MultiSpeak Version 3.0 Interoperability Assertion

Statement of Interoperable Functionality Between:

Vendors	Products	Product Version	Role	Web Client Interfaces	Web Server Interfaces
Milsoft Utility Solutions, Inc.	Milsoft Integration Server Milsoft DisSPatch Outage Management System	8.1	GIS EA	GIS→STK_Server EA→STK_Server	STK→GIS_Server STK→EA_Server
GeoDigital International	WorkStudio StakeOut	8.2	STK	STK→GIS_Server STK→EA_Server	GIS→STK_Server EA→STK_Server

Summary:

Milsoft's GIS and Engineering products are capable of receiving staked work order data from GeoDigital's WorkStudio StakeOut application using MultiSpeak Web Services. GeoDigital's WorkStudio performs the task of creating a work order with detailed design data based on construction unit standards, existing GIS feature data, ancillary facility data and spatial drafting tools. The Milsoft Integration Server accepts staked work orders in either the designed state or the as-built state and creates a project within the Milsoft data model for the work order. It then generates Milsoft model changes according to the design defined in the work order. This project is then shared with DisSPatch Outage, Milsoft Map and Milsoft EA as a project using Milsoft's shared data bus for Engineering and Operations (E&O).

Since the Field Design is considered as the "System of Record" for field-designed Work Orders, any work order change notifications sent to Milsoft is overwritten with the latest subsequent updates from WorkStudio.

Milsoft's Project Management system creates a Project using the work order number from WorkStudio. In the event that a Milsoft or Milsoft Map client user is performing edits on the same specific project work order that WorkStudio resends, Milsoft will create another project with the same work order number followed by a revision digit number. This is done to prevent reconciliation issues.

Prerequisites:

The Milsoft Integration Server must be accessible to WorkStudio as a resolvable URL and configured in WorkStudio Server with the proper URL, username, and password.

The **workOrder** payload object can contain a very large amount and a very large variety of different objects. How this data is generated is highly configurable within the StakeOut application, which can be used to tailor the data transfer on an individual system installation basis. See the "Configuration Considerations" section below for details.

Certain codes are used to identify specific items such as transformers, conductors, etc. and both systems must be configured with the same set of identity codes.

Specific Vendor Assertions:

1) The WorkStudio Server will send WorkOrders to the Milsoft Integration Server for GIS, EA & OMS.

Importance to user: A design in WorkStudio that details out the changes to be made or were made to an electrical distribution network can be sent to the GIS, EA and OMS systems, so that those systems can properly reflect the distribution circuit changes without having to manually draft/edit the electrical model changes in those systems.

How Achieved: Within WorkStudio, either an automation or an action is configured to send the MultiSpeak workOrder object to the Milsoft Integration Server. Once the export is initiated, WorkStudio will first call the GetMethods method so that it can discover which methods the receiver can support. For this interface, the MultiSpeak method names can be different depending on whether the receiver used the buss-based MultiSpeak WSDLs or the point-to-point-based MultiSpeak WSDLs. As long as the receiver responds with either the WorkOrderChangeNotification or the WorkOrderChangedNotificationToGIS method names, then WorkStudio will send the workOrder using the discovered method.

These events are either initiated from a user selecting a work order and clicking a button from the user interface, or automatically by an automation event configured in the WorkStudio workflow engine. These events can be executed either from within WorkStudio Editor or within WorkStudio Server, as the executable is the same in either case. However, the URL of the Milsoft Integration Server must be accessible from the process that is calling the Web Service.

Products: Milsoft Integration Server and GeoDigital WorkStudio StakeOut

Summary of Interoperability Test Results
STK→GIS
Table 1
Recommended MultiSpeak Methods

Method Name	Importance to User	Supported by Server[1] (GIS and EA)	Supported by Client[2] (STK)	Verified Inter-operable[3]
PingURL	Is used to discover if the client can successfully communicate with the server, usually during configuration.	X	X	X
GetMethods	Requests the method names served by the server. Is used by StakeOut to discover if the server supports the WorkOrderChangedNotificationToGIS method or the WorkOrderChangedNotification method.	X	X	X
WorkOrderChangedNotificationToGIS	This method call will pass an entire list of WorkOrder objects from the staking system to the GIS system. This method is defined by the point-to-point WSDL.		X	
WorkOrderChangedNotification	This method call will pass an entire list of WorkOrder objects from the staking system to the GIS system. This method is defined by the buss WSDL.	X	X	X

Table 2
Optional MultiSpeak Methods

Method Name	Importance to User	Supported by Server[1] (GIS and EA)	Supported by Client[2] (STK)	Verified Inter-operable[3]
GetDomainMembers	Requests the members of a given domain (type of fixed information, such as all of the counties in the database).	X		
GetDomainNames	Requests the domains (lists of fixed information, such as the counties served, or the acceptable status codes for this installation).	X		
GetFeatureTypes	This method will retrieve a list of material types from the GIS.			

GetModifiedFeatures	This method will retrieve a list of features from the GIS that have changed since a previous version.			
GetFeaturesByBounds	This method will retrieve a list of features from the GIS that are contained within a certain bounds (envelope).			
GetFeaturesByType	This method will retrieve a list of features from the GIS that are of a certain feature type.			
GetFeaturesByTypeAndBounds	This method will retrieve a list of features from the GIS that are of a certain feature type and within a certain bounds.			
GetFeatureByTypeAndObjectID	This method will retrieve a feature from the GIS that has a certain objectID.			
GetAllBackgroundGraphics	Returns all background graphics. The calling parameter lastReceived is included so that large sets of data can be returned in manageable blocks. lastReceived should carry an empty string the first time in a session that this method is invoked. When multiple calls to this method are required to obtain all of the data, the lastReceived should carry in subsequent calls the objectID of the data instance noted by the server as being the lastSent. If the sessionID parameter is set in the message header, then the server shall respond as if it were being asked for a GetModifiedXXX since that sessionID; if the sessionID is not included in the method call, then all instances of the object shall be returned in response to the call.			
GetModifiedBackgroundGraphics	Returns all background graphics that have been modified since the previous session identified. The calling parameter previousSessionID should carry the session identifier for the last session of data that the client has successfully received. The calling parameter lastReceived is included so that large sets of data can be returned in manageable blocks. lastReceived should carry an empty string the first time in a session that this method is invoked. When multiple calls to this method are required to obtain all of the data, the lastReceived should carry in subsequent calls the objectID of the data instance noted by the server as being the lastSent.			
GetAllSpatialFeatureGroups	Returns all spatial features groups. Spatial feature groups are groups of spatially related features, such as all of the features attached to a pole. This is an alternative means to send GIS data, rather than sending individual features by type. The calling parameter lastReceived is included so that large sets of			

