parametric 2.0.

Drawing Sheet Information

Superseded Function	New Function
pfcSheetOwner.GetSheet Data	pfcSheetOwner.GetSheet Info

Sample Applications

This section lists the sample applications provided with Web.Link.

Installing Web.Link

Web.Link is available on the same CD as Creo Parametric. When Creo Parametric is installed using PTC.SetUp, one of the optional components is `API Toolkits`. This includes Creo Parametric TOOLKIT, J-Link, Web.Link, and VB API.

If you select Web.Link, a directory called `weblink` is created under the Creo Parametric loadpoint and Web.Link is automatically installed in this directory. This directory contains all the libraries, example applications, and documentation specific to Web.Link.

Sample Applications

The Web.Link sample applications are available at: `<creo_weblink_loadpoint>/weblinkexamples`

pfcUtils

Location
`<creo_weblink_loadpoint>/weblinkexamples/jscript/pfcUtils.js`

The sample application `pfcUtils.js` provides the PFC related utilities to enable interaction between PFC objects and the web browser.

pfcComponentFeatExamples

Location
`<creo_weblink_loadpoint>/weblinkexamples/jscript/pfcComponentFeatExamples.js`
`<creo_weblink_loadpoint>/weblinkexamples/html/pfcComponentFeatExamples.html`

The sample application `pfcComponentFeatExamples` contains a single static utility method that 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.

pfcDimensionExamples

Location
`<creo_weblink_loadpoint>/weblinkexamples/jscript/pfcDimensionExamples.js`
`<creo_weblink_loadpoint>/weblinkexamples/html/pfcDimensionExamples.html`

The sample application `pfcDimensionExamples` contains a utility function that sets angular tolerances to a specified range.

pfcParameterExamples

Location
`<creo_weblink_loadpoint>/weblinkexamples/jscript/pfcParameterExamples.js`
`<creo_weblink_loadpoint>/weblinkexamples/html/pfcParameterExamples.html`

The sample application `pfcParameterExamples` contains a single static utility method that creates or updates model parameters based on the name-value pairs in the URL page.

pfcDisplayExamples

Location
<creo_weblink_loadpoint>/weblinkexamples/jscript/pfcDisplayExamples.js
<creo_weblink_loadpoint>/weblinkexamples/html/pfcDisplayExamples.html

The sample application `pfcDisplayExamples` demonstrates the use of mouse-tracking methods to draw graphics on the screen.

pfcDrawingExamples

Location
<creo_weblink_loadpoint>/weblinkexamples/jscript/pfcDrawingExamples.js
<creo_weblink_loadpoint>/weblinkexamples/html/pfcDrawingExamples.html

The sample application `pfcDrawingExamples` contains utilities that enable you to create, manipulate, and work with drawings in Creo Parametric.

pfcFamilyMemberExamples

Location
<creo_weblink_loadpoint>/weblinkexamples/jscript/pfcFamilyMemberExamples.js
<creo_weblink_loadpoint>/weblinkexamples/html/pfcFamilyMemberExamples.html

The sample application `pfcFamilyMemberExamples` contains a utility method that adds all the dimensions to a family table.

pfcImportFeatureExample

Location
<creo_weblink_loadpoint>/weblinkexamples/jscript/pfcImportFeatureExample.js
<creo_weblink_loadpoint>/weblinkexamples/html/pfcImportFeatureExample.html

The sample application `pfcImportFeatureExample` contains a utility method that returns a feature object when provided with a solid coordinate system name and an import feature's file name.

pfcInterferenceExamples

Location
<creo_weblink_loadpoint>/weblinkexamples/jscript/pfcInterferenceExamples.js
<creo_weblink_loadpoint>/weblinkexamples/html/pfcInterferenceExamples.html

The sample application `pfcInterferenceExamples` finds the interference in an assembly, highlights the interfering surfaces, and calculates the interference volume.

pfcProEArgumentsExample

Location
<creo_weblink_loadpoint>/weblinkexamples/jscript/pfcProEArgumentsExample.js
<creo_weblink_loadpoint>/weblinkexamples/html/pfcProEArgumentsExample.html

The sample application `pfcProEArgumentsExample` describes the use of the `GetProEArguments` method to access the Creo Parametric command line arguments.

