EDV Software Service GmbH & Co KG
Bahnhofstrasse 8, 9500 Villach ÖSTERREICH
**Tel.** 0043 4242 27876
www.ax3000-group.at

# IFC ROUNDTRIP

## BRICSCAD PLUGIN

*Anforderungen, Entwicklung, Installation, Anmerkungen*

Februar 2021, Markus Griesser

# INHALTSVERZEICHNIS

# INHALTSVERZEICHNIS

# 1. Projektgrundlagen

## FORSCHUNGSPROJEKT IFC ROUNDTRIP

Grundlage der Entwicklung ist das Forschungsprojekt „IFC Roundtrip“ des BMLV Bundesministerium für Landesverteidigung. Als erstes Forschungsziel wird der softwareübergreifende Datenaustausch des digitalen Gebäudemodells zwischen den Softwareprodukten Autodesk AutoCAD Architecture und GRAPHISOFT ARCHICAD sowie Bricsys BricsCAD auf Basis der IFC-Schnittstelle definiert. Dies erfordert:

- einen softwareübergreifenden Datenaustausch ohne Informationsverlust im Lebenszyklus und die Übernahme der Bestandsdaten aus Autodesk AutoCAD Architecture in GRAPHISOFT ARCHICAD und BricsCAD. IFC Roundtrip und Plangrafiken Zwischenbericht Mai 2020 6/34
- Erstellung, Übertragung und Wiedergabe der geometrischen Repräsentation der Bauteile des digitalen Gebäudeinformationsmodells von Autodesk AutoCAD Architecture nach GRAPHISOFT ARCHICAD und BricsCAD.
- einen softwareübergreifenden Datenaustausch ohne Informationsverlust im Lebenszyklus und die Übernahme der Bestandsdaten aus Autodesk AutoCAD Architecture in Bricsys BricsCAD.
- Erstellung, Übertragung und Wiedergabe der geometrischen Repräsentation der Bauteile des digitalen Gebäudeinformationsmodells von Autodesk AutoCAD Architecture nach Bricsys BricsCAD.

## ANFORDERUNGEN

Es wurde die Anforderung gestellt, alle Gebäudetaten aus bestehenden Plänen Anbieter-unabhängig zwischen verschiedenen Systemen auszutauschen. Dazu wurde das allgemeine Austauschformat "IFC" gewählt.

Obwohl der IFC-Standard den Austausch von 2D-Daten prinzipiell ermöglicht, ist eine Umsetzung in den meisten CAD-Systemen nicht oder nur sehr eingeschränkt gegeben. Da die Plandarstellung aber einen wesentlichen Anteil an einem Projekt darstellt, soll im Umfang dieses Roundtrips die Möglichkeit des Imports und Exports evaluiert und auch erweitert werden.

**In diesem Dokument wird der Import/Export von IFC in Bricsys BricsCAD erläutert.**

# 2. BricsCAD

## GRUNDLAGEN

Als Basis für die Entwicklung wurde BricsCAD V21 verwendet. Dieses verfügt mit aktivierter BIM-Lizenz über einen IFC-Import, der Modelldaten des Gebäudes (Wände, Fenster, Türen, etc.) berücksichtigt. Planinformationen wie Linien, Texte, Layouts werden nicht bzw. nur sehr rudimentär importiert. Am Beispiel des vom ÖBH zur Verfügung gestellten DWGs 7E00-011-EG-B sollen diese Punkte veranschaulicht werden:

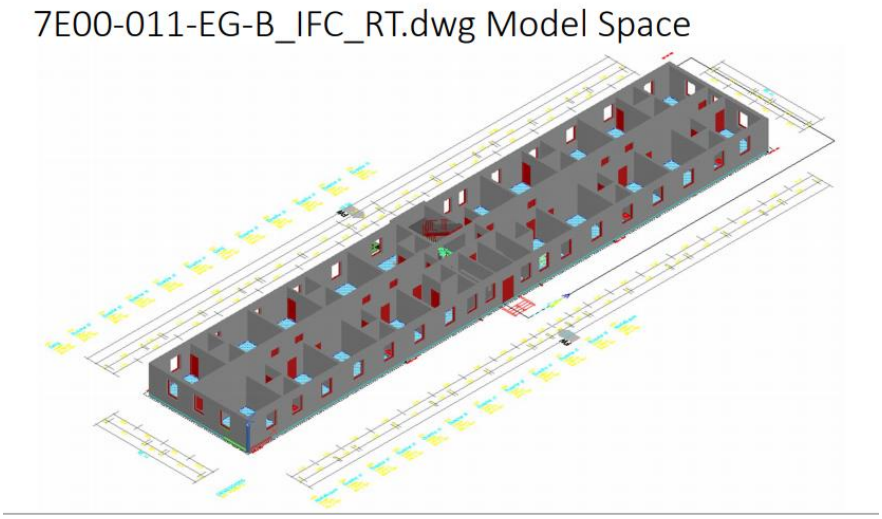


*Abbildung 1: Original-Datei in AutoCAD Architecture*

Direkt in BricsCAD V21 über IFC importiert, ergibt sich folgendes:

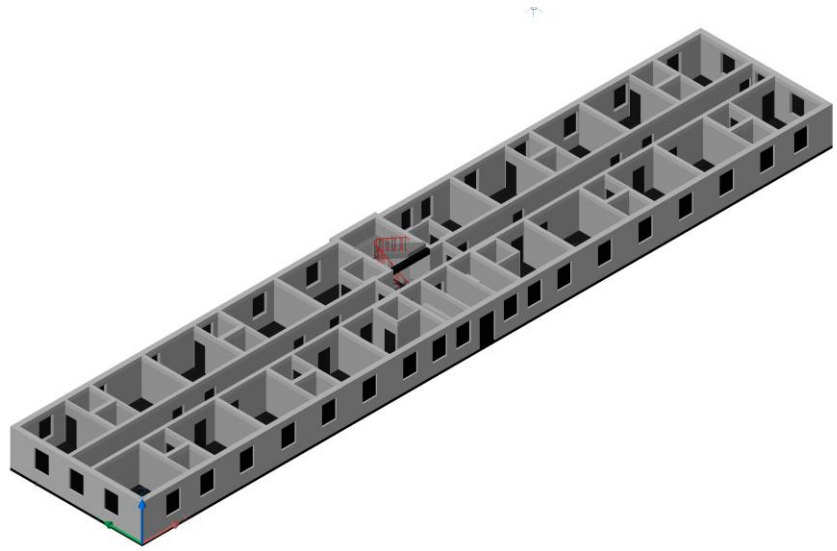

*Abbildung 2: Import in BricsCAD V21 BIM*

Wie leicht zu erkennen ist, wurden zwar alle Gebäudedaten übernommen, allerdings ohne die zusätzlich vorhandenen Beschriftungen, Bemaßungen, Symbole und Schraffuren.

Daten des Papierbereichs werden in BricsCAD über IFC nicht importiert oder exportiert. Als Beispiel soll auch hier ein Layout des Original-Plans dienen:

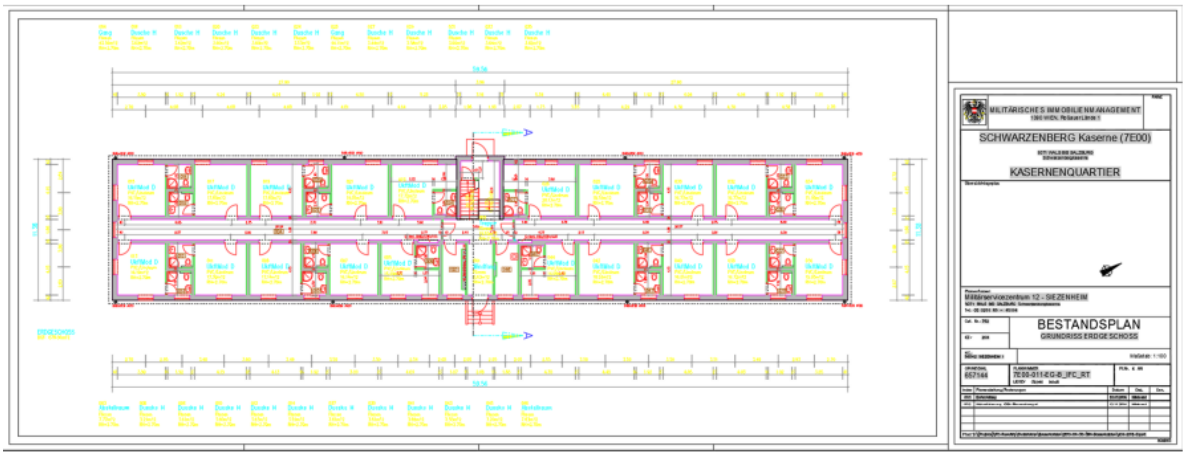


*Abbildung 3: Layout "Bestandsplan 1:100"*

Für den Roundtrip ist es notwendig, diesen Import auch um Plandaten zu erweitern. Die Darstellung dieses Plans wird auch als Papierbereich bezeichnet. Dieser besteht aus einem oder mehreren Viewports (die unterschiedliche Ansichten auf das Modell bieten) und einem Plankopf, der wiederrum aus 2D-Elementen besteht, wie sie auch im Modellbereich vorhanden sind.

Im Unterschied zu 2D-Elementen aus dem Modell, sind jene des Papierbereichs nur exakt auf dem Papierbereich zu sehen, auf dem sie erstellt wurden.

Ein erneuter Import mit aktiviertem Plugin führt zu folgendem Ergebnis:

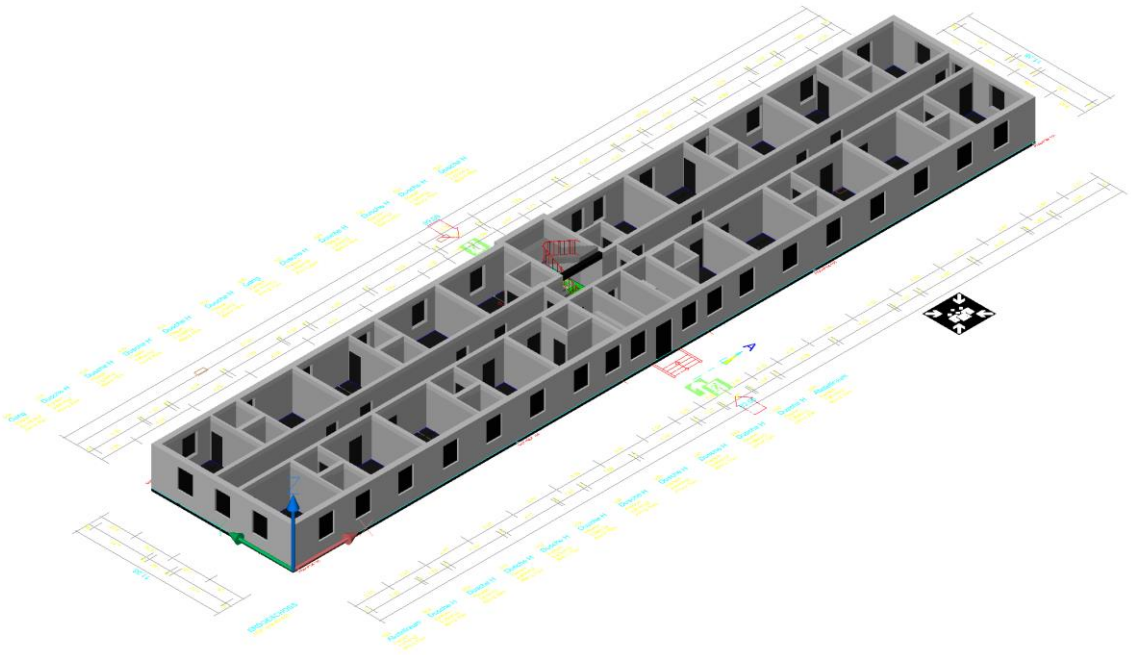

*Abbildung 4: Import mit Plugin*

## ERWEITERUNG IFC IMPORT

BricsCAD bietet für den IFC Import und Export die Möglichkeit der Erweiterung an. Als Grundlage dient ein Reactor, der Plugins während des Imports eines IFC-Elements benachrichtigt. Das Plugin kann dann entscheiden, ob es den Import selbst durchführt, oder BricsCAD die Arbeit überlässt. Diese Möglichkeit wurde von uns aufgegriffen und als Basis für den Roundtrip verwendet. Planelemente wie z.B. Texte und Linien werden über `IfcAnnotation` exportiert.

```
bool AxIfcImportReactorInstance::onIfcProduct(const Ice::IfcApi::Entity& product, bool
isParent, const Ice::IfcApi::Entity& parentProduct)
{
	auto name = static_cast<const char*>(*product.isA());
	if (product.isKindOf(IfcAnnotation)) {
		// Code
	}
	return false;
}
```

An diesem Punkt setzt der Plugin-Code ein und erzeugt das Element. Eine detaillierte Beschreibung der Vorgehensweise und Abläufe findet sich in späteren Kapiteln.