	data can be returned in manageable blocks. lastReceived should carry an empty string the first time in a session that this method is invoked. When multiple calls to this method are required to obtain all of the data, the lastReceived should carry in subsequent calls the objectID of the data instance noted by the server as being the lastSent.If the sessionID parameter is set in the message header, then the server shall respond as if it were being asked for a GetModifiedXXX since that sessionID; if the sessionID is not included in the method call, then all instances of the object shall be returned in response to the call.			
GetModifiedSpatialFeatureGroups	Returns all spatial features groups that have been modified since the session identified by the sessionID parameter. The calling parameter previousSessionID should carry the session identifier for the last session of data that the client has successfully received. Spatial feature groups are groups of spatially related features, such as all of the features attached to a pole. This is an alternative means to send GIS data, rather than sending individual features by type. The calling parameter lastReceived is included so that large sets of data can be returned in manageable blocks. lastReceived should carry an empty string the first time in a session that this method is invoked. When multiple calls to this method are required to obtain all of the data, the lastReceived should carry in subsequent calls the objectID of the data instance noted by the server as being the lastSent.			
GetBackgroundGraphicsByBounds	Returns background graphics within a rectangular bounding box. The calling parameter lastReceived is included so that large sets of data can be returned in manageable blocks. lastReceived should carry an empty string the first time in a session that this method is invoked. When multiple calls to this method are required to obtain all of the data, the lastReceived should carry in subsequent calls the objectID of the data instance noted by the server as being the lastSent.			
GetSpatialFeatureGroupsByBounds	Returns spatial feature groups within a rectangular bounding box. Spatial feature groups are groups of spatially related features, such as all of the features attached to a pole. This is an alternative means to send GIS data, rather than sending individual features by type. The calling parameter lastReceived			

	is included so that large sets of data can be returned in manageable blocks. lastReceived should carry an empty string the first time in a session that this method is invoked. When multiple calls to this method are required to obtain all of the data, the lastReceived should carry in subsequent calls the objectID of the data instance noted by the server as being the lastSent.			
GetSpatialFeatureGroupByObjectID	Returns a specific spatial feature group by objectID.			
ModifyGISFeatureData	Allow requester to modify GIS data for any GIS feature. GIS returns information on failed transactions by returning an array of errorObjects.			
MaterialManagementAssemblyNotification	This method will send changes in a MaterialManagementAssembly in the Staking System to the GIS System.			

1) Supported by Server means that the server has demonstrated in some interoperability test (not necessarily with this client) that it can support the method.
2) Supported by Client means that the client has demonstrated in some interoperability test (not necessarily with this server) that it can call the method.
3) Verified Interoperable means that both the client and server have demonstrated in this interoperability test that they can usefully transfer data using this method.

GeoDigital Configuration Considerations:

When StakeOut is creating a MultiSpeak **WorkOrder** object to pass to GIS, it is essentially converting a Station & Assembly-based model into a Feature-based model. The assemblies applied to the design work order are typically defined in the accounting system and sent to StakeOut to use within a design work order. StakeOut must be configured to properly define how those assemblies that came from the accounting system should be treated physically, spatially, and electrically so that they can be properly used for designs as well as properly modeled within a MultiSpeak **WorkOrder** object to pass to a GIS/EA system.

WorkStudio to GIS Preferences

- "Enforce Field Size Limits to Schema Defined Sizes" can be checked or unchecked.
- Overhead Line Direction must be set to "From Station to Headspan Station" for proper import into WindMilMap.
- "Export All Station Data to MultiSpeak Extensions" must be checked if user-defined data needs to be sent from StakeOut on station objects. If not, leaving it unchecked will create a smaller MultiSpeak payload.
- "Export All Unit Data to MultiSpeak Extensions" must be checked if user-defined data needs to be sent from StakeOut on assembly objects. If not, leaving it unchecked will create a smaller MultiSpeak payload.

Web Services Preferences

The Default MultiSpeak Web Services Client Connection to GIS must have the following values set:

Host – This is the full URL to the Milsoft Integration Server (MIS) such as "http://192.168.1.100:83/soap/EA_Server".
UserID – This is a username defined by the Milsoft Integration Server environment to authenticate communicating with the MIS.
Password – This is the password defined by the Milsoft Integration Server environment to authenticate communicating with the MIS.

The **Work Order Export…** button will allow the administrator the ability to configure transforming specific fields of data as the StakeOut design job is transformed into the MultiSpeak **WorkOrder** object. Clicking this button will provide field aliasing options and field scripting options. By default, these do not need to be configured unless advanced data transformation is needed on a case-by-case basis.

MultiSpeak Nouns Associated to Assemblies

Each **MaterialManagementAssembly** that is received from the accounting system must be configured in StakeOut in order to be used. At a minimum, setting the **UnitType** of the unit will define how that unit can be used, and thus, its data model and behavior. For example, a pole unit behaves very differently than a conductor unit. Within StakeOut, any number of **UnitTypes** can be defined based on the preferences of the utility, but each **UnitType** that is defined will have an associated **UnitClass**. The **UnitClass** defines the core behavior for each of the units of the same unit class.

Each **UnitType** that is defined has this associated **UnitClass**, but it also may have certain behavior options that can be turned on, turned off, or configured for all the units of the same **UnitType**. For example, StakeOut can have a **PrimaryPole UnitType** and a **SecondaryPole UnitType**. Both of these

UnitTypes use the same **UnitClass** called "**TPoleUnit**", but the **SecondaryPole** units may be configured to have a different map symbol than the **PrimaryPole**.

StakeOut must be configured to define which MultiSpeak objects (nouns) the units are to be exported as when applied into a MultiSpeak **WorkOrder** object that is being sent to a GIS or FA system. Some **UnitClasses** within StakeOut have one and only one association to a MultiSpeak noun. For example, the **TPoleUnit** class in StakeOut can only be considered a MultiSpeak Pole object.

However, other **UnitClasses** within StakeOut can be used to model many different MultiSpeak nouns. Also, one **UnitType** defined in StakeOut can be used to model many different MultiSpeak nouns. For this reason, all units that are in the StakeOut unit database can be directly assigned an associated MultiSpeak noun type. Further, some MultiSpeak noun types are not fully specific and need to have an additional sub-type defined, such as Breaker, Fuse or Recloser in the case of the OvercurrentDeviceBank.

The following is a list of the possible MultiSpeak nouns that can be configured in StakeOut to send to a GIS system. These associations need to be set in the Units Configuration within StakeOut, under the MultiSpeak preferences.

MultiSpeak Noun	**StakeOut Supported (Auto-Association by UnitClass or Manually Configured)**	**Milsoft Supported Element Types**
capacitorBank	Manual – TconstructionUnit[1]	Capacitor
overcurrentDeviceBank	Manual - TConstructionUnit	Overcurrent Device
measurementDevice	Manual - TConstructionUnit	Marker
meter	Manual - TConstructionUnit	Updates the meter number on a Consumer
outageDetectionDevice	Manual - TConstructionUnit	N/A
switchDeviceBank	Manual - TConstructionUnit	Switch
primaryCabinet	Manual - TConstructionUnit	SwitchGear
regulatorBank	Manual – TconstructionUnit[2]	Regulator
transformerBank	Auto - TransformerUnit	Transformer Bank
serviceLocation	Manual - TConstructionUnit	Consumer
riser	Manual - TConstructionUnit	Assembly
pole	Auto - TPoleUnit	Map point Pole
streetLight	Manual - TConstructionUnit	Map point Street Light
surfaceStructure	Manual - TConstructionUnit	Assembly
secondaryJunctionBox	Manual - TConstructionUnit	Assembly
substation	Not Supported	Source
powerSystemDevice	Manual - TConstructionUnit	N/A
ohPrimaryLine	Auto - TConductorUnit (based on UnitType's Kind=Primary & Construction=Overhead)	Overhead Line
ugPrimaryLine	Auto - TConductorUnit (based on UnitType's Kind=Primary & Construction=Underground)	Underground Line

ohSecondaryLine	Auto - TConductorUnit (based on UnitType's Kind=Secondary & Construction=Overhead)	Overhead Line
ugSecondaryLine	Auto - TConductorUnit (based on UnitType's Kind=Secondary & Construction=Underground)	Underground Line

1. Capacitor Banks can be modeled in StakeOut but the embedded Capacitor objects cannot.
2. Regulator Banks can be modeled in StakeOut but the embedded Regulator objects cannot.

