Making .Net Add-ins for Parcel Fabrics

Introduction

This document is for third-party developers that want to use the parcel fabric libraries shipped with the ArcObjects .Net SDK product. The names of the libraries are:

GeoDatabaseExtensions Library
Cadastral Library
CadastralUI Library
GeoSurvey Library

Through the use of ArcObjects and .Net, it is possible to programmatically edit any of the attributes or the geometry for any rows of the parcel fabric tables. However, it is important to understand the model and concepts before doing this development work, because very often a programmatic change to a row in one table will require programmatic updates to, or inserts of, rows to other fabric tables. This is required in order to maintain a valid parcel fabric dataset, to avoid data corruption of parcels, and to ensure that all the parcel editing tools will continue to function as expected after the add-in code has run.

Many non-editing customizations can be done via use of python scripting and the ArcPy library. See the Resources section for more information. Prior to starting development of customized add-ins using .Net for purposes that do not require parcel editing, consider using Python, as it is a simpler environment by comparison.

There is a large amount of information regarding parcel fabric data model, concepts and editing that are available via:

- product documentation,
- recorded land record meetups,
- technical workshop sessions at the Esri User Conferences,
- training resources provided by Esri Training Instructor services.

This document builds on the knowledge presented at those places, and adds additional information required from the development perspective.

Parcel Fabric Data Model & Concepts

The on-line help covers the fundamental concepts and data model of the parcel fabric in the section called:

The Elements of a Parcel.

Before developing add-ins for parcel fabrics this information should be well understood, and is a pre-requisite to the content in this document.

Programmatically Editing Parcel Fabric Attributes

Code to edit the attributes of fabric tables must be written with the following three field classifications in mind:

1. *Geometry & System fields* – examples: Shape_Length, MiscloseDistance, SystemStartDate. These are fields that the user does not directly edit; the parcel editing system updates these based on changes made to other attributes.

2. *Private fields* – examples: Bearing, Distance, Radius. These are fields that are changed in the parcel editing controlled environment, such as in the Lines grid after opening a parcel. These cannot be changed directly in the table window, nor in the editing attributes dialog.
3. *Public fields* – examples: Name, CustomField1. These fields are those that are completely open for editing in the table window, in the editor's attributes dialog, and also in the property pages presented when you click Open Parcel.

The Parcel Locking Model: Overview

The parcel fabric has a multi-user model that uses exclusive edit locks on database records. Locks are not required for all parcel edits. Whether or not a lock is required depends on the classification of the field to be edited, as defined above. The next section on Parcel Locking and Jobs provides more details about how the Jobs and Parcel Locks are used. Parcel locks are reserved only for SDE, and are not required for file geodatabases. Jobs can be used for different purposes, and in general, are optional for file geodatabase, but are required in an SDE multi-user environment for the role that they play in acquiring parcel locks.

Parcel Locking and Jobs

Why are Parcel Locks Needed?

Since each single parcel in a parcel fabric is represented by multiple records across four different tables, in multi-user editing environments there is parcel-fabric-specific logic implemented for doing reconciles of child versions with the default version.

In the context of fabric reconciles, the three classifications of attribute fields defined in the previous section are relevant.

The fabric reconcile logic relies on acquiring an *edit* lock only for parcel edits that update private attributes. Edit locks are not required for public attributes, or for geometry and system attributes. The following conditions are implemented in the system:

1. There will never be any reconcile conflicts of parcel *geometry and system* attributes, based on the principle of *last-one-in-wins*. (see note below: More Information: Last One in Wins.)
2. A reconcile conflict will never be presented on parcels' *private* attributes.
3. Parcel's *public* attributes are treated the same way as standard features, and so any conflicts in parcels and lines records will be presented and let through by the parcel fabric reconcile process.

This implementation includes a limitation that edits are only permitted in the child versions of default. A version that is a child of a child (grandchild) version will be presented in the user interface as read-only. Attempting to make edits on the grandchild version will result in a message indicating that the edits are not permitted. There is however an ability to create a Surrogate default version. This is an optional configuration that is useful to some customers, and is implemented by applying a setting on the fabric property. This is described in the section Surrogate Default.

You only need to acquire parcel edit locks if you are changing a *private* attribute on a parcel or a *private* attribute on a line that belongs to that parcel.

Once an Edit Lock is successfully acquired, it guarantees the edits to the parcel and the parcel's lines will be persisted in the database during a reconcile-and-post.

Examples of edits that require a parcel lock include: deleting a parcel, updating the bearing or distance attribute on a parcel's line, or unjoining a parcel.