# 3. Installation

## VORAUSSETZUNG

Ein Setup wurde nicht erstellt, weshalb die Programmdateien und Daten manuell kopiert und eingerichtet werden müssen.

## SOFTWAREVORAUSSETZUNGEN

| BESCHREIBUNG | |
|---|---|
| **BricsCAD V21.1.01+** | |
| **VC++ Redistributable Setup 2015-2019** | https://aka.ms/vs/16/release/vc_redist.x64.exe |

Der aktuell Stand des Plugins befindet sich hier: TODO

Alle Daten sind in einer ZIP-Datei zusammengefasst. Um das Plugin in BricsCAD V21 zu installieren müssen die folgenden Schritte beachtet werden:

1. ZIP-Datei in einen beliebigen Ordner entpacken
2. Eintrag in der Registry erzeugen, der auf den oben erzeugten Ordner zeigt
3. BricsCAD starten und den Befehl APPLOAD eintippen
4. Im erscheinenden Dialog die AxIFCExtenderPlugin.arx auswählen
5. Auf „Laden“ klicken

Zu 1) Der Ordner muss nach dem Entpacken der ZIP-Datei so aussehen:

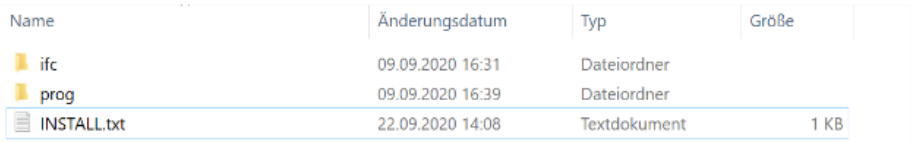


*Abbildung 5: Installationsverzeichnis*

Die Programmdateien sind von der Firma ESS EDV Software Service GmbH & Co KG digital signiert. Das kann über die Eigenschaften der Datei kontrolliert werden. Eine Signatur garantiert den Hersteller und dass seit der Signierung keine Änderungen an den Dateien durchgeführt wurden.

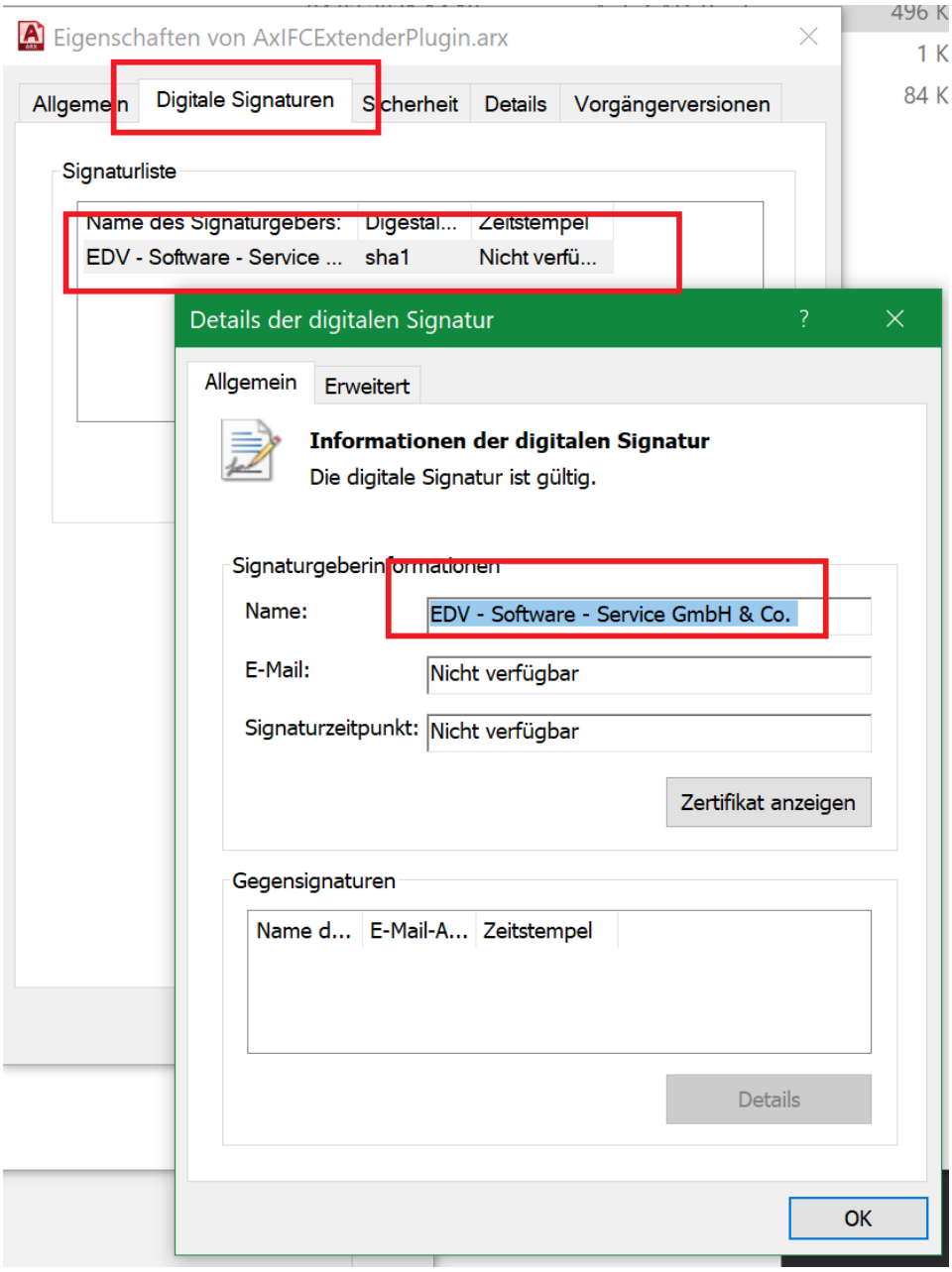


*Abbildung 6: Digitale Signatur des Plugins*

Zu 2)

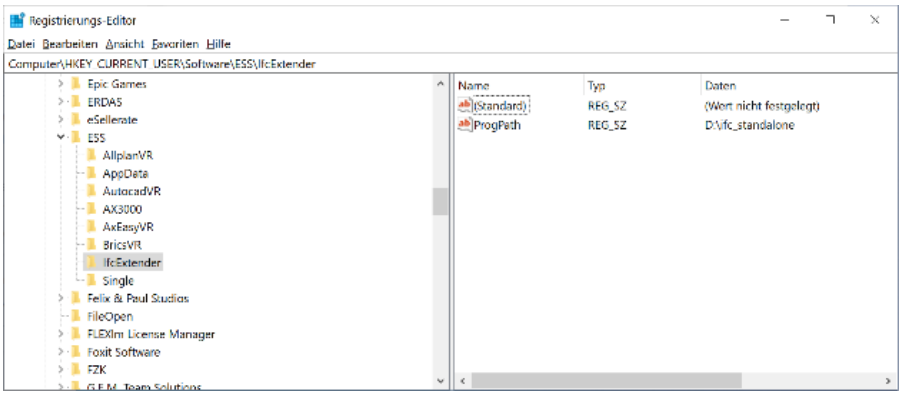


*Abbildung 7: Eintrag in der Registry*

Das Plugin ist nun geladen und beim Import bzw. Export von IFC-Dateien werden nun auch `IfcAnnotation`, Planlayout und Viewports beachtet. Eine IFC-Datei kann in BricsCAD entweder über das Datei-Menü oder per Drag&Drop importiert werden. Hierbei ist zu beachten, dass in beiden Fällen die aktive Vorlagendatei als Grundlage für den Import verwendet wird. Das betrifft eventuell vorhandene Schraffuren, Linienstile, Textstile wie auch die eingestellte Zeichnungseinheit.

## PROBLEMBEHEBUNG

Sollte es beim Laden des Plugins zu Fehlermeldungen kommen, bitte die oben aufgeführten Schritte kontrollieren. Wurde alles korrekt durchgeführt, kann das Eintragen des prog-Verzeichnisses in die Suchpfade von BricsCAD abhilfe schaffen. Dazu die Einstellungen von BricsCAD öffnen und „Support File Search Path“ oder „Suchpfad für Support-Dateien“ ins Suchfeld eintippen. Anschließend den Pfad wie im Bild gezeigt hinzufügen.

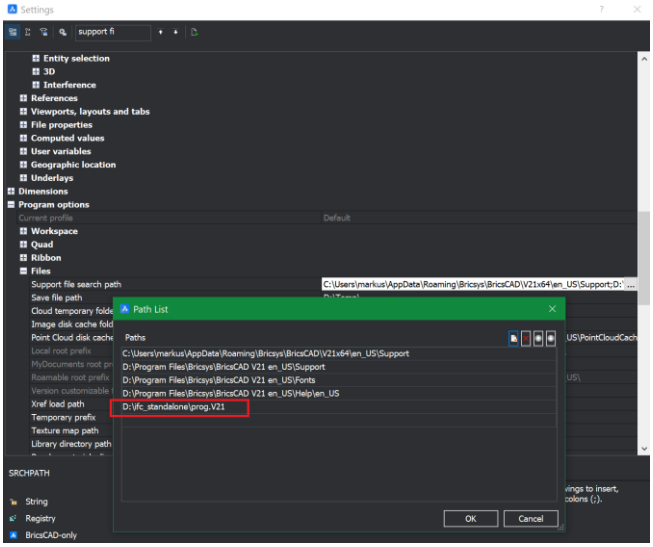


*Abbildung 8: Prog-Pfad hinzufügen*

## BEISPIEL-IMPORT

Am oben genannten Beispiel vom ÖBH soll das Laden und der Import dargestellt werden.