Geometry Considerations

Both WorkStudio and Milsoft must be using the same defined coordinate system.

The Direction of conductor geometry is important to Milsoft and thus the Overhead Line Direction preference in StakeOut must be set to "From Station to Headspan Station".

Bulges in the curved polyline segments can be modeled in StakeOut for any polyline object, such as underground secondary and underground primary conductors. These bulges represent a curving factor from one point of a line segment to the previous point of the line segment. WorkStudio will export these bulge factors but Milsoft will ignore them. The net result is that the line segment will be straight instead of curved.

Within StakeOut, the designer has the ability to draft some objects, while other objects are automatically assigned geometry, given the parent object that owns it (such as a transformer hanging on a pole). When geometry can be drafted by the designer, they can choose to snap the end-points of the geometry to other objects or not. In the case where some geometry is not snapped, or it is drawn in a location that is off from where it should be drawn, it may produce undesirable visualizations in Milsoft because connectivity takes precedence over geometry.

Electrical Connectivity Considerations

When a workOrder MultiSpeak object is imported into Milsoft, Milsoft is updating its connectivity model. The connectivity model is section-based, where each section may optionally have a parent-section and that parent-section defines the section that provides electrical current to the current section. Milsoft can import MultiSpeak WorkOrder objects that do not have connectivity defined and will generate corresponding line-section objects that are not connected to any other line-section objects. StakeOut can generate the parentSection identifiers on each of the objects in the ObjectList contained within the WorkOrder object as the WorkOrder object is being created to export out of StakeOut. It does this based on the UnitClasses' behaviors and the UnitTypes' configurations. In some rare cases, there could be more than one possible ParentSection ID that could be automatically assigned to one of the objects generated by StakeOut. In this case, StakeOut will choose one of them, given the best-guess logic coded into StakeOut. Where StakeOut incorrectly generates a parent line-section ID, the end-users in Milsoft will need to change the connectivity to be correct after importing the WorkOrder.

Milsoft Modeling Rules

Terms

External System - A GIS and or Staking system sending connectivity data to Milsoft. In this case GeoDigital's Stakeout.

GUID – Globally Unique Identifier – This is a random computer unique generated identifier. It is used as an element ID in Milsoft.

Loose vs. Tight Integration

When the user creates a new circuit element in Milsoft, that element is given a **globally unique identifier** (**GUID**) which Milsoft uses to identify it in our database. The user never sees this GUID. Instead, the user identifies a circuit element by its human readable element name. The user may change the name of an element as often as he likes, but he cannot change the element's GUID. Circuit element names in Milsoft can be no longer than 25 characters and must be unique in the model. It is common for Milsoft users to transfer an entire feeder to a different model, make changes to the elements on that feeder (including renaming elements), and then transfer the entire feeder back to the original model. Because element names may be changed by the user, Milsoft must use the GUID to uniquely represent each element and to properly link elements in the source model to elements referencing them in the destination model.

An External System interfacing with Milsoft may either use loose or tight integration. Loose integration means that elements are referenced by their element names. Tight integration means that elements are referenced by a GUID. If the External system uses tight integration and gives each element a GUID, Milsoft will persist that GUID in its database. This makes it possible to maintain the link between elements in Milsoft and elements in the External System even when the user has changed the element names. MultiSpeak 2 and 3 provide the capability of sending incremental changes rather than a full dump of the model. However, in order for this to work we must have some way of knowing which elements in the incremental change packet correspond to which elements in the existing Milsoft model. If elements are referenced by an unchangeable GUID this is safe; if elements are referenced by a changeable element name, it becomes unreliable since names can change where GUID's do not.

If an External System will always update the Milsoft model with a full dump, loose integration is safe to use. However, if the External System is to take advantage of the speed and flexibility of incremental updates, tight integration is recommended. A third option would be to use loose integration and to inform the user that if he changes any element names in Milsoft or his External system, those elements will not be properly updated between systems.

Resolving Name Collisions

Milsoft will always check to see if an element already exists before deciding how to import it. If an element in the XML file has a name that does not yet exist in the model, it will always be added to the model with the given name. However, if the element name already exists in the model, Milsoft will take one of three different actions. The user decides in advance which action will be taken by selecting one of the following choices from a combo box on the import dialog:

If a duplicate element name is found…modify the existing element's data

If this option is selected, Milsoft will apply the data from the element in the XML file to the existing element in the model. If some of the XML fields are empty, those fields in the model be left unmodified; they will not be zeroed. This selection is the default and represents the way the user would most likely expect to the system to work.

If a duplicate element name is found…ignore the new element

This is useful if you are not able to determine during your export which elements have been newly added. Milsoft can make this determination for you by refusing to import any elements that already exist in the model and only importing new elements.

If a duplicate element name is found…import but rename the new element

Some External Systems are not capable of insuring the uniqueness of element names. This option allows them to import a new feeder or set of elements into an existing Milsoft model with the guarantee that any existing elements that happen to have the same name will not be trampled. Milsoft will automatically rename duplicates by affixing a ".1" to the name of the new element. If it finds that that element name is also taken, it will affix a ".2", and so on until it finds an unused name.

Incremental Updates

Milsoft allows both batch and incremental importing of MultiSpeak XML files as well as real time updates using MultiSpeak. In most cases, the user desires to import a full dump of the entire circuit model into Milsoft. However, it is also very useful to be able to import incremental changes to the model. Suppose the user has only made changes to (or added or deleted) five elements in the GIS system and would like Milsoft to reflect those changes. It is possible to generate an XML file containing only those five elements that changed and import them into an existing Milsoft model.

The MultiSpeak XML file used in an incremental update is identical to the file used in a complete dump except that the ***documentType*** attribute of the *MultiSpeak* tag should be set to *incremental* rather than *dump*. When you set the ***documentType*** to *incremental*, Milsoft will require you to import into an existing model. In order to create a new model, you must use *dump*. Also, when *dump* is used and you are importing on top of an existing model, Milsoft will ask you whether or not the existing model should be overwritten. If you choose to overwrite the existing model, the old model will be discarded and the

elements in the XML file will be imported as a full dump. If you choose not to overwrite the existing model, Milsoft will perform an incremental update and treat the elements in the XML file as modifications to the existing model. On the other hand, when *incremental* is used, you are not asked whether or not you would like to overwrite the existing model. It is just assumed that the exported elements should be treated as modifications to the existing model. So, you can still do incremental updates even if the document type is set to *dump*, but by using the *incremental* keyword you can avoid the possibility of accidentally replacing your entire model with just the incremental changes.

