



Business-friendly Solutions

ITC Infotech India Ltd  
#18 Banaswadi Main Road  
Bangalore, India - 560005  
+91-80-22988601  
[www.itcinfotech.com](http://www.itcinfotech.com)

## **Windchill Reference Designator Loader For Existing Product Structure**

*Author: JGS.Raj*

**Contents**

---

<b>Executive Summary</b>	<b>3</b>
<b>Business Challenge</b>	<b>3</b>
<b>I3L – ISC Solution</b>	<b>3</b>
<b>Technical Specifications</b>	<b>4</b>
<b>Benefits</b>	<b>8</b>
<b>Target Market</b>	<b>8</b>
<b>Summary</b>	<b>8</b>

## Executive Summary

People in product based company would know the terminology “Occurrence”. A logical representation of identical part / object in multiple locations in the same sub assembly is termed as occurrence in engineering design processes.

Location specifications of occurrence part / object are managed distinctly among all Modeling solution leaders. PLM software too maintained the uniqueness among them on occurrence management.

Windchill, a PLM market leader uses “Reference Designator” to set the location information for occurrence parts / objects. This would control the display logic while loading the occurrence structure on the UI. The major risk / issue that could affect manufacturing process due to reference designator is

- Having void or wrong reference designator existing in structure, would add confusion at assembly shop floor as MBOM would not contain location details for an occurrence part.

This issue / risk could be controlled by loading correct reference designator values to an existing product structure.

## Business Challenge

A leading Windchill user from North America faced business critical challenge due to the existence of erratic reference designators in their system.

- Customer had recently migrated their CAD data from Pro/I to Windchill. The Meta structure created during migration, either lost relevant reference designator information or not provided.
- New structures created at Windchill system also not provided with correct reference designator or given void. This could be due to lack of adoption.
- Their MBOM doesn’t carry reference designator information for shop floor people.
- Manual effort to correct structures with relevant reference designator value is laborious.
- And OOTB Windchill doesn’t have loader application / tool to load reference designator on existing product structure. OOTB provides loader to load reference designator along with structure load into Windchill system

## I3L – ISC Solution

India Solution Center proposed following complete solution to counter the risk posted due to incorrect reference designator in the system

- ❖ Create new reference designator data set pertaining to the occurrence structure existing in Windchill system.

- ❖ Design & develop a application, which could load correct reference designator value from the data set to existing occurrence structure
- ❖ Execute the application to load data

This solution would achieve following, minimize the risk due to incorrect reference designator values and reduce time for data correction, which would enable the organization to achieve targeted ROI on time.

### Technical Specifications

This section describes the solution in detail.

**Data Preparation:** Data can be prepared in either csv or xml formats. The custom application is designed such a way to handle data with out error. Data file needs to contain following information

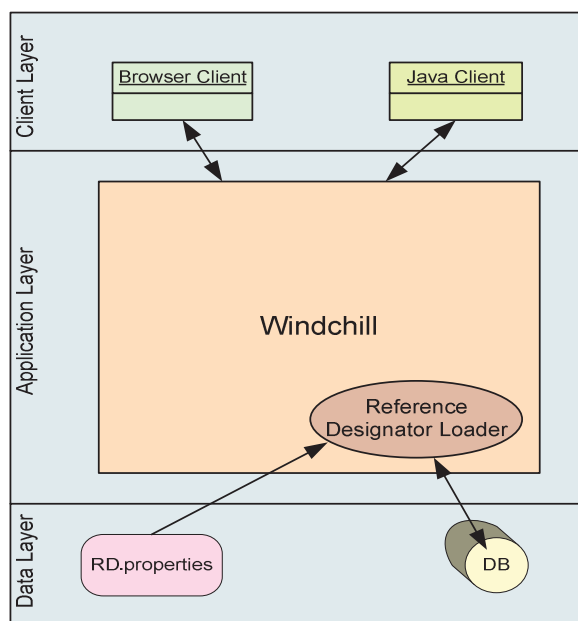
- assemblyPartNumber
- assemblyPartVersion
- assemblyPartIteration
- assemblyPartView
- constituentPartNumber
- occurrenceLocation
- lineNumber
- referenceDesignator