The Parcels table fields that require a lock are:

```
Joined
Compiled
Historical
SystemEndDate
Group
Unclosed
Construction
BacksightBearing
```

The Lines table fields that require a lock are:

```
FromPointID
ToPointID
Bearing
Distance
Type
Category
Radius
ArcLength
Delta
CenterPointID
Historical
RadialBearing
TangentBearing
LineParameters
InternalAngle
SystemEndDate
```

The following code has two functions, showing how to create a Job and get edit locks. The next section provides further background information about the Cadastral Job system tables that are involved in this process.

```
bool IsFileBasedGDB = (ArcMap.Editor.EditWorkspace.WorkspaceFactory.WorkspaceType !=
  esriWorkspaceType.esriRemoteDatabaseWorkspace);

if (!IsFileBasedGDB)
{
  //for file geodatabase creating a job is optional
  //see if parcel locks can be obtained on the selected parcels. First create a job.
  string NewJobName = "";
  if (!CreateJob(pCadEd.CadastralFabric,"Sample Code change line to curve", out NewJobName))
    return;

  if (!TestForEditLocks(pCadEd.CadastralFabric, NewJobName, pParcelsToLock))
    return;
}

private bool CreateJob(ICadastralFabric Fabric, string JobDescription, out string NewJobName)
{
  DateTime localNow = DateTime.Now;
  string sTime = Convert.ToString(localNow);
  ICadastralJob pJob = new CadastralJob();
  pJob.Name = NewJobName = sTime;
  pJob.Owner = System.Windows.Forms.SystemInformation.UserName;
  pJob.Description = JobDescription;
  try
  {
    Int32 jobId = Fabric.CreateJob(pJob);
    return true;
  }
  catch (COMException ex)
```

```
  {
    if (ex.ErrorCode == (int)fdoError.FDO_E_CADASTRAL_FABRIC_JOB_ALREADY_EXISTS)
    {
      MessageBox.Show("Job named: '" + pJob.Name + "', already exists");
    }
    else
    {
      MessageBox.Show(ex.Message);
    }
    return false;
  }
 }


private bool TestForEditLocks(ICadastralFabric Fabric, string NewJobName, ILongArray ParcelsToLock)
{
  ICadastralFabricLocks pFabLocks = (ICadastralFabricLocks)Fabric;
  pFabLocks.LockingJob = NewJobName;

  ILongArray pLocksInConflict = null;
  ILongArray pSoftLcksInConflict = null;
  try
  {
    pFabLocks.AcquireLocks(ParcelsToLock, true, ref pLocksInConflict, ref pSoftLcksInConflict);
    return true;
  }
  catch (COMException pCOMEx)
  {
    if (pCOMEx.ErrorCode == (int)fdoError.FDO_E_CADASTRAL_FABRIC_JOB_LOCK_ALREADY_EXISTS ||
      pCOMEx.ErrorCode == (int)fdoError.FDO_E_CADASTRAL_FABRIC_JOB_CURRENTLY_EDITED)
    {
      MessageBox.Show("Edit Locks could not be acquired on all parcels of selected lines.");
      // since the operation is being aborted, release any locks that were acquired
      pFabLocks.UndoLastAcquiredLocks();
    }
    else
      MessageBox.Show(pCOMEx.Message + Environment.NewLine + Convert.ToString(pCOMEx.ErrorCode));
     return false;
  }
}
```

In the above code you can see a reference to “Soft Locks.” A soft-lock is transferable if editing within the same version. This is described in more detail in the Appendix section called Technical Implementation for the Cadastral Jobs Role in Parcel Locks, but it is a rare scenario where these types of locks come into play if the recommended workflows for parcel fabric editing are followed.

Multi-user edits: Best Practices

It is recommended that a single user edits within the child version of default and that whenever possible the version edits do not overlap spatially. Also, the envelope of the edited parcels should not cover a very large spatial data extent; rather the edits should be contained within smaller areas, and to edit a different feature that is spatially further away, a different version used instead. This is because the reconcile logic uses the envelope of the all edited features to compare features across versions.

It is also recommended that in a high volume production environment, that:

- versions should be work-order based and posted back to default as soon as possible; versions that remain un-posted for a long time, spanning more than a week, for example, should be avoided when possible.

- a database compress should be scheduled to run on a nightly basis.

Cadastral Jobs' Role in Parcel Locks

Jobs contain the list of features, (parcels and control points) that describe a group of objects that will be edited. In addition, the job also flags which parcels have a lock. Control points are not edit locked, they are added to the job automatically, by virtue of being associated with one of the parcels in the same job.

The primary role of a cadastral fabric job in an enterprise environment is to identify the version that holds onto the parcel locks. Parcel locks are not defined by an attribute on the parcel table; rather they are managed by two non-versioned tables that reference the records in the parcels table. For a conceptual overview, please see the Overview section of the documentation for the *GeoDatabaseExtensions Library*.