Every XML element derived from *mspObject* has a *verb* attribute that specifies how the importing system should handle it in the case of an incremental update. If you wish to inform Milsoft that a certain element should be deleted from the model, set the *verb* attribute to *Delete*. Otherwise, set the attribute to *New* or *Change*. Ultimately, Milsoft will decide for itself whether an element is new or changed, so right now Milsoft makes no distinction between these two keywords. However, we may find a reason in the future to make use of this distinction. So, if you are able to determine whether an element has been newly added since the last export to Milsoft, go ahead and set the verb to *New*.

It is up you, the exporter, to remember what element changes you have sent to Milsoft so that you can determine which elements to send in the next incremental update XML file and what their verbs should be.

ObjectID vs. SectionID

Milsoft can import all MultiSpeak nouns which are derived from **mspConnectivityPoint** and **mspConnectivityLine**. Milsoft supports the sectional model, but does not support the nodal model. Therefore, all nouns imported by Milsoft must have an ***objectID,*** a ***sectionID,*** and a ***parentSectionID*** as described by the MultiSpeak 3 XML schema. The contents of these three items differ depending on whether loose or tight integration is being used.

If tight integration is desired (referencing by GUID)…

> … the ***objectID*** attribute should be the GUID representing this circuit element.
>
> … the ***sectionID*** tag should contain the (human readable) circuit element name.
>
> … the ***objectID*** attribute of the ***parentSectionID*** tag should be the GUID representing the element's parent.

If loose integration is desired (referencing by element name)…

> … the ***objectID*** attribute should be the (human readable) circuit element name.
>
> … the ***sectionID*** tag should also contain the circuit element name.
>
> … the *name* attribute of the ***parentSectionID*** tag should be the circuit element name of the element's parent.

Element Connectivity

The connectivity of elements in a Milsoft circuit model is based entirely on the direction of current flow through the system. If current flows from element A to element B then element A is said to be the parent of element B. Throughout this document the notation: A ← B will mean "A is the parent of B". Substations (and elements that are disconnected from the circuit) have no parent, but it is customary to use "ROOT" as the parent name when none exists. Also, the Milsoft model is assumed to be a tree, which means that each element has only one parent but each element can have many children. (Multi-parent nodes are the one exception to this rule.)

Notice that, unlike in a nodal model, the parent-child linkage of the circuit elements reverses when a backfeed operation is performed because the direction of current flow has changed. Consider the following model consisting of two sources and some overhead lines.

SRC1 ← OH1 ← OH2 OH3 → SRC2

Suppose the user disconnects OH1 from SRC1 and backfeeds OH2 so that all three overhead elements are fed from from SRC2. WindMil would change the parent-child linkages to:

SRC1 OH1 → OH2 → OH3 → SRC2

Phasing Rules

It is important to export the *phaseCode* for every element. If Milsoft finds an element with no *phaseCode* it will import the element and make an educated guess about its phasing, usually setting it to the phasing of its parent. The general rule in Milsoft is that an element may not have any phases that its parent does not have. For instance, it is illegal to parent an ABC consumer to an AB transformer. The only exceptions to this general rule are certain types of transformers (like Open Y-D or Y-D One) which can have more phases than their parent. Milsoft will automatically correct any phasing errors it detects during the import, so it will never make a model with illegal phasing. The warnings log will contain a report of all rephased elements after the import finishes.

Equipment Names

Some tags in the MultiSpeak XML are meant to contain the names of equipment. For instance the *conductorType* tag in the *conductorList* for *ohPrimaryLine* should contain the name of the conductor on a particular phase of that overhead line element. Defining the electrical properties of the various pieces of equipment is the responsibility of the engineer using Milsoft and is not expected of a GIS system. However, it is important to know that these equipment names are not simply for display purposes in Milsoft. They actually provide a link to a piece of equipment with that name in Milsoft's equipment database. So, while it might seem like a good idea to change "4_ACSR" to "4 ACSR" in a GIS system, such a change would break the equipment links in the engineering model until the Milsoft user reassigned all the affected equipment.

When an element is encountered during the import which references equipment that does not yet exist in the equipment database, the equipment is automatically created in the model's equipment database. However, the newly created equipment has no electrical properties defined, so the Milsoft user must enter the equipment properties manually, import them from another equipment database, or rename previously defined equipment to match the equipment name maintained by the GIS system.

The equipment name can be no more than 20 characters in length. If it is longer than 20 characters, Milsoft will truncate it before searching the equipment database for that equipment.

Multi-parent nodes

MultiSpeak allows multiple parentSectionIDs for each element. This means that a circuit element can have up to three different parents at the same time. This is useful in Milsoft for tying together three single-phase taps to form a three-phase line. A multi-parent situation is modeled in Milsoft using a multi-parent node. Multi-parent nodes are the only entities in Milsoft that are allowed to have more than one parent. For instance, if three single-phase underground lines must come together to form a three-phase underground line, a multi-parent node must exist at that junction.

In MultiSpeak 1.1, the multi-parent node had to be explicitly modeled. In MultiSpeak 2 and later, however, it is only necessary to set multiple parentSectionIDs for the downline element being fed by multiple parents. Milsoft will insert the multi-parent node automatically during the import process.

It is important to note that multi-parent nodes should be used with discretion. The only appropriate usage of a multi-parent node in Milsoft is to bring single-phase taps together to parent a multi-phase line. While it might be tempting to use multi-parent nodes for other clever purposes, such as tying together three single-phase overcurrent devices, regulators, or transformers, you should not do this. Milsoft provides a way to model each of these without using multi-parent nodes.

Map Points

Everything Milsoft draws on the screen is called a **feature**. A feature can be either a **circuit element** or a **map point**. Circuit elements are features that have electrical connectivity (overhead lines, transformers, overcurrent devices, etc…). Map points are features that have no electrical connectivity (poles, pads, pedestals, junction boxes, etc…). For historical reasons, if you want to export a map point to Milsoft you can use either of these MultiSpeak objects: pole or *genericPointFeature*. Milsoft makes no distinction between the two and they are imported identically.

Only the following fields from the pole or *genericPointFeature* are used by Milsoft:

objectID

The objectID identifies the map point in the circuit model. The objectID may contain either a GUID (which uniquely represents the map point in the circuit model) or it may contain the name of the map point. (Map point names may not exceed 32 characters.)

If the objectID is a valid GUID, then the map point name is read from the *facilityID* tag instead. If a map point with the same GUID already exists in the circuit model, that map point will be overwritten with the newly-imported name and map point data. (See the discussion of Loose vs. Tight Integration earlier in this document.)

If the objectID is not a valid GUID (which means it will be treated as a map point name) then if a map point with the same name already exists in the circuit model, it will be overwritten with the newly-imported map point data.

If the objectID is left blank, and a map point of the same type already exists at the exact location specified by the *mapLocation* tag, that map point will be overwritten with the newly-imported map point data.

facilityID
If the objectID tag contains a GUID, *facilityID* is used to specify the human-readable name of the map point. If you are using loose integration, the map point name should go in the objectID field instead. The map point name is read from *facilityID* by default, but this is configurable in Milsoft's import options.

poleUse or featureType or extensions/structureType
This tag contains the type of the map point. The map point type can be one of the following strings:

- Pole
- Tower
- Pad
- Pedestal
- Junction Box
- Enclosure
- Vault
- PullBox
- Foreign Structure
- Marker

Two map points of the same type may not have the same map point name. However, a map point may have the same name as a circuit element or a map point of a different type.