If it is csv file, then the delimiter used for multiple refernceDesignator values needs to be configured through properties file for better scalability

**Application Design:** Reference Designator loader application is designed in such a way, a separate application would be executed in

Windchill method server JVM. See the logical diagram of the application, it contains following components

1. Clients
2. Reference Designator Loader
3. Data Sources



**1. Clients:** Reference designator loader application design is visualized to have two different types of clients

- ❖ Browser Client
- ❖ Java Client

Web browser application developed and would be launched from specific location (Preferably Business Administrator page). This pop up window would ask for confirmation to further proceed to change Reference Designator of an existing product structure, on click of ok, it would call appropriate method to execute the load. This would capture error message and display back to the user if any.

Java client is an ordinary java class; the main method of this client class would call

appropriate method to execute the Reference Designator load

## 2. Reference Designator Loader:

The server side implementation is visualized as component based approach and provides high scalability. This application comprises of many java classes and the details of those classes are discussed in details below along with graphical representations of the classes

RDLDataLoader
-parentParts -childrenParts
+RDLDataLoader() +getParentParts() +getChildrenParts() -loadDataSet() -loadFileContents() -loadCSVFileContents() -loadXMLFileContents() -processString() -getReferenceDesignators()

RDLDataLoader class is implemented to parse the data from data file (s) (either csv or xml) and fill those into relevant Hashtables.

**parentParts:-** This is a Hashtable, meant to hold all parts which are identified as parents (children parts attached under these parts would get modified with new reference designators) . Part number would be the key and vector object containing all other details about parent part would be value.

**childrenParts:-** This Hashtable is to have all information about the children parts which are all needs to be get changed with new reference designators. The parent part number would be a key and a Hashtable of

children part details. This inner Hashtable would have child part number as key and Hashtable of reference designators. This further inner Hashtable would have line number as key and Vector of reference designators as values.

- getParentParts & getChildrenParts methods would return respective Hashtable object.
- RDLDataLoader, would be a constructor. This would read the file path of CSV or XML file and reference designator delimiter (if it is csv file) from properties file. This would call loadDataSet method by passing these file path and delimiter value as arguments
- loadFilecontents method would be called from loadDataSet method. A file object is a argument. This method would validate whether the file object is XML or CSV or Directory. If directory this would process all files under the directory. This would call appropriate methods to handle different file types.
- loadCSVFileContents is the method responsible to parse data from csv file and fill in respective Hashtable objects. This would call processString & getReferenceDesignators methods
- loadXMLFileContents would parse values from xml file and load Hashtable objects appropriately. This would call methods from another class "RDLXMLParser".

RDLXMLParser
-parents -children
+parse() -clearAllData() +getParents() +getChildren()

RDLXMLParser class is implemented for XML file parsing. Both attributes and methods in this class are static. Both attributes are Hashtable.

- “parse” method would be called from outside passing file object as argument. This method would use standard XML parsing methodology to parse the file and fill data into both Hashtable objects appropriately.
- clearAllData method would clear the contents of both Hashtables.
- getParents & getChildren methods would return relevant Hashtable object.

After processing data from the input files, the data (Reference Designator) loading is taken care by RDLProcessor class. This application is capable to both update existing reference designators and add new one in the existing product structure without changing its iteration or version. All of these logics are built in this class

RDLProcessor
+processForRDLoad() -getParentPart() -updateReferenceDesignators() -insertUsageLink() -simpleSaveUsesOccurrenceAndData() -updateUsageLink()

Following are more details about the implementation.

- ❖ processForRDLoad, method would take two hashtables as arguments. One is parentParts

& another one is childrenParts from RDLDataLoader class.