1. BricsCAD starten und Befehl APPLOAD eintippen

   Mit dem „Plus“-Symbol die ARX-Datei aus dem angelegten Ordner wählen

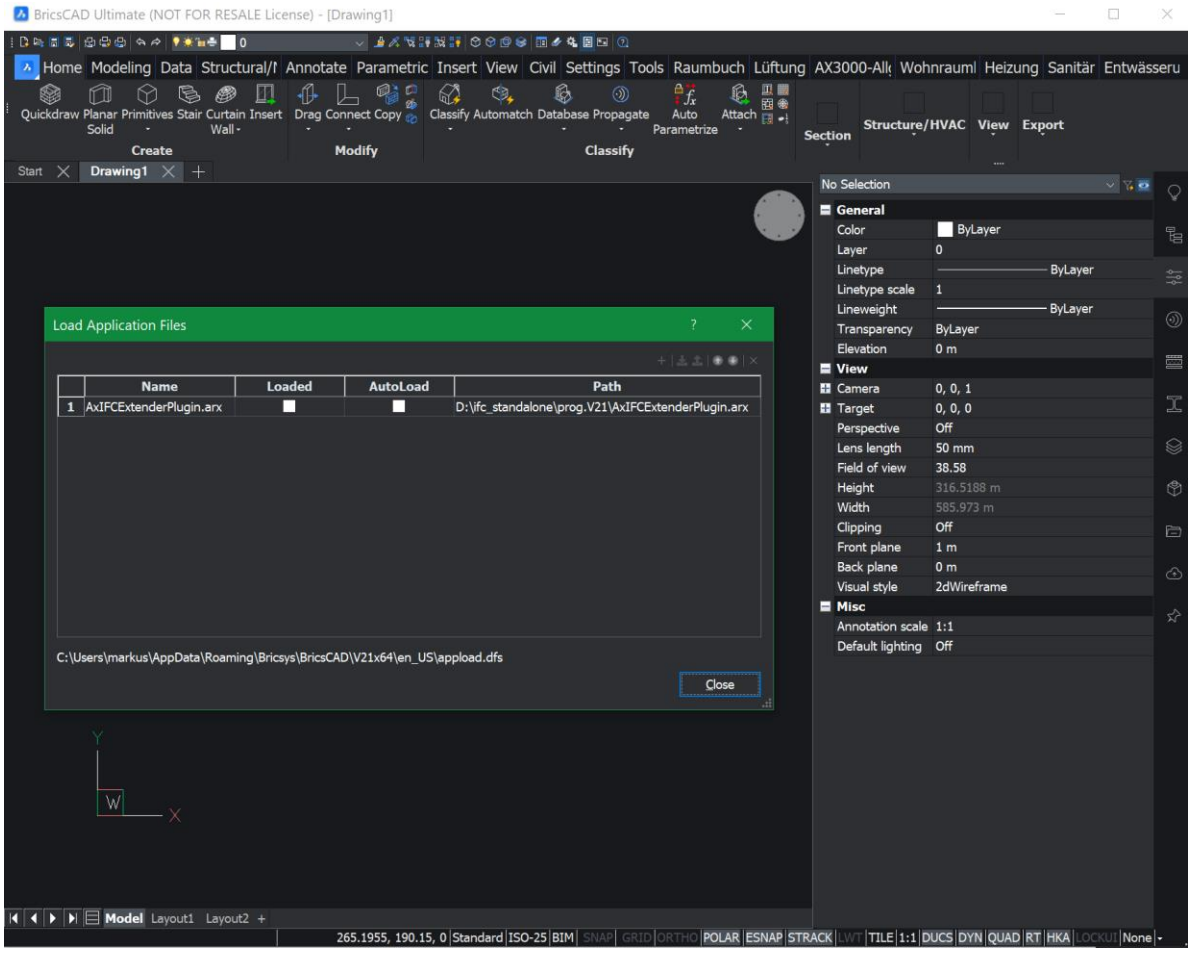


*Abbildung 9: BricsCAD mit Appload-Dialog*

2. „Geladen“ Hacken setzen. In der BricsCAD-Statusleiste erscheinen daraufhin folgende Ausgaben:

```
Loading D:\ifc_standalone\prog.V21\AxIFCExtenderPlugin.arx
AX3000 IFC Reactor base path: D:\ifc_standalone
AX3000 IFC Import Reactor attached!
AX3000 IFC Export Reactor attached!
D:\ifc_standalone\prog.V21\AxIFCExtenderPlugin.arx successfully loaded.
```

3. Per Drag&Drop eine IFC-Testdatei importieren

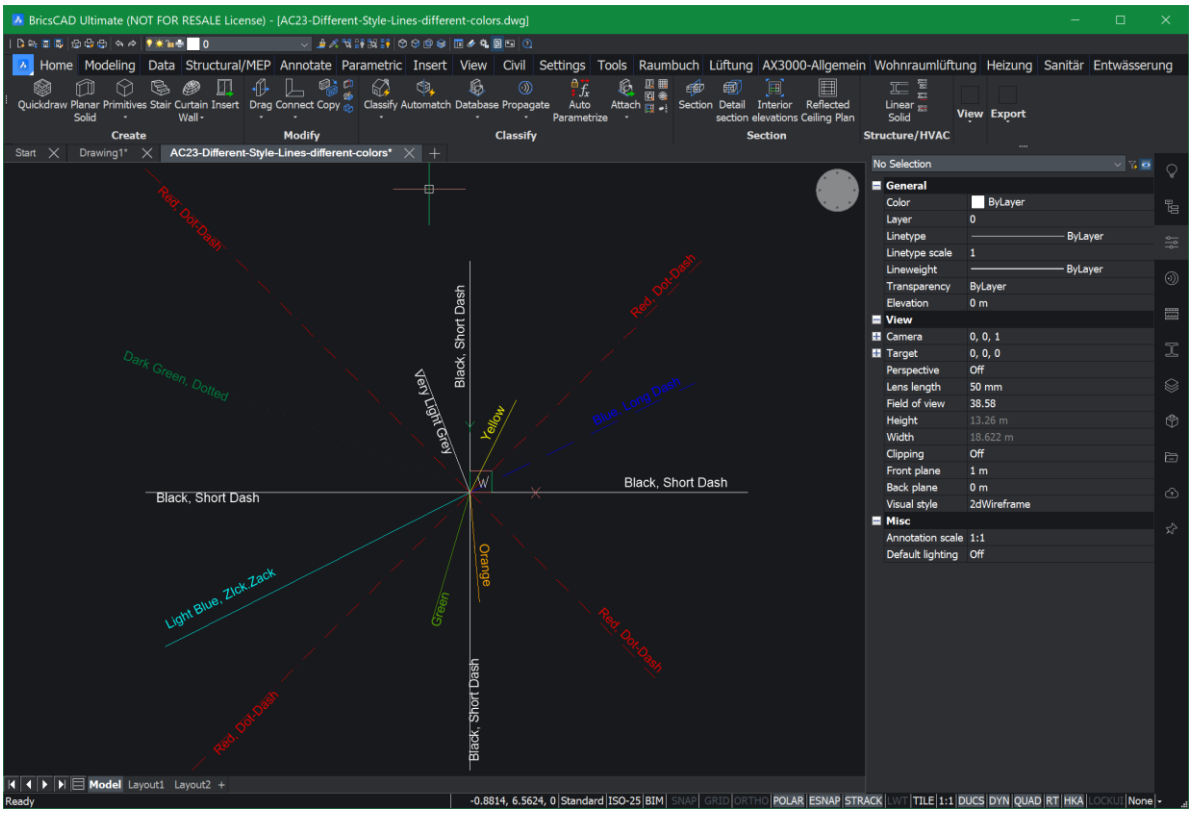


*Abbildung 10: Beispieldatei geladen*

## KONFIGURATION

Im Installationsverzeichnis befindet sich ein Unterordner „ifc“ mit Konfigurationsdateien im XML-Format. Über diese Dateien kann das Import/Export-Verhalten des Plugins geändert werden. Standardmäßig werden Elemente vom Typ „*IfcAnnotation*“ und „*IfcProxy*“ importiert. Inhalt der Datei „IfcImportSettings.xml“:

```
<?xml version="1.0" encoding="utf-8"?>
<IfcImportSettings>
  <Setting Type="IfcAnnotation" Import="true" />
  <Setting Type="IfcProxy" Import="true" />
</IfcImportSettings>
```

Ist es gewünscht, dass *IfcAnnotation*-Objekte nicht mehr über das Plugin importiert werden, können diese hier deaktiviert werden indem „Import=“false““ eingetragen wird.

Die Einstellungen zum Export befinden sich in „IfcExportSettings.xml“ und definieren die grundlegenden CAD-Elemente, die verarbeitet werden sollen:

```
<?xml version="1.0" encoding="utf-8"?>
<IfcExportSettings>
  <Setting Type="AcDbLine" Export="true" />
  <Setting Type="AcDbPolyline" Export="true" />
  <Setting Type="AcDb3dPolyline" Export="true" />
  <Setting Type="AcDb2dPolyline" Export="true" />
  <Setting Type="AcDbArc" Export="true" />
  <Setting Type="AcDbCircle" Export="true" />
  <Setting Type="AcDbEllipse" Export="true" />
  <Setting Type="AcDbText" Export="true" />
  <Setting Type="AcDbSpline" Export="true" />
  <Setting Type="AcDbPoint" Export="true" />
  <Setting Type="AcDbMline" Export="true" />
  <Setting Type="AcDbRay" Export="true" />
  <Setting Type="AcDbHatch" Export="true" />
  <Setting Type="AcDbBlockReference" Export="true" />
  <Setting Type="AcDbRasterImage" Export="true" />
  <Setting Type="AcDbMText" Export="true" />
</IfcExportSettings>
```

Innerhalb von Blöcken (sofern AcDbBlockReference exportiert wird) werden immer alle Objekte exportiert.

Auf das Mapping von Textstilen kann über die Datei „IfcFontMapping.xml“ Einfluss genommen werden. Sollten Textstile (per Name) im IFC definiert sein, die in BricsCAD nicht verfügbar sind, kann ein Mapping zwischen Quelle und Ziel angegeben werden:

```
<?xml version="1.0" encoding="utf-8"?>
<IfcFontMapping>
  <Mapping Source="xyz" Target="Arial" />
  <Mapping Source="*" Target="Arial" />
</IfcFontMapping>
```

Der letzte Eintrag mit „*“ ist ein Catch-All Mapping, greift also dann, wenn keiner der vorangehenden Einträge greift.