pfcSelectionExamples

Location
`<creo_weblink_loadpoint>/weblinkexamples/jscript/pfcSelectionExamples.js`
`<creo_weblink_loadpoint>/weblinkexamples/html/pfcSelectionExamples.html`

The sample application `pfcSelectionExamples` contains a utility to invoke an interactive selection.

pfcSolidMassPropExample

Location
`<creo_weblink_loadpoint>/weblinkexamples/jscript/pfcSolidMassPropExample.js`
`<creo_weblink_loadpoint>/weblinkexamples/html/pfcSolidMassPropExample.html`

The sample application `pfcSolidMassPropExample` contains a utility to retrieve a MassProperty object from the provided solid model.

pfcUDFCreateExamples

Location
`<creo_weblink_loadpoint>/weblinkexamples/jscript/pfcUDFCreateExamples.js`
`<creo_weblink_loadpoint>/weblinkexamples/html/pfcUDFCreateExamples.html`

The sample application `pfcUDFCreateExamples` contains a utility that places copies of a node UDF at a particular coordinate system location in a part.

Digital Rights Management

This section describes the implications of DRM on Web.Link applications.

Introduction

Digital Rights Management (DRM) helps to control access to your intellectual property. Intellectual property could be sensitive design and engineering information that you have stored within Creo Parametric parts, assemblies, or drawings. You can control access by applying policies to these Creo Parametric objects. Such objects remain protected by the policies even after they are distributed or downloaded. Creo Parametric objects for which you have applied policies are called DRM-protected objects. For more information on the use of DRM in Creo Parametric Creo Parametric 4.0, refer to the DRM online help.

The following sections describe how Web.Link applications deal with DRM-protected objects.

Implications of DRM on Web.Link

Any Web.Link application accessing DRM-protected objects can run only in interactive Creo Parametric sessions having `COPY` permissions. As Web.Link applications can extract content from models into an unprotected format, Web.Link applications will not run in a Creo Parametric session lacking `COPY` permission.

If the user tries to open a model lacking the `COPY` permission into a session with a Web.Link application running, Creo Parametric prompts the user to spawn a new session. Also, new Web.Link applications will not be permitted to start when the Creo Parametric session lacks `COPY` permission.

If a Web.Link application tries to open a model lacking `COPY` permission from an interactive Creo Parametric session, the application throws the `pfcXToolkitNoPermission` exception.

When a Web.Link application tries to open a protected model from a non-interactive or batch mode application, the session cannot prompt for DRM authentication, instead the application throws the `pfcXToolkitAuthenticationFailure` exception.

Exception Types

Some Web.Link methods require specific permissions in order to operate on a DRM-protected object. If these methods cannot proceed due to DRM restrictions, the following exceptions are thrown:

- `pfcXToolkitNoPermission`—Thrown if the method cannot proceed due to lack of needed permissions.
- `pfcXToolkitAuthenticationFailure`—Thrown if the object cannot be opened because the policy server could not be contacted or if the user was unable to interactively login to the server.
- `pfcXToolkitUserAbort`—Thrown if the object cannot be operated upon because the user cancelled the action at some point.

The following table lists the methods along with the permission required and implications of operating on DRM-protected objects.

Methods	Permission Required	Implications
pfcSession.BaseSession.Retrieve AssemSimpRep pfcSession.BaseSession.Create DrawingFromTemplate pfcSession.BaseSession.Retrieve GraphicsSimpRep pfcSession.BaseSession.Retrieve GeomSimpRep pfcSession.BaseSession.Retrieve Model pfcSession.BaseSession.Retrieve ModelWithOpts pfcSession.BaseSession.Retrieve	OPEN	If file has OPEN and COPY permissions, model opens after authentication. Throws the pfc XToolkitNoPermission exception otherwise.