The XML tag that we pull the map point type from is configurable by the user in the import options. Milsoft will look for the map point in the following places (in order) until it finds something:

1) Whatever XML tag the user entered in the "Map Point Type" field on the "pole" tab of the "MultiSpeak Field Mappings" page of the import wizard. This page only appears when the user chooses the "Explicitly map MultiSpeak XML tags" option.
2) ***The extensions/structureType tag of a pole or genericPointFeature***.
3) The poleUse tag (a valid child of pole in the MultiSpeak XML schema)
4) ***The featureType tag (a valid child of genericPointFeature)***
5) If no map point type is found in any of these places, the map point type defaults to "Pole".

mapLocation
The X,Y location of the map point. This location is transformed to the appropriate coordinate system in the same way the coordinates of the circuit elements are transformed.

If the map point already exists in the model and it shares an endpoint or mid-point with a line element, that point of the line will be moved to the new location of this map point. This

behavior is configurable with the import option called “Move attached line if import moves map point”.

There is another import option that will snap existing lines to a map point if the line is a certain configurable distance away from the map point.

Assemblies

Map points in Milsoft can have a list of assemblies associated with them. Circuit elements cannot. The assembly list on a map point represents the various pieces of physical equipment that are connected to that map point. Some of these pieces of equipment (transformers, capacitor banks, etc.) will represent circuit elements in the Milsoft circuit model. Other pieces of equipment (guys, anchors, pole top assemblies, etc.) are not represented anywhere else in the Milsoft circuit model; they exist only as assemblies attached to a map point.

In order to eliminate duplication of data, Milsoft is very careful to make a distinction between **element assemblies** and **non-element assemblies**. An element assembly is an assembly that also exists as a circuit element in the model). A non-element assembly is an assembly that does not correspond to any Milsoft circuit element.

Milsoft imports and stores non-element assemblies on the map points they are associated with. The Milsoft user can use the Map Point Editor to examine and edit the various assemblies on that map point.

Milsoft ignores element assemblies during the import unless those element assemblies are marked for retirement. If an element assembly is marked for retirement in the import, the corresponding circuit element will be deleted.

Currently, the only way to import assemblies via MultiSpeak is to import them as part of a work order. There is no other way in MultiSpeak to directly import the assemblies associated with a map point. If you need to do this you should use the .ASM import instead.

Importing Work Orders

The MultiSpeak XML schema describes a *workOrder* tag, which Milsoft can use to import work orders from a staking system. The *workOrder* tag can appear in a MultiSpeak XML document as a first-level child of the MultiSpeak tag. When the batch import is used this way, it is typically manually imported into Milsoft using WindMil's import dialog. The *workOrder* tag may also be sent in a WorkOrderChangeNotification, using the web services interface. Milsoft will import the work order the same way, regardless of how it receives it.

When a work order is imported, a new project will always be created in Milsoft's project manager. This new project gets its name from the *woNumber* tag in the work order. After the import is finished, this project will contain a complete list of the modeling operations that were performed by importing the work order. By default, this new project will appear in a shared folder in the Project Manager named From Staking Note: This is configurable in the import settings of the Milsoft Integration Server.

If a project with that name (*woNumber*) already exists and it hasn't been posted yet, the previous contents of the project will be replaced by the current import. If a project with that name has already been posted (or for some other reason is not editable by the Milsoft Integration Server), a numeric suffix (woNumber.1, woNumber.2, etc.) will be affixed to the work order name.

Milsoft recognizes only these children of the *workOrder* tag:

woNumber: Used as the name of the project the work order is imported into.

jobDescr: Imported into the description field of the newly-created project.

stationList: A list of the stations in the work order, described in more detail below.

extensionsList: Milsoft currently supports one MultiSpeak extension to the workOrder tag. The stakingSheetUrl extension is used to specify the URL of a staking sheet that is associated with this work order.

Here is an example of a work order tag.

```
<workOrder>
  <woNumber>12345</woNumber>
  <jobDescr>A test work order</jobDescr>
  <extensionsList>
    <extensionsItem>
      <extName>stakingSheetUrl</extName>
      <extValue>http://StakingServer:80/Job-12345.pdf</extValue>
    </extensionsItem>
  </extensionsList>
  <stationList>
        ... A sequence of station tags ...
(See SmallSample.xml for a more detailed example.)
  </stationList>
</workOrder>
```

Stations

Each workOrder contains one or more station objects in a *stationList*. Milsoft imports the following tags from each station object:

objectList

The object list contains a list of circuit elements or map points to be imported into the circuit model. The portion of this document entitled "Milsoft Circuit Element Types" describes different types of objects Milsoft is capable of importing. The very same routines are used to import these elements from a work order as if they had been imported directly from a GIS import, so all the information in this document about importing these element types is applicable for a work order import as well.

Milsoft can only associate assemblies with map points, not with circuit elements. So, in order to properly import any non-element assemblies in the assemblyList, there needs to be at least one map point in the objectList to attach them to. All the assemblies in the assemblyList will be associated with one map point (pole or *genericPointFeature*) in the objectList. If there are multiple map points in the objectList we will use the last one.

objectID

The *objectID* of the station is usually ignored. However, if there are no map points in the objectList, we will search for an existing map point whose GUID matches the *objectID* of the station. If such a map point exists in the model, all of the assemblies in this station's assemblyList will be associated with that map point.

mapLocation

The *mapLocation* of the station is usually ignored. However, if any of the map points in the objectList have a missing coordinate, those map points will be placed at the location specified by the *mapLocation* of the station.

assemblyList

The assembly list may contain a list of element assemblies and non-element assemblies. For each assembly in the list, the following tags are recognized.

- *objectID* – If a GUID *objectID* is provided for the assembly, Milsoft will assign that GUID to the assembly when it creates it. That GUID will then appear in any future exports of the assemblies. If no objectID is provided for the assembly, Milsoft will create one.

- *featureID* – If an *assembly* has a *featureID* tag containing the GUID of a circuit element or map point, that assembly will be treated as an element assembly. This means that it will not appear in the list of assemblies in the Map Point editor. If an element assembly is marked as being retired (unitActn='R'), then the circuit element that this assembly is associated with will be deleted by the import.

- *unitCode* – This is the human-readable name of the assembly (e.g. 35-5 or E-105). It is used to look up the appropriate assembly equipment in Milsoft's equipment database (EQDB). If no assembly with this name already exists in the EQDB, it will be automatically created during the import.

- *unitActn* – This may contain a value of either C, E, or R.
  - C = Construction. Specifies that this assembly was newly added.
  - E = Existing. Specifies that this assembly already existed.
  - R = Retired. Specifies that his assembly should be removed.

  Note: Milsoft does not support incremental changes to assembly lists. If an assembly list is provided for a map point, it will always replace the existing assembly list on that map point. So, the C and E unitActn codes are treated the same by Milsoft. We make no distinction between new and existing assemblies. If the unitActn is R (Retired), the assembly is ignored unless it is an element assembly (has a featureID). If the assembly to be retired has a featureID, Milsoft will delete the associated circuit element from the circuit model.

Note: In order to abide by the MultiSpeak specification, distance, angle, comments, and damaged should be children of the extensions tag on assembly. However, the Milsoft importer will still import these tags correctly even if they are direct children of assembly.