## FUNKTIONSTEST

Das Plugin verfügt über eine Anzahl von IFC-Dateien zum Selbsttest. Diese können über den Befehl AXIFC_REACTOR_TEST gestartet werden und laufen im Anschluss automatisch durch. Nach Abschluss bleiben die erstellten DWG-Dateien geöffnet und das Programm zeigt ein Protokoll über den Test an. Dieses sieht so aus (Ausschnitt):

```
#001 16:21:40 - [Success] - TestIFC_ImportReactor1 (blaue Polylinie)
#002 16:21:41 - [Success] - TestIFC_ImportReactor2 (versch Linien als Poly)
#003 16:21:41 - [Success] - TestIFC_ImportReactor2a (versch Linien als Linie)
#004 16:21:42 - [Success] - TestIFC_ImportReactor3 (blaue Polylinie)
#005 16:21:42 - [Success] - TestIFC_ImportReactor4 (Arcs)
#006 16:21:43 - [Success] - TestIFC_ImportReactor5 (Kreise als Kreise)
#007 16:21:44 - [Success] - TestIFC_ImportReactor6 (Hatch)
#008 16:21:44 - [Success] - TestIFC_ImportReactor7 (3dPoly aus Allplan)
#009 16:21:45 - [Success] - TestIFC_ImportReactor8 (Texte mit Orientierung)
#010 16:21:45 - [Success] - TestIFC_ImportReactor9 (Texte mit Ausrichtung)
#011 16:21:46 - [Success] - TestIFC_ImportReactor10 (Schraffur aus Revit)
#012 16:21:47 - [Success] - TestIFC_ImportReactor11 (Symbole und Text)
#013 16:21:48 - [Success] - TestIFC_ImportReactor12 (großer 2D Plan)
#014 16:21:49 - [Success] - TestIFC_ImportReactor13 (Allplan 2021)
#015 16:21:49 - [Success] - TestIFC_FontMapper
#016 16:21:49 - [Success] - TestIFC_ImportSettings
#017 16:21:49 - [Success] - TestIFC_ExportSettings
#018 16:21:50 - [Success] - TestIFC_ExportReactor1 (3 Linien)
#019 16:21:51 - [Success] - TestIFC_ExportReactor2 (Linie, Poly, Layer)
#020 16:21:52 - [Success] - TestIFC_ExportReactor3 (3D-Poly)
#021 16:21:53 - [Success] - TestIFC_ExportReactor4 (Arcs)
```

Getestet werden hier verschiedene Funktionen wie das Einlesen einer vorgegebenen IFC-Datei (mit Validierung der erzeugten Elemente im DWG), den Export einer vorgegeben DWG-Datei als IFC (mit Validierung des IFC durch das KIT IFCCheckingTool) und das Mapping von Textstilen und Füllflächen. Die zum Test verwendeten IFC-Daten sind auf Anfrage verfügbar, aber nicht auf GitHub hochgeladen.

# 4. Umsetzung auf Basis der BricsCAD API

Die Entwicklung erfolgte mit Visual Studio 2019 und C++ (2017er Standard). Die gesamte Funktionalität ist in zwei DLLs enthalten, wobei diese wie folgt aufgeteilt ist:

| | |
|---|---|
| AxIFCExtenderPlugin.arx | Wird von BricsCAD geladen und aktiviert die Import/Export Reactor |
| AxIFCExtender.dll | Einlesen und verarbeiten der IFC-Elemente und CAD-Objekte |

Beim Laden der AxIFCExtenderPlugin.arx Datei werden die beiden Reactor in BricsCAD registriert und beim Entladen wieder entfernt. Ein manuelles aktivieren ist nicht notwendig.

## IFC-IMPORT

Der Reactor-Code befindet sich in „AxIFCExtenderPlugin\AxIFCImportReactor.cpp“. In der Funktion

```
AxIfcImportReactorInstance::onStart
```

werden die oben genannten Einstellungen (XML-Dateien) eingelesen.

```
m_FontMapperTraits.FontMappingXMLFile = PathHelper::MakePath({ ProgramSettings::Get()->GetProgPath(), _T("ifc"), _T("IfcFontMapping.xml") });
	m_ImportSettingsTraits.SettingsXMLFile = PathHelper::MakePath({ ProgramSettings::Get()->GetProgPath(), _T("ifc"), _T("IfcImportSettings.xml") });
	m_PatFileTraits.PatFileLocationToUse = SupportPath;
```

Das Abarbeiten der IFC-Elemente erfolgt in

```
AxIfcImportReactorInstance::onIfcProduct
```

Am Beispiel der *IfcAnnotation*:

```
if (product.isKindOf(IfcAnnotation)) {
	if (m_ImportSettings->ShouldImport(CString(name))) {
		m_ImportWorker->DoIfcAnnotation(product);
	}
	return true;
}
```

Das product-Element wird von BricsCAD an die Reactor-Funktion durchgereicht. Hier wird geprüft um welchen Typ es sich handelt. Product ist verallgemeinert ein Ice::IfcApi::Entity Element. Mit isKindOf wird die Spezialisierung geprüft. Handelt es sich um *IfcAnnotation*, wird in der Konfiguration geprüft, ob das Element zu importieren ist. In weiterer Folge wird das Element an den ImportWorker weitergereicht.

Die vom Importer unterstützten IFC-Entities wurden als C++-Klasse abgebildet. Das Lesen der Attribute und eventueller zugehöriger (Aggregates) Objekte erfolgt in der Klasse selbst. Die Klassenhierarchie ist der IFC-Hierarchie nachempfunden (mit ein paar Abweichungen bezüglich der Stile).

```
auto p = std::make_shared<AxIfcAnnotation>(AxIfcType::IfcAnnotation, product, m_GetLocalPlacement, m_Conversion);
p->Parse(m_Logger);
```

Die Methode Parse() erhält das Logger-Objekt, das während des Imports einmalig erzeugt wird und am Ende – im Falle von Meldungen – angezeigt wird. *IfcAnnotation* ist direkt von der abstrakten Basisklasse *IfcBaseEntityParser* abgeleitet. Diese enthält die Parse()-Methode, die jede abgeleitete Klasse implementieren muss.

```
class AxIfcAnnotation : public AxIfcBaseEntityParser
```

Innerhalb von Parse() für die *IfcAnnotation* wird z.B. das Attribut „Name“ über die BricsCAD-API gelesen:

```
Variant var;
auto result = m_Ent.getAttribute(Ice::IfcApi::Name, var);
auto type = var.type();
if (result == Result::eOk) {
        if (type == ValueType::eString) {
                m_Name = var.getString();
        } else {
                LogError(Logger, __FUNCTIONW__, AxIFCLogger::UnexpectedType);
        }
}
```

Der ermittelte Wert wird anschließend in der Klasse als Klassenmember gespeichert. Attribute mit Längen- oder Winkeleinheiten werden in die Zieleinheit (Einheit des DWGs) umgerechnet. Hier am Beispiel des XDIM-Attributs von *IfcBoundingBox*:

```
Variant varDim;
        result = m_Ent.getAttribute(XDim, varDim);
        type = varDim.type();
        if (type == ValueType::eReal) {
                m_XDim = varDim.getReal() * m_Conversion.LengthConversion;
        } else {
                LogError(Logger, __FUNCTIONW__, AxIFCLogger::UnexpectedType);
        }
```

Der Attributwert wird mit dem Längenfaktor multipliziert. Ist die Einheit im IFC Meter und auch das DWG wurde mit Meter (INSUNITS) erstellt, so ist dieser Faktor 1. Ist das DWG in Millimeter gespeichert, ist dieser Faktor 1000.

Komplexere Elemente wie z.B. die Representation sind als eigene Klassen in *IfcAnnotation* enthalten und verfügen über ihre eigene Parse()-Methode:

```
m_Representation =
std::make_shared<AxIfcProductRepresentation>(AxIfcType::IfcProductRepresentation, ent,
m_GetLocalPlacement, m_Conversion);
if(!m_Representation->Parse(Logger)){
        LogError(Logger, __FUNCTIONW__, AxIFCLogger::SubParseError);
}
```

Jedes Objekt ruft auch immer die Parse()-Methode der übergeordneten Klasse auf. Die Klassenbezeichnungen selbst wurden jeweils um ein „Ax“ am Anfang erweitert um eventuelle Überschneidungen mit BricsCAD eigenen Klassen zu vermeiden. Aus diesem Grund wurde auch immer

 

explizit der Ice::IfcApi-namespace verwendet, wenn BricsCAD-API Elemente/Funktionen verwendet wurde.

Während des Imports werden die IFC-Elemente zunächst von BricsCAD selbst auf Gültigkeit laut Schema geprüft. Elemente, die hier durchfallen, werden gar nicht an den Reactor weitergereicht.

Eine Hierarchie am Beispiel von `IfcTextLiteralWithExtent`:


*Abbildung 11: Klassenhierarchie-Beispiel*

Diese grundlegende Vorgehensweise ist für alle Ifc-Klassen identisch, variiert aber natürlich je nach Objekt. Die Aufgabe des IFC-ImportWorkers ist es, alle Ifc-Objekte während des Imports zu verarbeiten, Stile zu lesen, Aggregates zu ermitteln und diese Informationen je nach Hierarchie auch auf untergeordnete Elemente zu verteilen.

Eine Sonderbehandlung ist für `IfcProxy`-Objekte notwendig, da diese von BricsCAD nicht über onIfcProduct an den Reactor weitergegeben werden. Sie werden am Ende des Imports in der Reactor-Funktion beforeCompletion() separat ermittelt. Hier muss nochmals über alle Entities iteriert und die speziell definierten Proxy-Elemente eingelesen werden.

Am Ende des Vorgangs steht eine komplette Liste mit allen vollständig definierten Ifc-Elementen zur Verfügung. Diese wird im Anschluss an den IFC-Creator weitergegeben.

 

## ERZEUGUNG DER IFC-ELEMENTE IN BRICSCAD

### Schritt 1 – Erzeugung der Layouts und Viewports

Bevor die CAD-Elemente (Linien, Texte, Kreise, etc.) erzeugt werden, müssen der Papierbereich und die Layouts angelegt werden. Das hat Performance-Gründe, da im Laufe des Imports über den `IfcGeometricSubContext` pro Representation festgelegt ist, ob diese im Modell oder auf einem Papierbereich zu erzeugen sind.

Eine besondere Herausforderung beim Erstellen der Papierbereiche waren die benutzerdefinierten Papiergrößen in den Beispiel-Projekten. In BricsCAD ist es über die API nicht möglich diese zur Laufzeit zu erzeugen. In BricsCAD (und auch AutoCAD) werden diese in PC3-Dateien gespeichert. Diese enthalten Informationen zum Drucker und eben den Papiergrößen. Wird eine Papiergröße nicht in den Standardgrößen gefunden, so wird sie in der PC3-Datei zur Laufzeit angelegt. BricsCAD liest diese Datei allerdings nicht neu ein während der Laufzeit, sodass benutzerdefinierte Papiergrößen erst beim nächsten Import automatisch korrekt gesetzt werden. Einmal erzeugte Papiergrößen bleiben erhalten.

Nach dem Layout werden alle zum Layout gehörigen Viewports erzeugt. Hierbei wird auf die in den Attributen definierte Skalierung, Position und Höhe Rücksicht genommen. Die Viewports in BricsCAD sind sogenannten Live-Viewports, d.h. sie sind keine fixen Schnitte, sondern stellen basierend auf Kameraposition, Blickrichtung, etc. den Modellbereich dar. Änderungen im Modell sind somit sofort auch in den entsprechenden Viewports sichtbar. Im Zuge des Roundtrip-Projekts werden aus AutoCAD Architecture aber auch statische Viewports exportiert. Das erfolgt durch mehrere `IfcGeometricRepresentationSubContext` pro `IfcRepresentation`. Somit hat eine `IfcAnnotation` mehrere Darstellungen (hier am Beispiel mit dem EXPRESS-Data-Browser des FZK-Viewers von KIT):

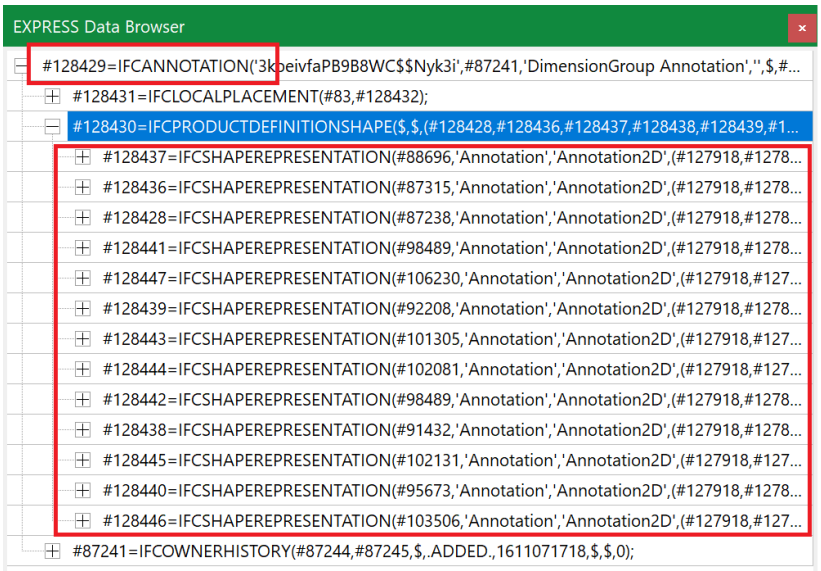


*Abbildung 12: EXPRESS-Data-Browser*

Diese statischen Darstellungen werden vom Plugin ignoriert und dienen der Darstellung in reinen Viewern, wie z.B. dem FZK-Viewer.

