**ADSDI XMap Serialization Framework (XSF)**

**Purpose**

This document describes the custom coded ADSDI XMap Serialization Framework (XSF).

**Introduction**

ADSDI XMap Serialization is a light weight, custom attribute based technique for constructing business objects from XML (deserialization) and for constructing XML from business objects. The XMap Serialization Framework consists of all code that provides serialization and deserialization services and is located in the Adsdi.GS library mainly in the Global directory.

**Requirements**

XMap serialization requires usage of the XMap custom attribute which is defined in the code file Attributes.cs which is located in the Adsdi.GS code in the Global directory.

Two different custom attributes are used as follows:

1. **XMap** – This is the primary custom attribute used with XMap serialization. This attribute can be applied to any attribute target but is primarily used on classes, properties and constructors. Only one instance of this attribute can be applied to a given target.
2. **XParm** – This custom attribute is used on constructors only and is used to describe the name and source of any parameters required to construct an object.

**Limitations**

XMap serialization currently supports collections based on generic Lists and Dictionaries. Objects may derive from and/or contain these collection types. The intention is to support all simple data types including enumerations. Support for specific simple data types is being implemented as any new unsupported types are encountered.

Current implementation (in progress) requires that all types reside in the Adsdi.GS DLL.

**Principles**

Null objects will return null XML.

Where non-existent XML is encountered (i.e., optional elements or attributes) the framework will not attempt to set property values or construct objects. In these cases the values of the properties (or other elements) are the responsibility of the objects constructor or other methods.

XML element and attribute names for properties will default to the object or property name, but may be overridden by use of the “Name” property of the XMap custom attribute.

**Functionality**

**Storage of Type Information**

The framework will (upon first use) create a list of types having the XMap custom attribute. These types are stored in a collection (so they do not need to be derived from the assembly each time type information is needed). The collection is based on the name of the type or the value of the “Name” property of the XMap custom attribute, if used. This collection of XMap-attributed types is build and referenced in the XmlMapper static class. See the examples below.

Type Referenced by the Type Name

    [XMap(XType = XType.Element)]

    public class Control : ControlBase

The Control object will be built from and will generate an XML element with the name “Control”.

Type Referenced by the Name Property of the XMap Custom Attribute

    [XMap(XType = XType.Element, Name="Task")]

    public class TaskConfig

The TaskConfig object will be built from and will generate and XML element with the name “Task”.

The type information is stored in a static name-keyed collection (name, type) called “Types” in the static XmlMapper class.

Since the “Types” collection is keyed on name, an exception will be thrown if the XmlMapper attempts to load from the assembly where the types reside a type that is being referenced by a duplicate name. This means that duplicate type names, named either with the name of the type or named using the “Name” property of the XMap attribute are not allowed and will cause an exception when the “Types” collection is built. The Types collection is built dynamically the first time the framework attempts to reference a type in the collection.

**Ideas for Possible Implementation**

Support for other collection types.

Post object construction method calls.

Dynamic default object creation (no XMap attributes).