PartSimpRep pfcSession.BaseSession.Retrieve SymbolicSimpRep		
pfcModel.Rename()	OPEN	File is saved with the current policy to disk if it has COPY permission.
pfcModel.Backup() pfcModel.Copy()	SAVE	File is saved with the current policy to disk if it has SAVE and COPY permissions. Throws the pfc XToolkitNoPermission exception if model has COPY permission, but lacks SAVE permission.
pfcModel.Save()	SAVE	File is saved with the current policy to disk if it has SAVE and COPY permissions. Throws the pfc XToolkitNoPermission exception if model has COPY permission, but lacks SAVE permission. Throws the pfc XToolkitNoPermission exception if the assembly file has models with COPY permission, but lacking SAVE permission.
pfcModel.Export() for pfcPlotInstructions pfcModel.Model.Export for pfcProductViewExportInstructions (only if the input model is a drawing) pfcSession.BaseSession.Export CurrentRasterImage	PRINT	Drawing file is printed if it has PRINT permission. Throws the pfc XToolkitNoPermission exception if drawing file lacks PRINT permission.

Copy Permission to Interactively Open Models

When the user tries to open protected content lacking COPY permission through the Creo Parametric user interface with a Web.Link application running in the same session:

1. Creo Parametric checks for the authentication credentials through the user interface, if they are not already established.

2. If the user has permission to open the file, Creo Parametric checks if the permission level includes COPY. If the level includes COPY, Creo Parametric opens the file.

3. If COPY permission is not included, the following message is displayed:
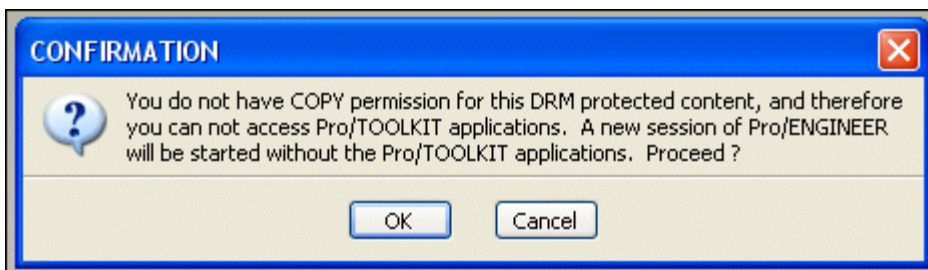
4. If the user clicks **Cancel**, the file is not opened in the current Creo Parametric session and no new session is spawned.

5. If the user clicks **OK**, an additional session of Creo Parametric is spawned which does not permit any Web.Link application. Web.Link applications set to automatically start by Creo Parametric will not be started. Asynchronous applications will be unable to connect to this session.

6. The new session of Creo Parametric is automatically authenticated with the same session credentials as were used in the previous session.

7. The model that Creo Parametric was trying to load in the previous session is loaded in this session.

8. Other models already open in the previous session will not be loaded in the new session.

9. Session settings from the previous session will not be carried into the new session.

10. The new session will be granted the licenses currently used by the previous session. This means that the next time the user tries to do something in the previous session, Creo Parametric must obtain a new license from the license server. If the license is not available, the action will be blocked with an appropriate message.

Additional DRM Implications

- The method `pfcModel.CheckIsSaveAllowed()` returns `false` if prevented from save by DRM restrictions.

- The method `pfcBaseSession.CopyFileToWS()` is designed to fail and throw the `pfcXToolkitNoPermission` exception if passed a DRM-protected Creo Parametric model file.

- The method `pfcBaseSession.ImportToCurrentWS()` reports a conflict in its output text file and does not copy a DRM-protected Creo Parametric model file to the Workspace.

- While erasing an active Creo Parametric model with DRM restrictions, the methods `pfcModel.Erase()` and `pfcModel.EraseWithDependencies()` do not clear the data in the memory until the control returns to Creo Parametric from the Creo Parametric TOOLKIT application. Thus, the Creo Parametric session permissions may also not be cleared immediately after these methods return.