### Schritt 2 – Erzeugung von Elementen

Für jede *IfcAnnotation* werden entsprechend des Aufbaus BricsCAD spezifische CAD-Elemente erzeugt. In manchen Punkten ist die Zuordnung eindeutig (z.B. *IfcCartesianPoint* zu AcDbPoint), in anderen versucht das Plugin eine für den Benutzer optimale Lösung zu finden. Das betrifft zum Beispiel *IfcIndexedPolyCurve*, die in IFC eine Polylinie, Linie, Kreis und einen Kreisbogen repräsentieren kann. Für all diese Objekte gibt es aber spezialisierte Entsprechungen in BricsCAD. Es wurde also wie folgt vorgegangen:

| IFCINDEXEDPOLYCURVE MIT | |
|---|---|
| Nur 2Punkten | AcDbLine |
| Mehreren Punkten, auf einer Ebene | AcDbPolyline |
| Mehrere Punkte, auf mehreren Ebenen | AcDb3dPolyline |
| 3 Punkten, die einen Kreisbogen darstellen | AcDbArc |
| 5 Punkten, die einen geschlossenen Kreis darstellen | AcDbCircle |

Es wäre natürlich weniger aufwändig gewesen immer eine AcDb3dPolyline zu erstellen, allerdings ist das Ändern eines Kreisradius in diesem Fall nicht so leicht möglich, wie bei einem Kreis-Element. Über ImportCreatorTraits, die das Verhalten des Creators verändern, ist es einfach möglich diese Vorgehensweise zu unterdrücken:

```
class AXIFCEXTENDER_EXPORT AxIFCImportCreatorTraits {
public:
	void LoadDefaults();

public:
	bool CheckForSimpleLines = false;
	bool CheckForCirclesInPolyCurves = false;
	bool ApplyDefaultSolidHatch = true;
};
```

Gut ersichtlich wird dies am Beispiel eines Kreises, der in IFC wie folgt definiert werden kann:

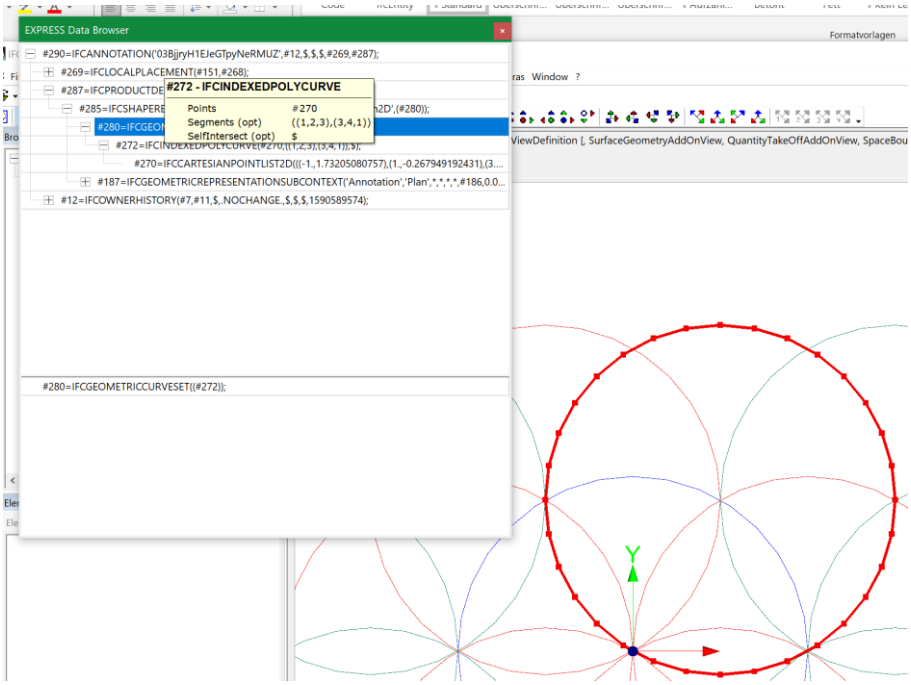


*Abbildung 13: Kreis als IfcIndexedPolycurve*

Eine direkte Übernahme als Polylinie führt in BricsCAD zu diesem Ergebnis:

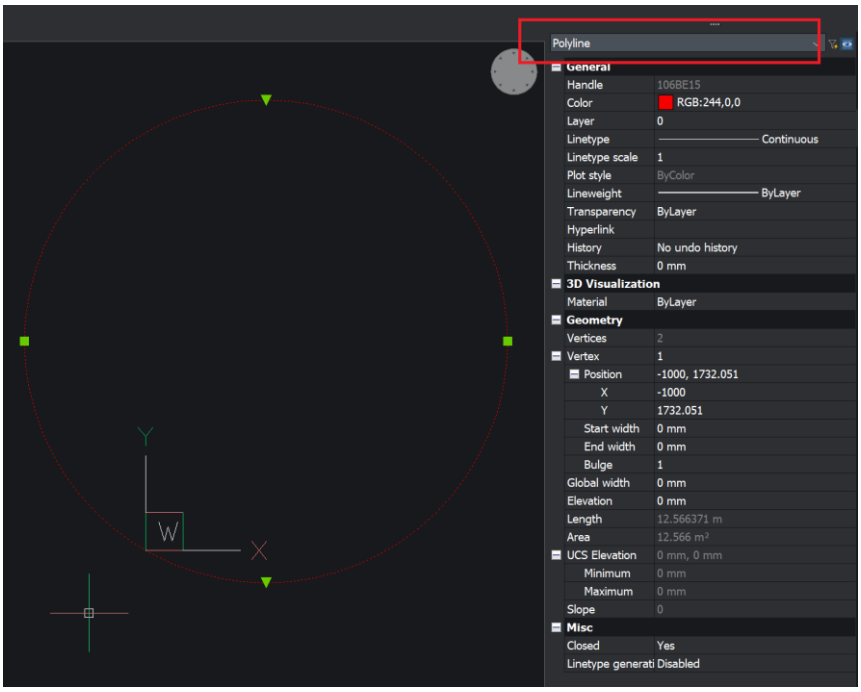


*Abbildung 14: Kreis als Polylinie in BricsCAD*

Im Gegenzug dazu hier als Kreis:

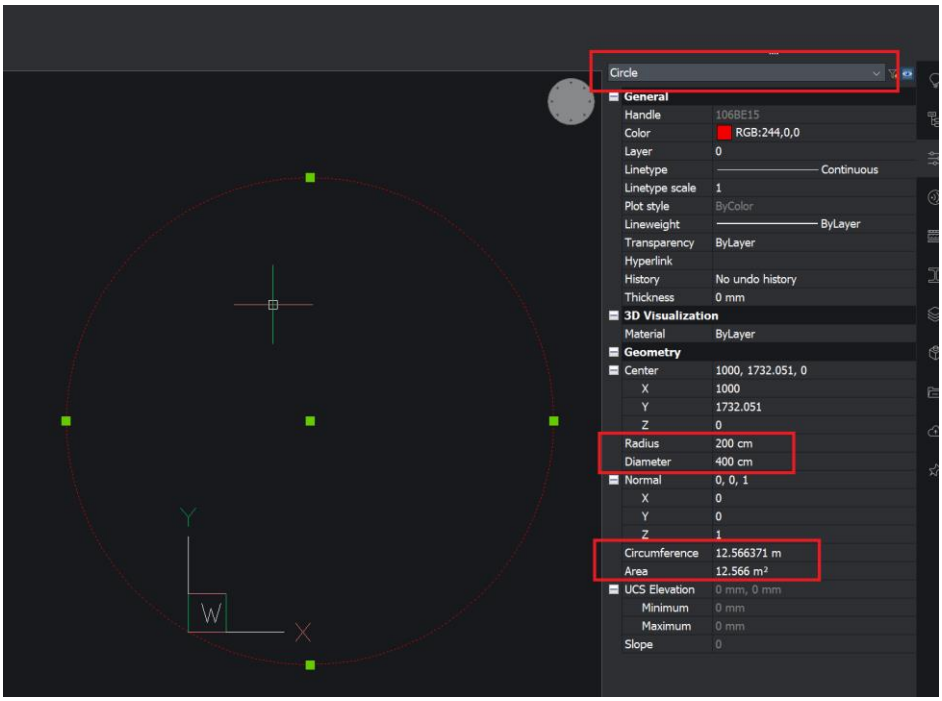


*Abbildung 15: Kreis als Kreis in BricsCAD*

Optisch ist kein Unterschied zu erkennen, sehr wohl aber in der Bedienbarkeit. Während man z.B. den Radius in der Polylinie nur über Verzerren ändern kann, ist dies bei einem Kreis direkt über die Eigenschaften möglich.

Die Elemente werden zunächst nur im Speicher erzeugt und je nach zugewiesenem Context auf dem Modellbereich oder auf den zuvor erstellten Layouts abgelegt.

### Schritt 3 – Stile

Die meisten der Ifc-Elemente können über einen oder mehrere Stile verfügen, die Eigenschaften wie Farbe, Layer, Liniendicke, Linienstil, Füllstil usw. definieren. Nach der Erzeugung der Elemente in Schritt 2, werden diese Stile evaluiert und im CAD erzeugt. Stile wie Farben und Liniendicke werden pro Element gesetzt und finden sich nicht in einer generellen Übersicht wieder. Anders verhält es sich mit benannten Stilen wie Textstilen, Füllstilen und Layern. Diese werden nicht für jedes Objekt neu erzeugt, sondern zunächst wird geprüft, ob ein betreffender Stil bereits vorhanden ist. Wenn nicht, wird der Stil entsprechend der Vorgaben im IFC erzeugt und anschließend dem Element zugewiesen.

Im speziellen bei Textstilen, Füllstilen und Linienstilen können nicht alle Möglichkeiten von AutoCAD und BricsCAD auch in IFC abgebildet werden. Hier mussten im Zuge des Roundtrips Vereinbarungen und Vereinfachungen getroffen werden um ein möglichst identisches Bild beim Import zu erhalten.

Als Beispiel sollen hier die Füllstile genannt werden, bei denen wir über das Description-Attribut der `IfcAnnotation` fehlende Eigenschaften wie den Winkel der Füllung und die Skalierung ausgetauscht haben. Viewer, die nicht im Roundtrip eingebunden sind, würden hier eine etwas andere Darstellung bieten als die im Roundtrip vertretenen Systeme. Es wurde aber die Kompatibilität gewahrt.