Milsoft Circuit Element Types

Milsoft can import all MultiSpeak nouns which are derived from mspConnectivityPoint and mspConnectivityLine. The following list indicates which Milsoft element types are created when these nouns are imported.

MultiSpeak Noun	Milsoft Element Type
ohPrimaryLine	Overhead
ohSecondaryLine	Overhead
ugPrimaryLine	Underground
ugSecondaryLine	Underground
fakeNodeSection	Node
capacitorBank	Capacitor
overcurrentDeviceBank	Overcurrent Device
switchDeviceBank	Switch
regulatorBank	Regulator
transformerBank	Transformer
serviceLocation	Consumer
substation	Source
generator	Generator
motor	Motor

Each of these elements types will be discussed in more detail.

Data Common to All Elements

phaseCode	Every element must have a phase code.
gridLocation	This appears in the map number field in Milsoft.

Overhead and Underground Elements

Milsoft makes no distinction between primary and secondary lines. To Milsoft, an overhead is an overhead. Although Milsoft can import ohSecondaryLine and ugSecondaryLine, it treats them exactly as it would a primary line.

There are quite a few tags for every element that Milsoft does not use. Here is a list of the tags for overhead and underground lines that Milsoft uses.

conductorList	Make sure each conductor in the list has a phase defined.
condN	Equipment name of the neutral conductor.
condLength	Impedance length of the conductor (in feet).
constr	Name of a user-defined construction code in Milsoft.

load	Actual kW demand goes here if it is known. For other billing load data use the *billingAccountLoad* noun.

Node Elements

As of Milsoft version 7, consumers are no longer a type of node; consumers are now their own separate element type. Consumers may still be imported using a *fakeNodeSection* whose *cktLvl* is set to 6, but this is not recommend as of MultiSpeak 2. Instead use the *serviceLocation* noun.

You may still use *fakeNodeSection* to create a multi-parent node, but it is not necessary as of MultiSpeak 2. Now that MultiSpeak allows elements to have more than one *parentSectionID*, Milsoft will automatically insert a multi-parent node when it finds an element with more than one *parentSectionID*.

The *fakeNodeSection* is no longer necessary for creating a feeder bay as of MultiSpeak 2. In the past, Milsoft modeled feeder bays as nodes whose circuit level was set to “feeder bay”. Now Milsoft models feeder bays using overcurrent devices.

The *fakeNodeSection* is still useful for modeling spot loads, though. Set cktLvl to 4 to specify that the node is a spot load.

Capacitor Elements

Milsoft uses only the following fields specific to *capacitorBank*:

connectionCd	The choices are enumerated in the MultiSpeak XML schema.
swType	The choices are enumerated in the MultiSpeak XML schema.
swStatus	The choices are enumerated in the MultiSpeak XML schema.
swOn	Value in volts or amps at which this capacitor switches on.
swOff	Value in volts or amps at which this capacitor switches off.
cntrCkt	The *name* attribute of this tag contains the name of the circuit element that will control capacitor switching.
bankKvar	Total bank size, in kVAR. If *bankKvar* contains a value, Milsoft will divide the kVAR equally among the existing phases of the capacitor element.
volts	Rated voltage of capacitor in kV. Voltage is L-G if capacitor is Wye connected and L-L if delta connected.
capacitorList	If a *capacitorList* exists, Milsoft will read and store the kVAR information for each phase from this list; however, the objectIDs

of the individual capacitors in the bank are not persisted in Milsoft. If the GIS system sends an incremental update in which the objectIDs of the capacitors in the *capacitorList* have changed, Milsoft will not know or care. The only reason to use the *capacitorList* is if the capacitor bank contains individual capacitors with different kVAR ratings. If all kVAR ratings are the same, it is much simpler to place the total kVAR for the bank in the *bankKvar* field and let Milsoft distribute the kVAR equally over the existing phases. The *phase* field in the *capacitor* tags in the *capacitorList* should contain only "A", "B", or "C".

Overcurrent Device Elements

Milsoft uses only the following fields specific to *overcurrentDeviceBank*:

facilityID Used as an alias name for the device and will appear in Milsoft's work environment as the name of the feeder (if this device is a feeder).

isGanged If "true", Milsoft will enforce the fact that all phases of the device open and close together.

mspOvercurrentDeviceList Described below

An overcurrent device in Milsoft is actually a bank element, so the user is allowed to specify overcurrent device equipment for each phase of the bank. The *mspOvercurrentDeviceList* should indicate which device equipment each phase of the bank is using. In order to do this it is important to correctly populate the *eaEquipment* and *phase* tags of each device in the list. If a device in the list has no *phase* indicated, Milsoft will not use it. The *eaEquipment* tag should contain the name of an actual piece of device equipment in Milsoft's equipment database. If the specified equipment is not found in the equipment database, Milsoft will automatically create it. If a new fuse with a *linkRtg* is found in the device list, Milsoft will set the current capacity of that fuse in the equipment database when it initially creates the equipment. If the fuse already exists in the equipment database, however, Milsoft will not modify its current capacity. The *position* tag for each device in the list will indicate whether that particular phase of the bank is open or closed. All other tags for devices in the *mspOvercurrentDeviceList* that are not mentioned here are ignored by Milsoft.

It is important to note that the overcurrent device elements themselves have no property to indicate whether they are a fuse, recloser, circuit breaker, etc… The device type in Milsoft is determined based on the equipment associated with the element. For example, if the first existing phase of an overcurrent device contains equipment named "FUSE 20A" which corresponds to a piece of fuse equipment in Milsoft's equipment database, then Milsoft will draw the overcurrent device as a fuse.

Overcurrent devices do not have partner elements in Milsoft; switches are the only elements with partners. So, the *partner* tag for the *overcurrentDeviceBank* is ignored by Milsoft.

As mentioned earlier, overcurrent devices now serve as feeder bays in Milsoft. To specify that an overcurrent device should serve as a feeder bay, use the *feederList* tag under the *substation* element. (See the "Source Elements" section of this document.)

For convenience, Milsoft also recognizes the following extension tags for the *overcurrentDeviceBank*. Remember that these tags must be children of the *extensions* tag in order to validate against the schema.

isFeeder	0 if the device is not a feeder bay, 1 if it is. Default is 0.
feederNumber	The feeder number associated with this feeder bay.
feederColor	An integer representing the feeder's color. 0x00BBGGRR

Switch Elements

Milsoft uses only the following fields specific to *switchDeviceBank*:

partner	The *name* or *objectID* attributes of this tag contain a reference to the switch's partner switch. Partners are described below.
mspSwitchDeviceList	The *mspSwitchDeviceList* provides far more flexibility than Milsoft is capable of representing. The only data in *mspSwitchDeviceList* we can use is *position*, and because Milsoft does not support independent switch phases, it will just use the first *position* tag it encounters in the *mspSwitchDeviceList* to determine whether the entire switch is open or closed.

Milsoft currently does not allow the phases of a switch to be opened and closed independently, however we plan to add this functionality in the future. For now, *isGanged* is not being used, but the default will be "true" if we ever do decide to use it.