Geometry Traversal

This section 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.
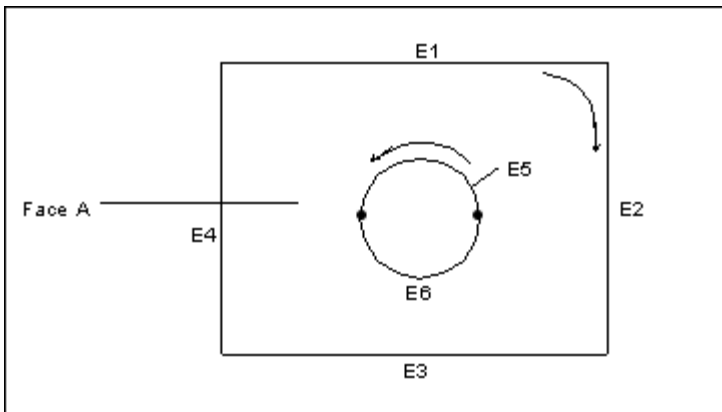
Example 1
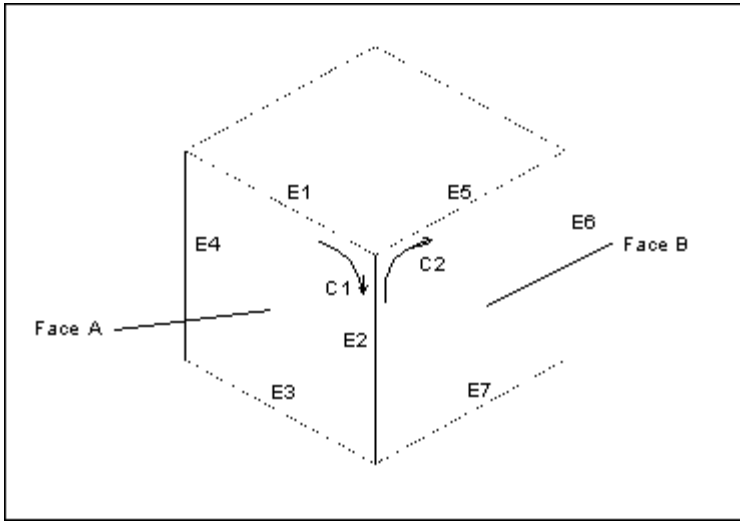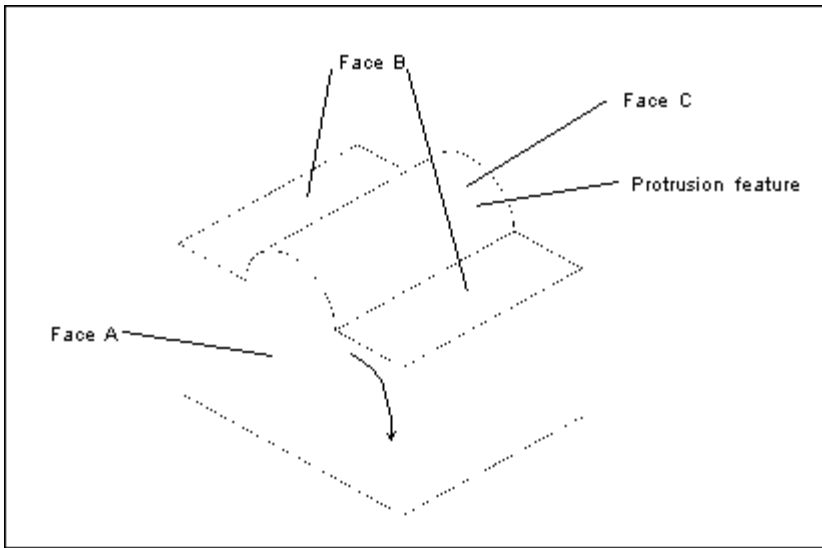

This part has 6 faces.

- 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.
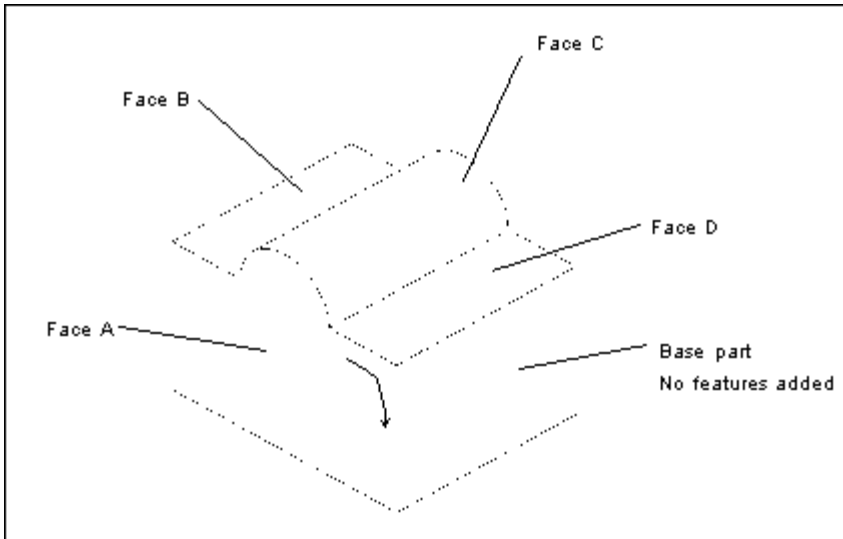
Example 2