### Symbole – Blöcke

Im Quellsystem sind verschiedene Symbole vorhanden, wie z.B. Feuerlöscher, Erste-Hilfe-Kästen usw. Diese werden als `IfcMappedItem` exportiert. In IFC ist es allerdings aktuell nicht möglich den Namen des Symbols anzugeben, was in BricsCAD zu unbenannten Blöcken führen würde. Das erschwert z.B. den Austausch eines Blocks oder das Ermitteln der Anzahl, da jeder Block einen anderen Namen hat. Auch hier wurde im Zuge des Roundtrip eine Umgehung gefunden und der Name als Attribut an das MappedItem angehängt.

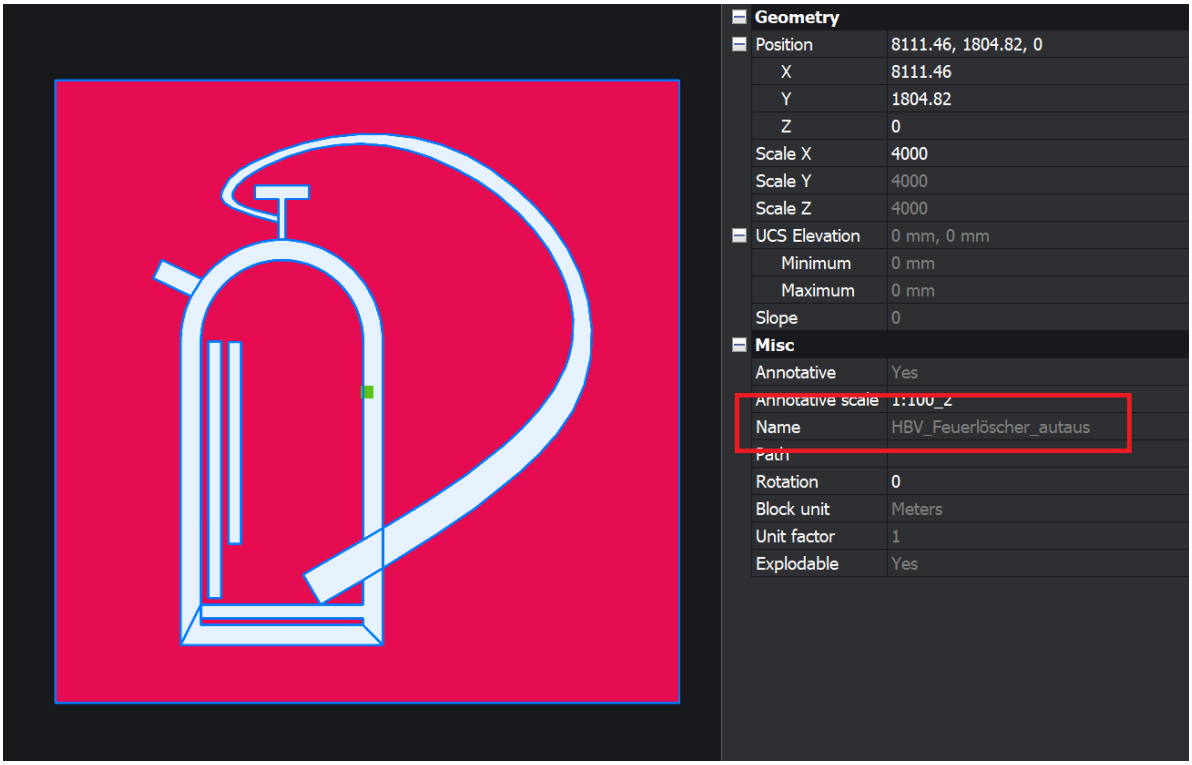


*Abbildung 16: Feuerlöscher-Symbol mit Namen*

Dadurch kann in BricsCAD über den BIM-Structure Browser sehr einfach und übersichtlich ermittelt werden, wo sich welche Anzahl von Symbolen befindet. Ebenfalls ist ein Austausch durch Ändern der Blockdefinition möglich. Wie dies am Beispiel des DWGs 7E00-011-EG-B aussehen kann soll hier gezeigt werden:

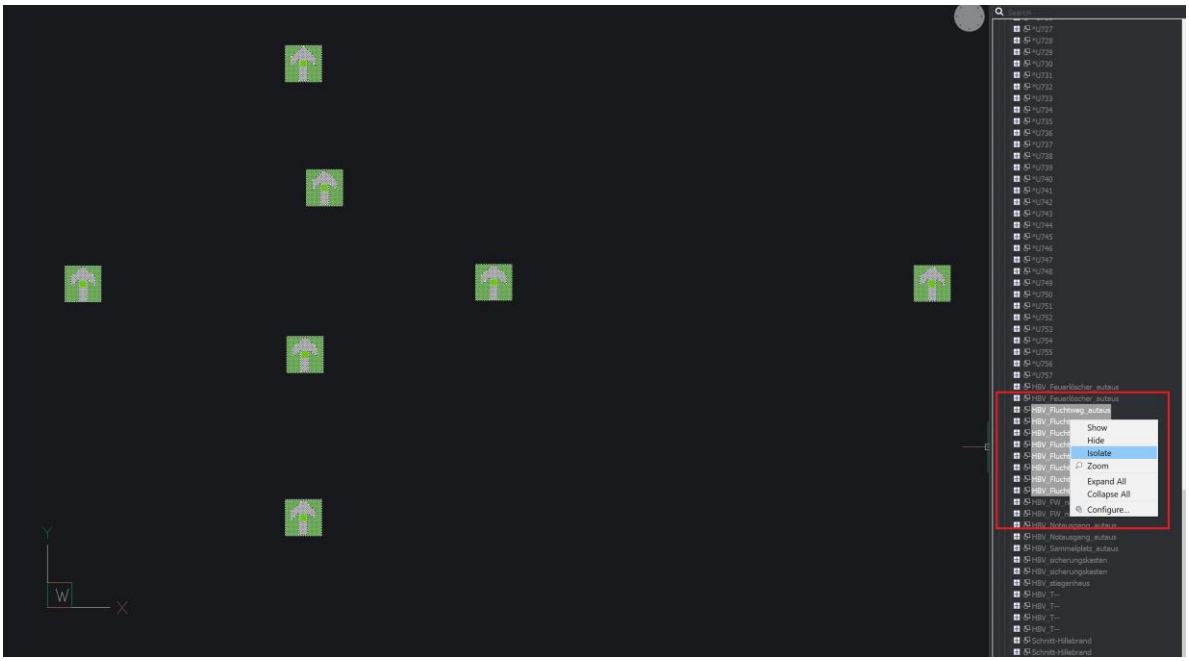


*Abbildung 17: Isolierte Ansicht von Symbolen*

### Papierbereich

Am Beispiel des „Flucht- und Rettungsplan“ hier der Import des Papierbereichs durch das Plugin:

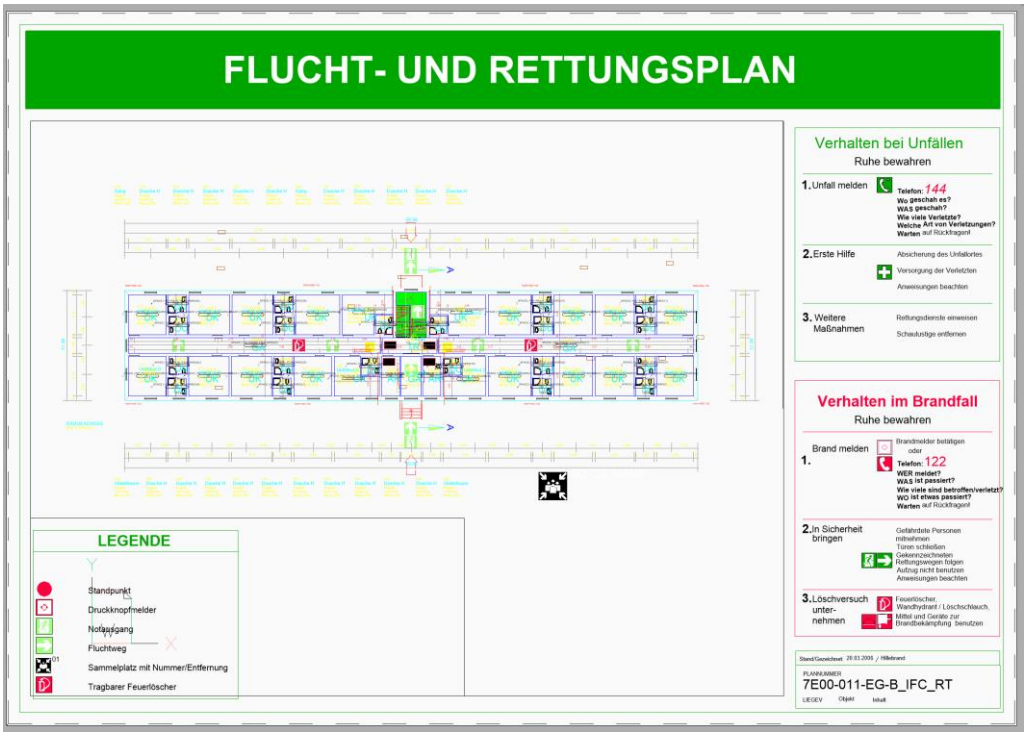


*Abbildung 18: Papierbereich Import*

An diesem Beispiel gut ersichtlich sind vor allem die Gestaltungen am Plankopf wie Symbole, Texte und Farben.

### Anmerkungen zum Import

Layer: In IFC ist es möglich mehrere unterschiedliche Layer mit dem selben Namen zu exportieren. Das kann in BricsCAD so nicht importiert werden. Die Vorgehensweise hier ist aktuell, dass der erste Layer mit einem Namen auch die Definition vorgibt. Sollte anschließend ein weiterer Layer mit gleichem Namen, aber anderen Stilen im IFC vorkommen, wird diese zweite Definition ignoriert.

## IFC-EXPORT

Der Reactor-Code befindet sich in „AxIFCExtenderPlugin\AxIFCExportReactor.cpp“. In der Funktion

```
AxIfcExportReactorInstance::onBeginIfcModelSetup
```

werden die Einstellungen gelesen.

```
m_ExportSettingsTraits.SettingsXMLFile = PathHelper::MakePath({ ProgramSettings::Get()-
>GetProgPath(), _T("ifc"), _T("IfcExportSettings.xml") });
        delete m_ExportSettings;
        m_ExportSettings = new AxIFCExportSettings(m_ExportSettingsTraits);
        m_ExportSettings->Init();
        m_ExportWorker->Init();
```

Von BricsCAD wird nun für jedes Element im Modellbereich die Funktion

```
AxIfcExportReactorInstance::onEntity
```

aufgerufen. Innerhalb dieser entscheidet der Reactor, ob das Element exportiert werden soll.

```
if (m_ifc4 && m_ExportSettings->ShouldExport(pEntity)) {
                bool Export = true;
                if (pEntity->isKindOf(AcDbBlockReference::desc())) {
                        // Blöcke nur dann exportieren, wenn sie entweder nicht klassifiziert
sind, oder als Annotations
                        if (!BimClassification::isUnclassified(pEntity->objectId()) &&
                                !BimClassification::isClassifiedAs(pEntity->objectId(),
BimApi::eBimAnnotation)) {
                                Export = false;
                        }
                }
                if (Export)
                        return m_ExportWorker->Export(ifcModel(), pEntity);
        }
```

Generell wird der Reactor nur aktiv, wenn ein IFC4 Export erfolgen soll. Wenn das der Fall ist, wird geprüft, ob das Element laut Konfiguration zu exportieren ist. Da nicht jeder Block automatisch als *IfcAnnotation* exportiert werden muss (und auch kann), wird zunächst geprüft, ob es sich um einen BIM-Definierten Block handelt. Das ist z.B. bei Fenster- und Türobjekten der Fall. Diese werden von BricsCAD selbst als *IfcWindow* bzw. *IfcDoor* exportiert.