Milsoft models a switch as two elements with different names but which are considered "partners" of one another. This representation allows Milsoft to handle the tricky case of an open tie switch in which one switch is actually on two different feeders. There are basically three possible configurations of a switch, and it will be helpful to understand how Milsoft models each of them. In the following examples OH1 and OH2 are overhead lines, SW-A and SW-B are the two partners of a switch, and SRC1 and SRC2 are substations. Milsoft models the three different switch configurations as follows:

Configuration 1: A closed switch on a single feeder.

SRC1 ← OH1 ← SW-A ← SW-B ← OH2

Notice that the downline partner of the switch, SW-B, is parented to SW-A. Although SW-A and SW-B are partners of the same switch, SW-A is also the parent of SW-B because current is flowing through the closed switch (from SW-A to SW-B).

Configuration 2: An open switch on a single feeder.

SRC1 ← OH1 ← SW-A SW-B ← OH2

Notice that SW-B has no parent. The switch is open, so SW-B and all elements downline from it are not powered (or disconnected). Opening the switch actually changes the circuit's linkage by removing the parent link between the downline and upline partners of the switch.

Configuration 3: An open tie switch (spanning two different feeders).

SRC1 ← OH1 ← SW-A SW-B → OH2 → SRC2

Notice that although SW-A and SW-B are partners of the same switch, they have different parents. SW-A is parented to OH1 and is considered to be on feeder SRC1. SW-B is parented to OH2 and is on feeder SRC2. Suppose the user disconnects OH2 from SRC2 and wishes to feed OH2 from SRC1 instead. Using Milsoft, he would simply close the switch, and Milsoft would deduce that a backfeed operation should be performed.

To illustrate, disconnecting OH2 from SRC2 would look like:

SRC1 ← OH1 ← SW-A SW-B → OH2 SRC2

Then, after closing the tie switch (which performs a backfeed operation), we would see:

SRC1 ← OH1 ← SW-A ← SW-B ← OH2 SRC2

Notice that closing the tie switch changed the direction of current flow through OH2, so the parent-child linkage of the model had to be updated to reflect that.

While this representation of switches is somewhat complicated, Milsoft's import routine will do most of the work for you. For cases 1 and 2 above, it is perfectly acceptable to export a switch as a single element only. If Milsoft attempts to import a switch that has no partner tag (or an empty partner tag), it will create the switch partner automatically and correctly reparent any elements connected to the switch. This is the recommended way to export most switches using MultiSpeak.

However, if you need to export an open tie switch, the only way to do so is to export the switch as two partners. In order to do this, each partner of the switch will need a different name. The typical naming convention is to append a "-A" and "-B" to the name of the switch, however this is only a convention and not a naming requirement. Also, the *partner* tag of each switch must contain the name of the switch's partner. Note that it is not necessary for both partners of the switch to appear together in the XML file. In fact, the two partners do not even need to appear in the same XML file. It is common for GIS systems to generate a different MultiSpeak XML file for each feeder, so for open tie switches it is usually convenient to export each partner of the switch along with the feeder to which it belongs. Of course, each switch partner will need to know the name of its partner at the time it is exported. This can be tricky for a GIS system to do. One possible technique is to append the feeder name or feeder number to the name of each partner of the tie switch so that you can know in advance what the name of the partner will be.

If, while importing, Milsoft finds a switch with a *partner* tag, it will immediately create both the switch and its partner and give them the appropriate names even if the partner has not yet been encountered in the XML file. When that switch partner is finally encountered in the XML file (or in another XML file), Milsoft will detect that that element already exists in the model and will modify it with the appropriate data. Suppose the two partners of an open tie switch (SW-A and SW-B) exist in different XML files (Feeder1.xml and Feeder2.xml). While importing Feeder1.xml, Milsoft encounters a switch element named SW-A that has a *partner* tag with a *name* attribute of "SW-B". At this point, Milsoft knows nothing about SW-B other than the fact that it exists and it is the partner of SW-A. So, the model after importing Feeder1.xml might look something like:

SRC1 ← OH1 ← SW-A SW-B

Notice that SW-B has no parent at this time; it is just the unconnected partner of an open tie switch. This is a legal configuration for a switch in Milsoft, so even if Feeder2.xml was never imported there would be no problem with the model. If the model were to be opened right now in Milsoft, SW-B would be shown in the work environment under "Disconnected Elements".

Now, suppose Feeder2.xml is imported on top of the existing model and the switch element SW-B is encountered in the XML file. Milsoft will detect that SW-B already exists in the model, so it will not create a new switch. It will also detect that its partner, SW-A, already exists in the model, so it will not create it either. What it will do is apply all the new data for SW-B (such as its parent name) to the existing SW-B element. So, the SW-B element now knows its new parent is OH2. The model after importing Feeder2.xml would look something like:

SRC1 ← OH1 ← SW-A SW-B → OH2 → SRC2

These examples of paired switches have only involved open tie switches. Although it is possible to export other switch configurations as paired switches, it is not recommended. Milsoft takes many precautions to avoid illegal situations such as parenting an element between the two partners of a switch. To eliminate the possibility of such errors, it is strongly recommended

that all non-tie switches be exported as single switch elements, giving Milsoft the responsibility of adding the switch partner.

Regulator Elements

Milsoft uses only the following fields specific to *regulatorBank*:

regType An integer (0 or 1) that indicates whether the phases of the regulator bank work together (0) or function independently (1). A *regType* tag of 0 only makes sense for three phase regulators.

ctrlPhase If *regType* is 0, the *ctrlPhase* tag indicates which phase (A, B, or C) controls the regulator.

wdgType An integer (1-10) which represents the regulator's winding type where:
1 = tranForm_E_1
2 = tranForm_E_2
3 = tranForm_E_3
4 = tranForm_E_4
5 = tranForm_E_5
6 = tranForm_E_6
7 = tranForm_E_7
8 = tranForm_E_8
9 = tranForm_E_9
10 = tranForm_E_10
These winding definitions can be found in the Electrical Transmission and Distribution Reference Book in Table 7.

regulatorList Contains descriptions of the regulators on each phase of the bank.

It is important that the *phase* tag be populated for each regulator in the list. The *eqEquipment* tag for each regulator in the list should refer to the name of the appropriate regulator equipment in Milsoft's equipment database. Milsoft uses every child tag of *regulator* except for *kva*. If all the *eaEquipment* tags for each phase contain the same equipment name, Milsoft will represent this element in the circuit element editor as "1-phase units with all phases same". Otherwise, it will be represented in Milsoft as "1-phase units with different settings".

Transformer Elements

Milsoft uses only the following fields specific to *transformerBank*:

wdgCode	(The transformer winding codes are enumerated in the schema.)
transDescr	(The name of the transformer definition in the equipment database)
vInput	(Input voltage in kV. In L-L if primary is delta connected.)
vOut	(Output voltage in kV. In L-L if secondary is delta connected.)
sourceSideConfig	(Source side configuration for Y-D open, D-D open, D-Y open.)
tertVolts	(Tertiary voltage in kV.)
tertChild	(The *objectID* attribute should contain the name or GUID of the transformer's tertiary child.)
vOutNom	(Nominal output voltage in kV.)
vOutNomTertiary	(Nominal output voltage of tertiary in kV.)
transformerList	(Contains information about the transformer's equipment.)

Milsoft models a transformer bank as a single element which references up to three transformers in its equipment database.