- ❖ This scans through all parents from the Hashtable and start updating the reference designator values to their children parts in the assembly if it is already existing, or inserting new reference designator.
- ❖ getParentPart method is called by the previous method to return exact object from the Windchill system.
- ❖ updateReferenceDesignator method would be called by passing parent part object and childrenParts Hashtable object.
- ❖ This would identify, either its is update process or new insert process and would call appropriate methods to act upon
- ❖ insertUsageLink method to perform inserting new reference designator into the existing product structure. This would call simpleSaveUsesOccurrenceAndData method.
- ❖ simpleSaveUsesOccurrenceAndData inserts the newly created PartUsesOccurrence object into the database.
- ❖ updateUsageLink method would replace the old value with new reference designator values

RDLErrorLog
-errorContainer +getErrorMessageString() +getErrorMessage() +setErrorMessage() +getErrorMessageTable() +createErrorLogFile() +isErrorMessageExist() +clearAllMessages()

RDLErrorLog class is implemented to capture and accumulate all issues and errors that the application faced during the run time for further analysis.

- ❖ errorContainer is a Hashtable object to accumulate all error message experienced during the run time.
- ❖ getErrorMessage & setErrorMessage are two methods to get and set values to the Hashtable object
- ❖ getErrorMessageString would generate complete string from stored information and return as String. This is ideal to call from Java client to display error message in the Java console.
- ❖ getErrorMessageTable method would generate HTML string containing all error messages formatted in table form. This could be useful for browser based client call.
- ❖ createErrorLogFile, method would create error log file and place in the server machine OS Folder.
- ❖ isErrorMessagesExist would check whether any error message is stored, if yes return true otherwise false. clearAllMessages is to indent to clean the content of Hashtable object.

RDLService
+loadReferenceDesignator()

This is an rmi class, is extended from RemoteAccess class and implemented with Serializable Interface. This would help the application to transfer the load into MethodServer, hence the

performance & stability of Windchill environment would be maintained.

This class would have only one method, in which, object of RDLDataLoader class would be created, then the parentParts & childrenParts of that class member would be obtained and passed to a method “processForRDLoad” from RDLProcessor.

RDLHelper
+loadReferenceDesignator()
+getErrorMessageString()
+isLoadSuccessful()
+getErrorMessageTable()

RDLHelper class is the actual interface class between client applications and server side implementation. Details are given below

- ✚ All four methods are wrapper methods, those calls relevant methods from different sources
- ✚ loadReferenceDesignator method would invoke remote call for the same method existing in RDLService class.
- ✚ getErrorMessageString and getErrorMessageTable calls same methods from RDLErrorLog class
- ✚ isLoadSuccessful method calls “isErrorMessagesExist” method from RDLErrorLog class.

**3. Data Sources:** There are two data sources identified for this application

- A file named “rd.properties”, which contains full file path of XML or CSV file location and delimiter string for reference designator values
- A Windchill data base which gets updated as reference designator load progress. Custom classes developed for this application uses OOTB APIs to update Windchill data base data,

hence the data integrity is maintained in the system

## Benefits

The solution described in this document posts many benefits to the Windchill users. See some of them below

1. Reference Designators shall be newly loaded into the PDM system as and when the correction required, which provides flexibility on working
2. This update (Reference Designator load) is not going to change iterations or versions of existing WTParts in the system, which avoids all confusion related to this scenario
3. Because of this capability, even baselined or instanced structures can also be modified with reference designators if required.
4. As this process is not taking more time (Only data preparation phase would take little longer time), the business critical process would be completed with in no time.
5. ROI would be achieved on time as the delay of manufacturing due to improper occurrence information is completely eliminated by this solution
6. This application is highly scalable as most of the logic developed for this application is using simple Java code. Only on Windchill interaction, used Windchill

APIs. Minimal changes in very minimal places would enable this application run on any version of Windchill environment.

## Target Market

Due to the scalability of this solution and criticality of changing Reference Designator values in the existing product structure

All most all Windchill users (irrespective of Windchill version they use) having occurrence structure in their system would be beneficiary by this application

## Summary

The solution described in this document would add following benefits.

- ❖ Customer can modify their BOM data as and when required with out much pain
- ❖ Error free operation
- ❖ More flexible
- ❖ Best scalability