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
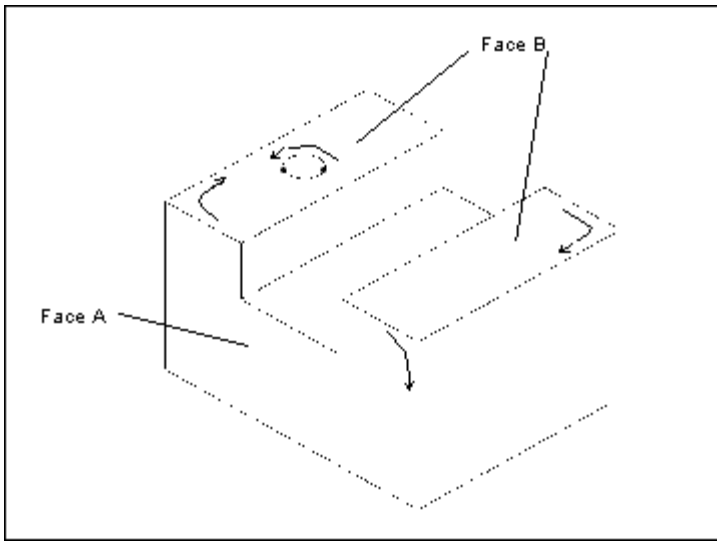
This part was extruded from a rectangular cross section. The slot and hole features were added later.

- Face A has 1 contour and 8 edges.
- Face B has 3 contours and 10 edges.

Geometry Representations

This section describes the geometry representations of the data used by Web.Link.

Surface Parameterization

A surface in 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:

- Cone
- Coons Patch
- Cylinder
- Cylindrical Spline Surface
- Fillet Surface
- General Surface of Revolution
- NURBS
- Plane
- Ruled Surface
- Spline Surface
- Tabulated Cylinder
- Torus

Plane
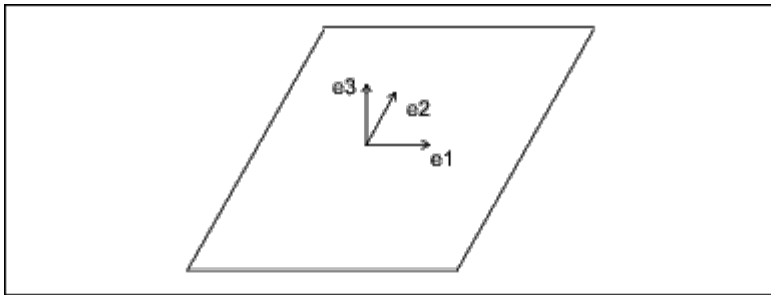
The plane entity consists of two perpendicular unit vectors (`e1` and `e2`), the normal to the plane (`e3`), and the origin of the plane.
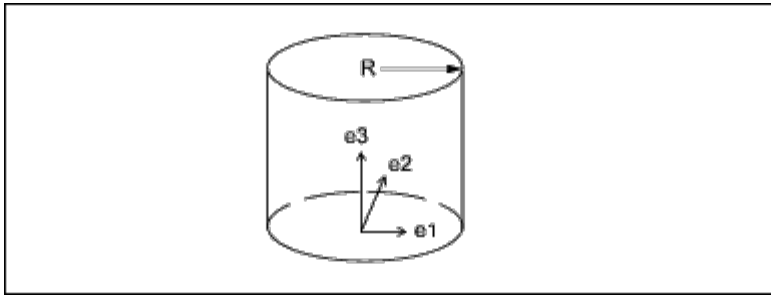
Data Format:

```
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
```
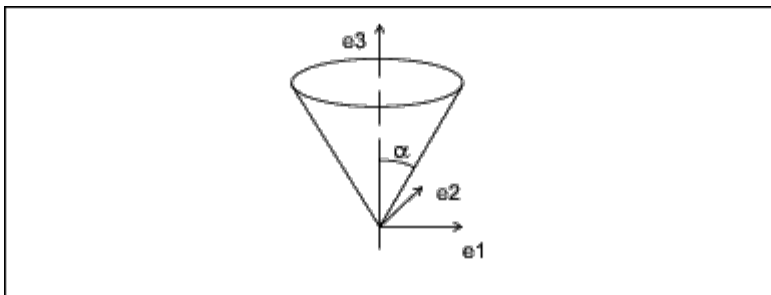
Parameterization:

```
(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)`.
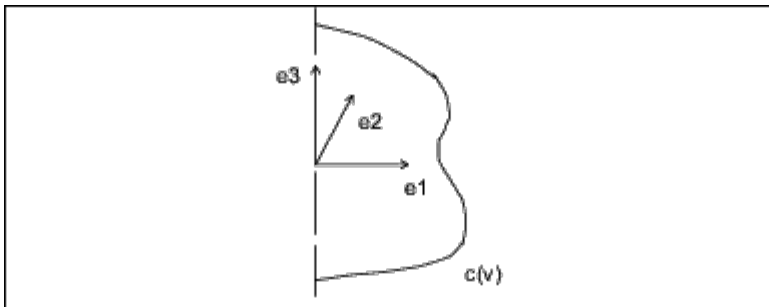
Data Format:

```
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
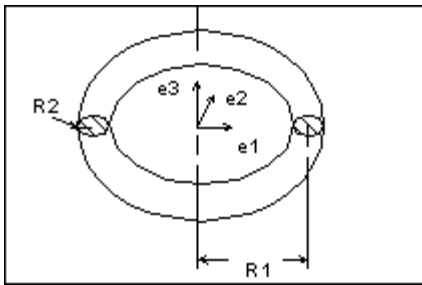
The generating curve of a torus is an arc of radius R2 with its center at a distance R1 from the origin. The starting point of the generating arc is located at a distance R1 + R2 from the origin, in the direction of the first vector of the local coordinate system. The radial distance of a point on the 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.
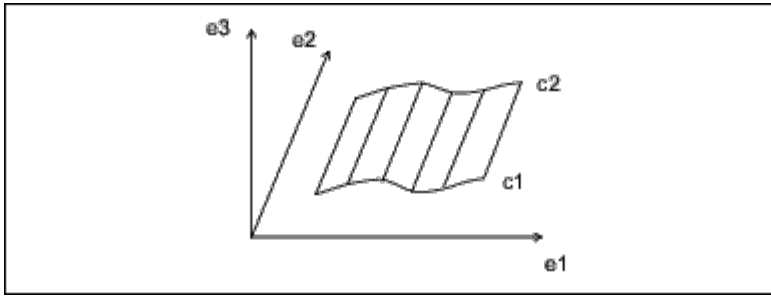
Data Format:

```
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.
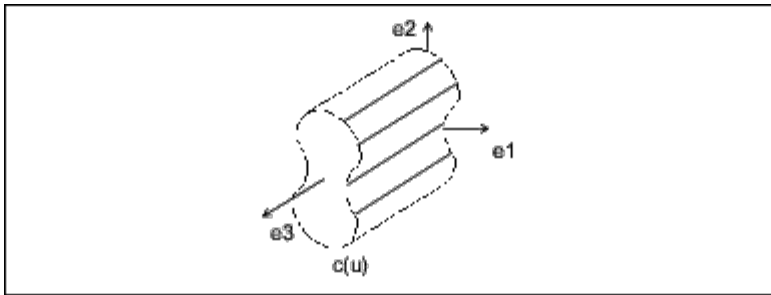
Data Format:

```
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.
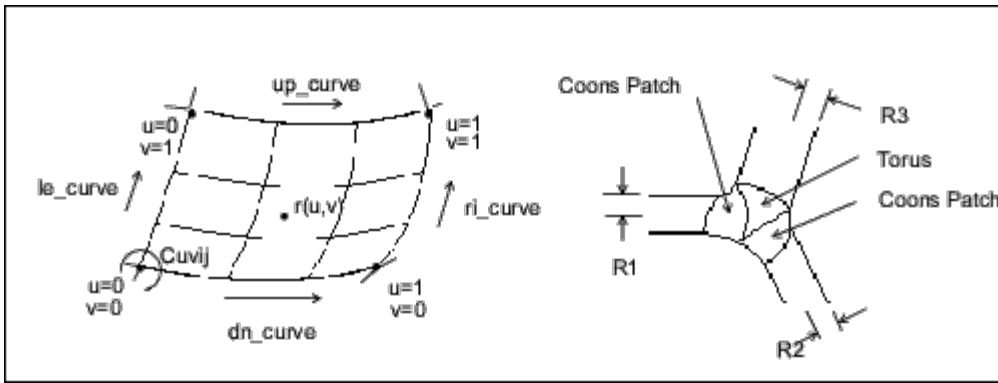
Data Format:

```
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
```
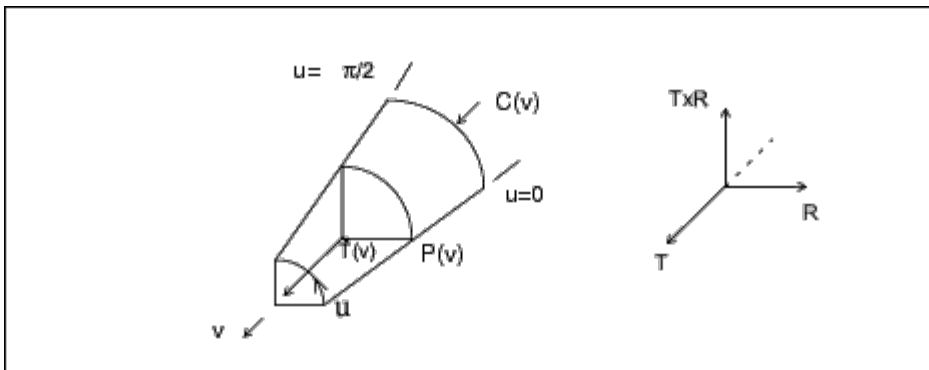
Coons Patch

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.
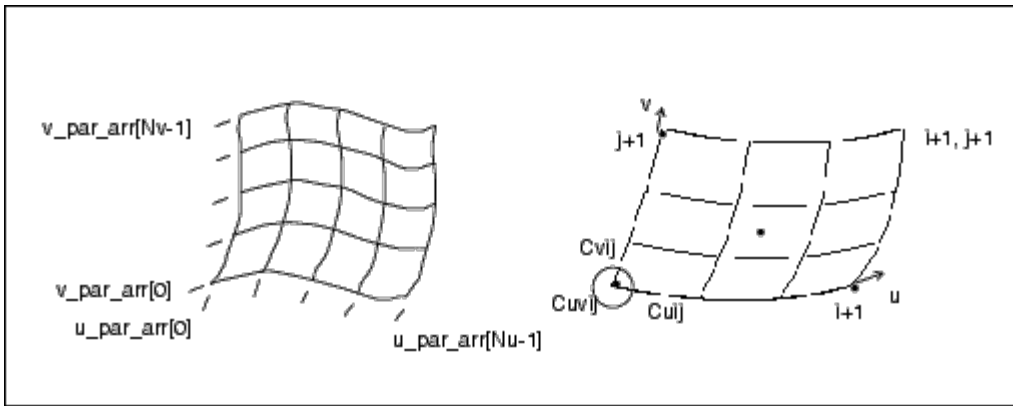
Data Format:

```
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
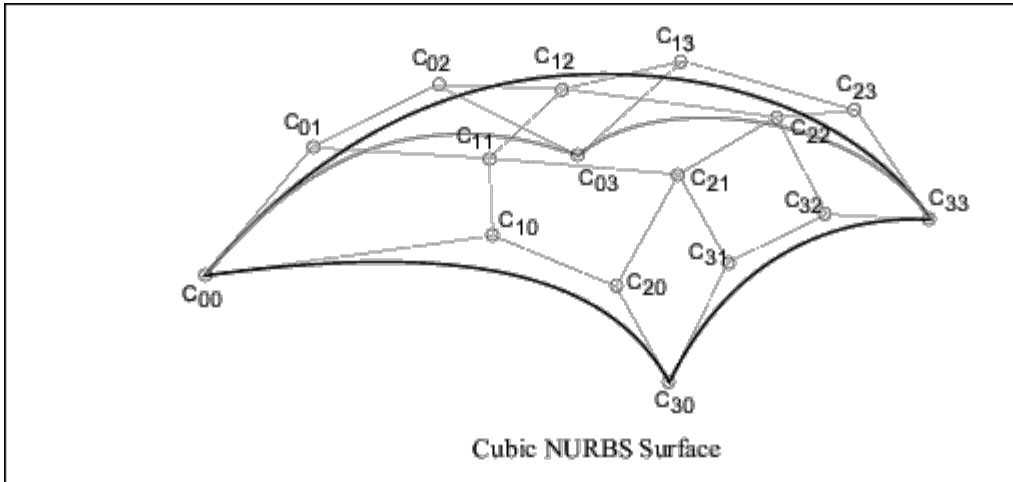
Data Format:

```
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:

$$R(u, v) = \frac{\displaystyle\sum_{i=0}^{N1}\sum_{j=0}^{N2} C_{i,j} \times B_{i,k}(u) \times B_{j,l}(v)}{\displaystyle\sum_{i=0}^{N1}\sum_{j=0}^{N2} w_{i,j} \times B_{i,k}(u) \times B_{j,l}(v)}$$

```
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.
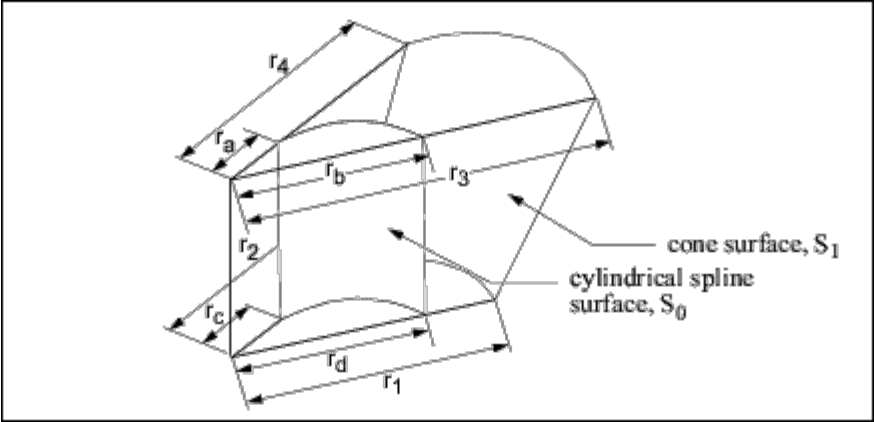
Data Format:

```
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).

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 ? r3).

If a replacement cannot be done (such as for the surface s0 in the figure (ra ? rb or rc ? 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
- Line
- NURBS
- Spline

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:
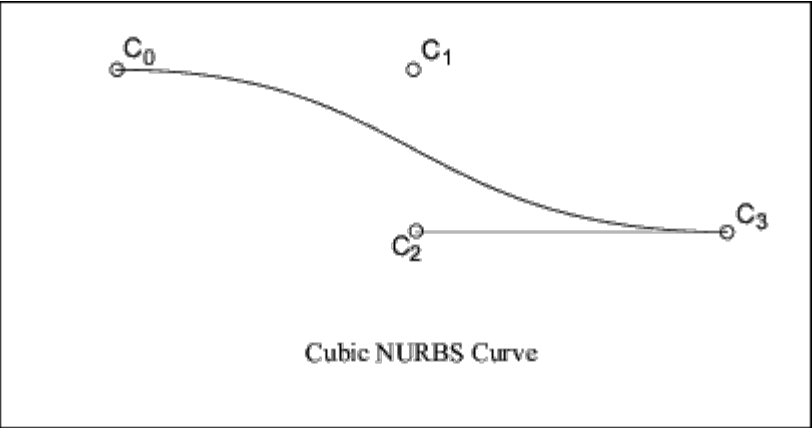
```
par_arr[i] < t' < par_arr[i+1]
```

(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.

Cubic NURBS Curve

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:

$$R(t) = \frac{\sum\limits_{i=0}^{N} C_i \times B_{i,\,k}(t)}{\sum\limits_{i=0}^{N} w_i \times B_{i,\,k}(t)}$$

```
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.