Es werden also nur Blöcke ohne BIM-Definition oder jene, die als Annotation deklariert sind, exportiert. Speziell bei den Blöcken ist es in BricsCAD auch möglich, dass diese verschachtelt definiert sind. Das bedeutet, dass ein Block aus mehreren anderen Blöcken bestehen kann, die wiederum aus anderen Blöcken bestehen können. Diesem Umstand wird durch eine rekursive Implementierung des Block-Exports Rechnung getragen.

Es werden die folgenden CAD-Elemente exportiert:

| CAD-Element | IFC-Typ |
|---|---|
| AcDbLine | IfcPolyline |
| AcDbPolyline, AcDb2dPolyline, AcDb3dPolyline | IfcPolyline |
| AcDbArc | IfcIndexedPolyCurve |
| AcDbCircle | IfcCircle |
| AcDbEllipse | IfcEllipse bzw. IfcTrimmedCurve |
| AcDbText | IfcTextLiteralWithExtent |
| AcDbMText | IfcTextLiteralWithExtent |
| AcDbSpline | IfcBSplineCurveWithKnots |
| AcDbPoint | IfcCartesianPoint |
| AcDbMline | Mehrere IfcPolyline |
| AcDbRay | IfcLine |
| AcDbHatch | IfcAnnotationFillArea |
| AcDbRasterImage | IfcAnnotationFillArea |
| AcDbBlockReference | IfcMappedItem |

### Layer

Nach dem Erzeugen der Objekte, werden die Stile und Layer des CAD-Elements ermittelt und ebenfalls als IFC-Element exportiert. Als Ifc-Entity für die Layer wird `IfcPresentationLayerWithStyle` verwendet, da das Plugin neben Namen und Beschreibung auch die Zustände „On, Frozen, Locked“ exportiert. Zusätzlich zu diesen Einstellungen wird auch die Layer-Farbe, der Linienstil und die Dicke berücksichtigt. Diese werden als `IfcCurveStyle`-Objekte exportiert.

Die Layer in BricsCAD bieten allerdings auch weitere Eigenschaften, die aktuell in IFC nicht abgebildet werden können. Beispielsweise die Transparenz.

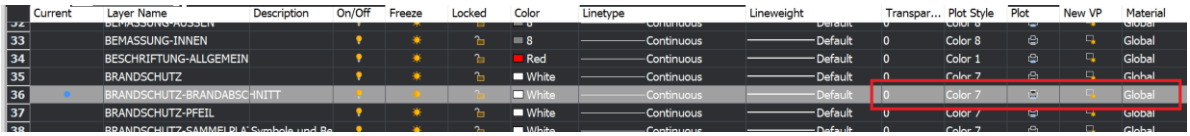


*Abbildung 19: Layereigenschaften*

Die rot markierten Eigenschaften können aktuell in IFC nicht abgebildet werden.

### Farben

Farben werden in BricsCAD entweder als Farbindex gespeichert, oder als RGB-Wert. Bei den Indizes gibt es spezielle Werte, die keine Farbe an sich definieren, sondern z.B. den Zustand „von Layer“ oder „von Block“. Beim Export wurde also berücksichtigt, ob eine Farbe zu den in `IfcDraughtingPreDefinedColour` definierten Indizes passt und dann als solches exportiert. Alle anderen Farben wurden als RGB geschrieben. Um bei sehr vielen Elementen die Dateigröße nicht explodieren zu lassen, wird jede Farbe (egal ob zu einem Element direkt gehörend, zu einem Layer oder einem Stil) nur einmal ins IFC geschrieben.

Hier noch die Übersicht der vordefinierten Farben in IFC:

| IFC | BricsCAD |
|---|---|
| Black | Index 7 – schwarz/weiß – wird je nach Hintergrundfarbe in BricsCAD gewechselt |
| Red | Index 1 |
| Green | Index 3 |
| Blue | Index 5 |
| Yellow | Index 2 |
| Magenta | Index 6 |
| Cyan | Index 4 |
| White | Index 7 |
| By Layer | Index 256 |

### Textstile

Bei den Textstilen unterstützt BricsCAD eigene Fonts, die so genannten SHX-Fonts. Diese sind BricsCAD (bzw. AutoCAD) spezifisch und in anderen Systemen so nicht vorhanden. Im Zuge des Roundtrips spielt dieser Punkt aber eine untergeordnete Rolle, da das exportierende System ebenfalls SHX-Fonts unterstützt und diese somit zwischen den Systemen ausgetauscht werden können.

### Linienstile

Einfache Linienstile wie „Strichpunkt“ oder „Strichliert“, bzw. allgemein alle Stile die nur mit Punkten und Strichen in einer Linie auskommen, können korrekt in IFC exportiert werden. Komplexere Linienstile, wie z.B. Zickzack oder mit enthaltenen Sonderzeichen werden nicht exportiert.

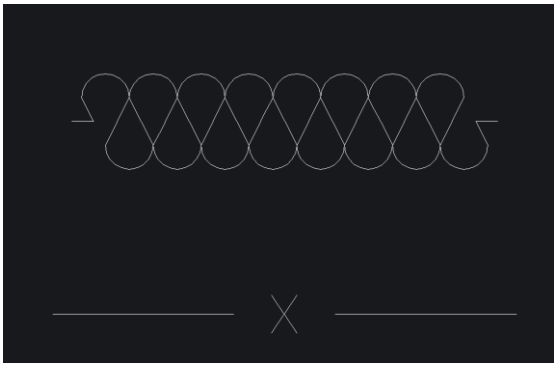

*Abbildung 20: Beispiel für Linienstile*

Die hier gezeigten Linienstile werden beim Export ignoriert.

### Liniendicken

In BricsCAD kann eine Liniendicke auf mehrere Arten definiert werden. Bei gewöhnlichen Linien (AcDbLine) kann dies über die „Lineweight“ erfolgen, bei Polylinien aber auch über die „GlobalWidth“. Beides wird im IFC Plugin als „LineWidth“ im `IfcCurveStyle` exportiert.

*Abbildung 21: Liniedicken in BricsCAD*

Die Linie im oberen Bereich verfügt über eine Liniendicke. Die Polylinie im unteren über GlobalWidth.

### Füllstile

Eine der größten Herausforderungen beim Export waren die Füllstile. BricsCAD unterstützt hier einige Varianten, die aktuell in IFC nicht abgebildet werden können. Ebenso fehlen einige Attribute, wie z.B. die Drehung oder die Skalierung des Füllstils.

Auch gibt es die Möglichkeit Farbverläufe in BricsCAD zu definieren. Dies wird aktuell nur für die erste definierte Farbe des Verlaufs unterstützt, da in den FillStyles in IFC nur eine Farbe angegeben werden darf.

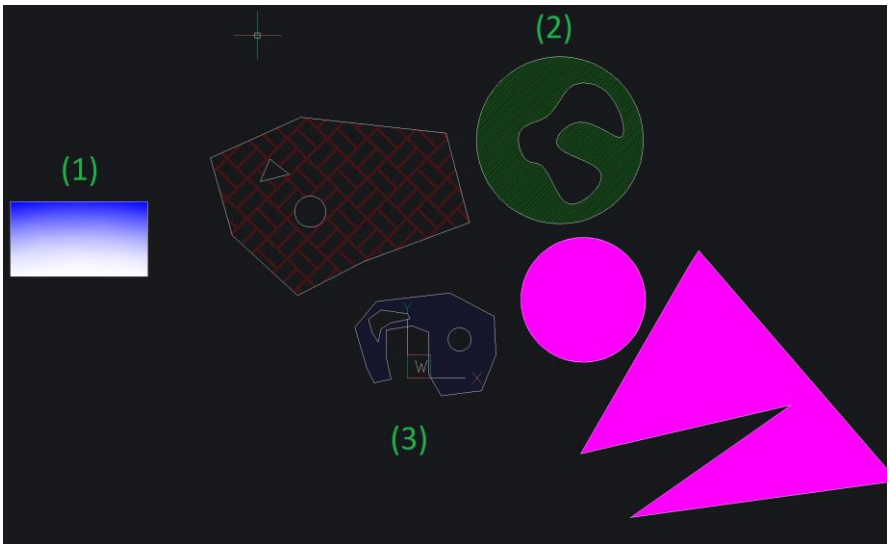


*Abbildung 22: Original*

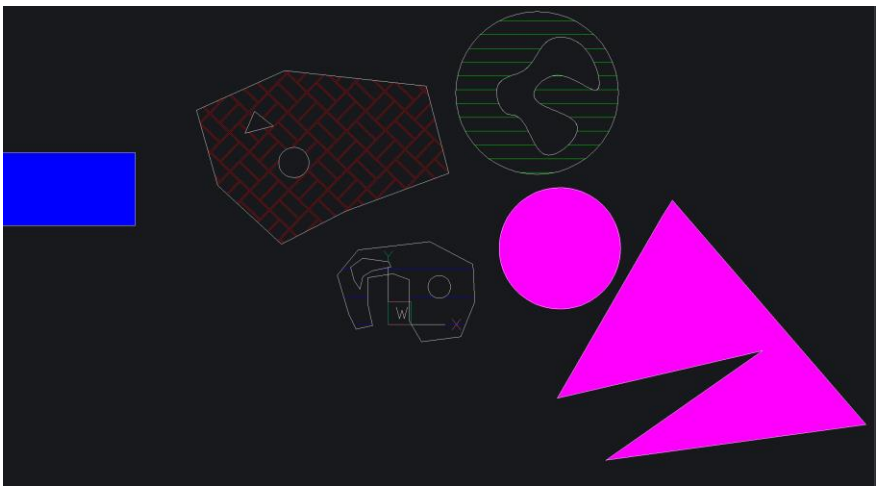

*Abbildung 23: Reimport*

Wie hier ersichtlich wird, fehlt der Farbverlauf von blau zu weiß bei Objekt (1), die Skalierung und Rotation des Füllstils bei Objekt (2) und die Skalierung bei Objekt (3). Komplexere Füllstile müssen in BricsCAD als PAT-Dateien gespeichert werden. Diese PAT-Dateien müssen in einem Support-Verzeichnis

von BricsCAD hinterlegt sein. Das Plugin erstellt diese Dateien und legt sie unter dem Namen des *IfcFillAreaStyle*-Objekts ab. Das bedeutet aber auch, dass Füllstile mit gleichem Namen aber unterschiedlicher Definition nicht unterstützt werden.

```
*Diagonale Kreuzschraffur
45, 0.000,0.000, 0.000, 300.000
135, 0.000,0.000, 0.000, 300.000
```

Inhalt einer PAT-Datei. Für weitere Details kann die Autodesk-Hilfe konsultiert werden:

https://knowledge.autodesk.com/support/autocad-lt/learn-explore/caas/CloudHelp/cloudhelp/2019/ENU/AutoCAD-LT/files/GUID-A6F2E6FF-1717-44B6-A476-0CA817ADD77E-htm.html

### Papierbereich

Aktuell gibt es keine IFC-Entities mit denen der Papierbereich inklusive Viewports exportiert werden könnte. Aus diesem Grund wurde im Zuge des Roundtrips das `IfcProxy` Objekt verwendet und ein Satz Attribute für den Austausch definiert.