The *transDescr* tag is deprecated and maintained only for backwards compatibility. Future implementations should use *transformerList* instead. If a transformer equipment name is passed in *transDecsr*, it will be used as the transformer equipment on all existing phases of the transformer bank. If the *transformerList* is also used, the data in the *transformerList* will be given precedence and will overwrite any data imported from the *transDescr* tag.

The *transformerList* contains a list of *transformer* tags which represent the individual transformers in the transformer bank. The following tags under *transformer* may be used; the other tags are ignored by Milsoft:

eaEquipment	This specifies the name of the transformer in Milsoft's equipment database that all of the following data refers to.
phase	If *phase* is ABC, the transformer is imported as a three-phase transformer. If *phase* is A,B,or C, the transformer is imported as a single-phase transformer and is applied to the appropriate phase on the circuit element.
kva	kVA of transformer.
impedance	Impedance of transformer.
nLLoss	No-load losses of transformer.

Consumer Elements

As of MultiSpeak 2, consumers are imported as *serviceLocation* elements.

Milsoft uses only the following fields specific to *serviceLocation*:

servStatus
0 indicates the consumer is inactive
1 indicates the consumer is active

servType
If servType contains an integer, it is mapped to the appropriate consumer type as follows:

0 = Residential

1 = Small Commercial

2 = Large Commercial

3 = Large Power

4 = Motor Load

5 = Irrigation

6 = Oil and Gas

7 = Traffic Lights

8 = Security and Street Lights

9 = Flat Rate Load

10 = Primary

The *servType* tag may contain a string instead. If so it must be one of the strings listed above. If *servType* contains an invalid consumer type, the default consumer type will be used. This is chosen by the user in the Circuit Element Preferences dialog in WindMil.

Source Elements

Milsoft uses only the following fields specific to *substation*:

zMin	The name of a ZSM impedance in the equipment database.
zMax	The name of a ZSM impedance in the equipment database.
busVolts	Bus voltage (in per unit, not volts. For example, 1.05 not 126).
ohGndZ	Overhead ground ohms
ugGndZ	Underground ground ohms
ldCon	Connection (W=wye connected, D=delta connected)
nomVolts	Nominal voltage (kV) (in L-G if wye connected, in L-L if delta connected)
ldAlloc	"true" if this source is a load allocation control point.
isRegulated	"true" if this source is regulated.

Because sources have no parent element, the parentSectionID should be set to ROOT or some other name that does not exist in the model.

The *feederList* is used to specify which overcurrent devices should serve as feeder bays for this source. There should be one *feederObject* in the *feederList* for each feeder bay on the substation. Using the *feederList* does not actually cause "feeder bay elements" to be added to the Milsoft model. The overcurrent devices to serve as feeder bays must be imported separately (using the *overcurrentDeviceBank* tag). This list simply designates those overcurrent devices as feeder bays and adds some feeder specific information to them.

For each *feederObject* it is important to specify the name and/or objectID of the overcurrent device using the *name* and *objectID* attributes of the *eaLoc* tag. The *feederName* tag is used to specify the feeder bay name to be associated with this device. This name will appear in Milsoft as the "alias" of the device, and it will also appear in Milsoft's work environment and reports. You may also specify a feeder number using the *feederNo* tag. This will appear in Milsoft in the "Feeder Number" field for the overcurrent device.

Generator Elements

Milsoft uses only the following fields specific to *generator*:

load	Defines the total generator kVA used by the negative load model.
ssDesc	The fault impedance at steady state. (Name of ZSM equipment.)
tranDesc	The fault impedance at transient. (Name of ZSM equipment.)
stDesc	The fault impedance at sub-transient. (Name of ZSM equipment.)
connected	"D" = Delta connected, "W" = Wye connected.
model	0 = Negative Load generator model 1 = unused

2 = unused
3 = Swing kVAR generator model

The following fields are used only by the swing kVAR generator model:

holdVoltsZ	The voltage the generator is to hold (In per unit)
kwOut	Generator output (in kW)
kvarLead	Maximum leading output (in kVAR)
kvarLag	Maximum lagging output (in kVAR)

Motor Elements

Milsoft uses only the following fields specific to *motor*:

load	Used only to obtain the motor load mix.
ssDesc	The fault impedance at steady state. (Name of ZSM equipment.)
tranDesc	The fault impedance at transient. (Name of ZSM equipment.)
stDesc	The fault impedance at sub-transient. (Name of ZSM equipment.)
status	0 = Disconnected 1 = Off 2 = Locked rotor 3 = Soft start 4 = Running
hp	Rated horsepower
pf	Power factor at full load (in per unit)
eff	Efficiency at full load (in per unit)
lgVolts	L-G voltage (in kV)
dropout	Motor drops out if % voltage is below this fraction.
nemaTyp	Motor NEMA code 0 = None 1 = A 10 = K 2 = B 11 = L 3 = C 12 = M 4 = D 13 = N 5 = E 14 = P 6 = F 15 = R 7 = G 16 = S 8 = H 17 = T 9 = J 18 = U
limit	Motor will not start if % voltage is below this fraction.
sftStTyp	0 = Starting across line 1 = Impedance soft start 2 = Auto transformer start

3 = Capacitor assisted start
4 = Partial winding start
5 = Delta Wye soft start

sftStR	For soft start, starting line resistance (real part of impedance)
sftStX	For soft start, starting line reactance (imaginary part of impedance)
sftStTap	For auto transformer start, at start the motor terminal voltage will be this fraction of the normal across line starting voltage. (In per unit)
lrPf	Locked rotor power factor (In per unit)
lrMult	Locked rotor kVA per horsepower.

MultiSpeak Extension Tags

In addition to the tags defined by MultiSpeak, Milsoft can also import and export some additional data. The following tags, if they are used, must appear in the XML file as child tags of the *extensions* tag. Furthermore, MultiSpeak requires the *extensions* tag to be the first child tag of an element.

symbolCoord	This is used like the *coord* tag in *mapLocation*; it contains an *X* and a *Y* tag which represent the location of the element's symbol as it appears in Milsoft. It is not necessary to export this from a GIS, but if you are importing from Milsoft, it might be useful.
note	The text of a note associated with this element.

Certified by:

For Milsoft Utility Solutions, Inc.:

Name: Luis Malavé

Executive Vice President
Title:

Date: 2/19/2013

For GeoDigital:

Name: Sean Solberg

Vice President/CTO
Title:

Date: 2/19/2013

Assertions Verified by:

Name: Gary McNaughton

MultiSpeak Technical Coordinator
Title:

NRECA/Cornice Engineering
Testing Agent

Date: 2/19/2013

Disclaimer:

The assertions made in this document are statements of the vendors offering the two products listed above. The Testing Agent has observed the software performing the tasks described in these vendor assertions.

Neither NRECA, Cornice Engineering, Inc. (MultiSpeak Project Coordinator) acting on behalf of NRECA, makes any warranty nor guarantee that the software will perform as described in this assertion when installed at any specific utility. Furthermore, neither NRECA, Cornice Engineering, Inc. makes any warranty nor guarantee that the software described will be suitable for any specific purpose or need.

As used herein, the word "*verify*" shall mean an expression of the Testing Agent's professional opinion to the best of its information, knowledge and belief, and does not constitute a warranty or guarantee by NRECA or the Testing Agent.