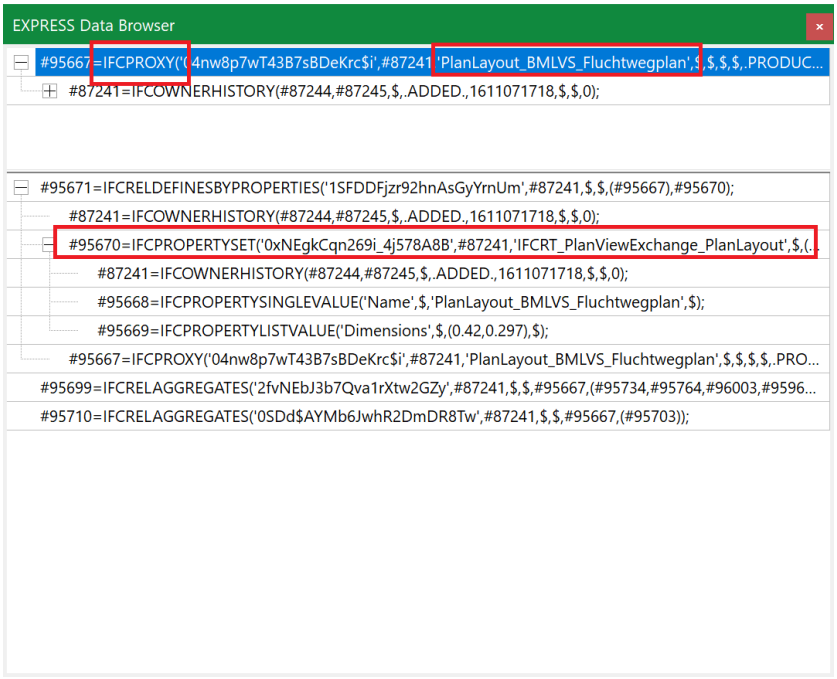


*Abbildung 24: Definition eines Layouts über IfcProxy*

Das Attribut „Dimension“ wird mit der Papiergröße gefüllt. Diese ist in BricsCAD über das „Page Setup“ einstellbar.

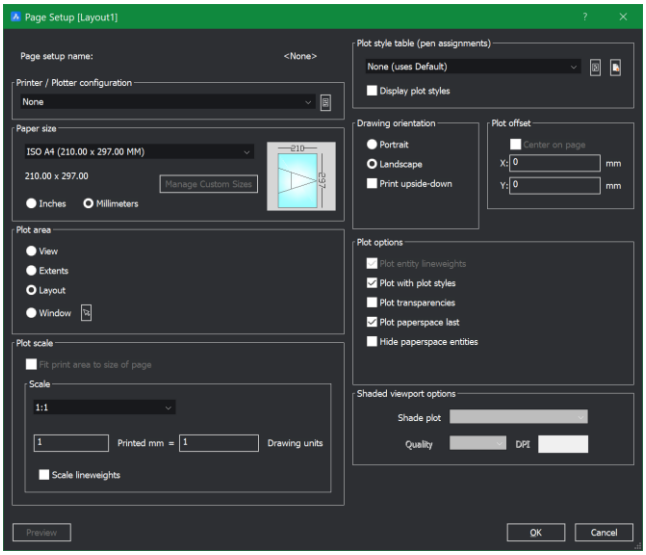


*Abbildung 25: Einstellungen Papierbereich*

Diese Proxy-Objekte werden von anderen Programmen nicht importiert, da hierfür spezielles Wissen um den Aufbau vorhanden sein muss, das nicht im IFC-Standard definiert ist.

Wie weiter oben bereits erwähnt, werden von BricsCAD nur Objekte aus dem Modellbereich an den Reactor weitergeleitet. Aus diesem Grund müssen die Papierbereiche und die Objekte darauf im Nachlauf separat ins IFC gespeichert werden. Das geschieht in

```
AxIfcExportReactorInstance::onEndIfcModelSetup
```

Die Viewports des Papierbereichs werden immer fix als Rechtecke exportiert, da eine in BricsCAD mögliche polygonale Geometrie nicht über die Attribute ausgegeben werden kann.

# 5. Beschränkungen des Plugins

Das Plugin erweitert den IFC-Import von BricsCAD um `IfcAnnotation` und aktuell um `IfcProxy` (sofern als Layout bzw. Viewport definiert). Es werden keine Änderungen am Import von Modelldaten (`IfcWall`, `IfcWindow`, etc) durchgeführt.

Auf Export-Seite werden keine Änderungen an den Gebäudedaten durchgeführt.

Während der Arbeiten am Plugin wurden einige Bugs in BricsCAD entdeckt, die teilweise umgangen werden konnten, teilweise aber auf Ausbesserung von Bricsys warten. Zum jetzigen Stand V21.1.06 sind folgende Punkte offen:

| Bricsys Support-Request-Nr | Beschreibung | Schweregrad |
|---|---|---|
| SR118206 | Wird direkt nach einem IFC-Import ein Export durchgeführt (ohne Speichern), werden falsche Transformationen exportiert | Unbedeutend |
| SR115855 | IfcVoidingFeature wird nicht importiert | Bedeutend |
| SR116638 | Geometrie wird immer mit Absolutkoordinaten exportiert. Die Spatial-Location Definition in BricsCAD unterstützt aktuell keine Offset-Koordinaten. | Unbedeutend |
| SR118838 | Der Layer von BuildingElementProxy-Objekten wird falsch importiert | Bedeutend |
| SR118918 | Die IfcId steht während des Exports noch nicht fest und wird es direkt beim Schreiben definiert. Das führt beim Roundtrip-Hack für die Blocknamen zu einem Problem. | Bedeutend |
| SR120254 | Benutzerpapiergrößen können beim ersten Anlegen nicht direkt verwendet werden. Ein Neustart von BricsCAD ist notwendig. Das passiert aber nur 1x pro Papiergröße. | Unbedeutend |
| SR118860 | Die Papierausrichtung (Hochformat/Querformat) ist in der API verkehrt implementiert. | Unbedeutend |
| SR105107 | Einige Linienstile werden nicht gleich dargestellt wie in AutoCAD | Unbedeutend |
| SR115759 | Binärdaten im IFC können nicht gelesen werden. | Schwer |
| SR115664 | Manche Texte werden nicht korrekt in IFC kodiert. | Unbedeutend |
| SR120388 | Door-Operation-Type wird über IfcDoorType in BricsCAD nicht korrekt importiert | Bedeutend |

Einschränkungen des Roundtrips generell, die auf Grund von Einschränkungen des IFC-Standards zustande kommen, sind nicht Teil dieser Dokumentation und werden an anderer Stelle im Zuge der Roundtrip-Präsentation veröffentlicht.

# 6. Erweiterbarkeit

Bei der Entwicklung wurde auf eine sauber Code-Struktur geachtet und wo sinnvoll wurden die Namen und Hierarchien aus IFC eingebaut. Die BricsCAD-API zum Zugriff auf IFC-Daten ist relativ simpel, wenn auch mächtig. Es gibt nur eine Handvoll Funktionen um z.B. auf Attribute zuzugreifen, was eine Erweiterung relativ einfach macht. Der Code selbst wirkt dadurch an manchen Stellen etwas aufgebläht, wenn man die Eigenheiten der API nicht kennt. Das Lesen eines Enum-Attributes als Beispiel:

```
CString AxIFCImportHelper::GetEnumProperty(const Ice::IfcApi::Entity& p)
{
	Variant var;
	ValidateResult res = p.getAttribute(Ice::IfcApi::EnumerationValues, var);
	CString ForRet;
	auto type = var.type();
	if (type == ValueType::eVector) {
		auto&& values = var.getVector();
		for (auto i = 0u; i < values.size(); i++) {
			type = values[i].type();
			if (type == ValueType::eSelect) {
				var = values[i].getSelect().getValue();
				type = var.type();
				if (type == ValueType::eSelect) {
					var = var.getSelect().getValue();
					type = var.type();
					if (type == ValueType::eString) {
						ForRet += var.getString().c_str() +
CString(_T(","));
					}
				}
			}
		}
	}

	ForRet.Trim(_T(","));
	return ForRet;
}
```

So weit wie möglich wurden Funktionen abstrahiert und verallgemeinert, um den Code an verschiedenen Stellen verwenden zu können. Vor allem die Hauptpunkte beim Import und Export sind leicht um neue Elemente (sowohl CAD-Elemente, als auch IFC-Elemente) zu erweitern. Stile und Layer werden beim Export auf Basisklassen-Ebene ermittelt.

# 7. Firmeninformationen

EDV Software Service GmbH & Co KG
Bahnhofstrasse 8, 9500 Villach ÖSTERREICH
**Tel.** 0043 4242 27876
www.ax3000-group.at

Wir, die Firma EDV Software Service GmbH & Co (ESS), sind Spezialisten für die Entwicklung von Gebäude- und Energietechnik-Lösungen, die auf unterschiedlichen Grafikplattformen laufen.

Unser Meisterstück ist AX3000.

Begonnen hatte unsere Geschichte 1978. Unsere Bausoftware entwickeln und verfeinern wir seit mehr als 40 Jahren weiter. Heute zählt ESS mehrere Tausend treue und zufriedene Kunden, welche die Früchte unserer Bemühungen ernten dürfen.

Unser Erfolg beruht auf drei Prinzipien:

**Zusammenarbeit mit Hochschulen**

Der enge Austausch mit Ingenieuren, Architekten und TGA-Spezialisten garantiert, dass unsere Produkte stets auf dem neusten Stand der Wissenschaft und Technologie sind.

**Enger Austausch mit Kunden**

Wir nehmen alle Kundenkontakte persönlich. Denn wir lernen von unseren Kunden sehr viel über die Anwendungspraxis, die Probleme, die Ansprüche. Das Resultat: Extrem effiziente, praxiserprobte Softwareprodukte.

**Denken im Gesamten**

Wir haben von Anfang an eine durchgängige Gesamtlösung angestrebt, die alle Bereiche der Gebäudetechnik und der Energietechnik umfasst. Denn unser oberstes Ziel ist es, die Produktivität unserer Kunden zu maximieren.

Diesen Prinzipien sind wir die ganzen 40 Jahre unseres Schaffens hindurch treu geblieben. Und wir werden auch in den nächsten 40 Jahren danach handeln.

Denn sie haben AX3000 zur führenden Softwarelösung in Europa gemacht.

# 8. Schlusswort

Wir arbeiten seit über 10 Jahren eng mit der Firma Bricsys zusammen und bieten neben diesem Plugin auch eine umfangreiche Lösung für die Haustechnik auf Basis BricsCAD an. Die BricsCAD-API ist unser „zweites Zuhause“. Gute Dokumentation und rasche Hilfe wenn es mal doch zu Problemen kommt kennzeichnen die Firma Bricsys. Dieser Umstand und die hervorragende Zusammenarbeit aller Projektbeteiligten im IFC-Roundtrip-Projekt hat es uns ermöglicht in vergleichsweise kurzer Zeit eine Lösung zu erarbeiten, die sowohl die Anwender als auch Auftraggeber zufrieden stellt.