Using or Creating a Cadastral Job for a Purpose other than Locking

Jobs can also be used for easily collecting information for other purposes, such as demonstrated in the code below, showing a job being created to export a Cadastral XML file representation for a subset of the parcels in the fabric. When jobs are used in this way there is no distinction made on whether the fabric is in an enterprise database or not, as it does not have an impact on edit-locking. In the code below, the subset is based on the collection of parcels held by a selected Plan.

```
protected override void OnClick()
{
  try
  {
    //getting the fabric from the extension. Requires an edit session.
    IEditor pEd = (IEditor)ArcMap.Application.FindExtensionByName("esri object editor");
    //if we're not in an edit session then bail
    if (pEd.EditState != esriEditState.esriStateEditing)
      return;
    //get the cadastral editor extension
    UID pUID = new UIDClass();
    pUID.Value = "esriCadastralUI.CadastralEditorExtension";
    ICadastralEditor pCadEd = (ICadastralEditor)ArcMap.Application.FindExtensionByCLSID(pUID);
    //Return the currently set target Fabric
    ICadastralFabric pFabric = pCadEd.CadastralFabric;
    //QI to get the cadastral selection
    ICadastralSelection pCadaSel = (ICadastralSelection)pCadEd;
    //Get the selected plans
    IEnumGSPlans pEnumGSPlans = pCadaSel.SelectedPlans;
    pEnumGSPlans.Reset();
    IGSPlan pGSPlan = pEnumGSPlans.Next();
    string s_IDs = "";
    int iCnt = 0;

    //Build a list of plan ids for the in clause query
    while (pGSPlan != null)
    {
      if (iCnt == 0)
        s_IDs = pGSPlan.DatabaseId.ToString();
      else
        s_IDs += "," + pGSPlan.DatabaseId.ToString();
      iCnt++;
      pGSPlan = pEnumGSPlans.Next();
    }

    //get the parcel fabric table
    ITable pTable = pFabric.get_CadastralTable(esriCadastralFabricTable.esriCFTParcels);
    //assign the in clause to a new query filter on the parcels table to get the parcels contained
```

```
    // by the selected plans
    IQueryFilter pQuFilter = new QueryFilterClass();
    pQuFilter.WhereClause = "planid in (" + s_IDs + ")";
    //get the search cursor and create an FIDSet for the parcels contained by the selected plans
    ICursor pCur = pTable.Search(pQuFilter, false);
    IFIDSet pFIDSetParcels = new FIDSetClass();
    IRow pParcel = pCur.NextRow();
    while (pParcel != null)
    {
      bool bExists = false;
      pFIDSetParcels.Find(pParcel.OID, out bExists);
      if (!bExists)
        pFIDSetParcels.Add(pParcel.OID);
       Marshal.ReleaseComObject(pParcel); //garbage collection
      pParcel = pCur.NextRow();
    }
    Marshal.ReleaseComObject(pCur); //garbage collection
    //Create a job.
    ICadastralJob pJob = new CadastralJobClass();
    Guid g;
    g = Guid.NewGuid();
    DateTime localNow = DateTime.Now;
    string sTime = Convert.ToString(localNow);
    //ensure the job name is unique by including a guid in the name
    pJob.Name = sTime + ":" + g.ToString();
    pJob.Owner = SystemInformation.UserName;
    pJob.Description = "Save Plan as XML";
    pJob.AdjustmentAreaParcels = pFIDSetParcels;
    int jobId = pFabric.CreateJob(pJob);
    IGeoDataset pGeoDS = (IGeoDataset)pFabric;
    IProjectedCoordinateSystem pPCS = (IProjectedCoordinateSystem)pGeoDS.SpatialReference;
    //extract the parcels into a cadastral xml stream, and save to the Cadastral XML to a file
    IXMLStream xml = pFabric.ExtractCadastralPacket(pJob.Name, pPCS, null);
    SaveFileDialog saveFileDialog = new SaveFileDialog();
    // Set File Filter
    saveFileDialog.Filter = "Cadastral XML (*.xml)|*.xml|All Files|*.*";
    saveFileDialog.FilterIndex = 1;
    saveFileDialog.RestoreDirectory = true;
    // Warn on overwrite
    saveFileDialog.OverwritePrompt = true;
    // Don't need to Show Help
    saveFileDialog.ShowHelp = false;
    // Set Dialog Title
    saveFileDialog.Title = "Save Plan to Cadastral XML";
     // Display Open File Dialog
    if (saveFileDialog.ShowDialog() != DialogResult.OK)
    {
      saveFileDialog = null;
      return;
    }
     xml.SaveToFile(saveFileDialog.FileName);
  }
  catch (Exception ex)
  {
    MessageBox.Show(ex.Message);
    return;
  }
  finally
  { }
}
```

What happens if you edit a private field without getting an edit lock?

Without a parcel lock, the fabric reconcile logic would result in scenarios where an add/delete conflict would not be presented in the conflicts dialog when it should be or the opposite where conflicts are presented that should be. This could potentially result in orphan lines, or other types of data corruption. Orphan lines are rows in the parcel lines table that do not have a corresponding parcel record.

Fabric Reconcile Logic

Fabric reconcile has been designed, to avoid conflict resolution in the information that the fabric manages. Extending on the three field classifications described previously, the fabric reconcile logic handles this as follows:

1. Edit Locks are created on parcels that have had Private fields edited, so that no other version may edit a parcel that has been edit-locked. This prevents edit conflicts. Edits in the tables window are blocked, but the Parcel Editor Details pages allow the edits so that the parcel editor system can apply these edit locks. (When writing code that changes these fields, edit locks should be acquired programmatically, as demonstrated by the example code in the previous section.)
2. *Last-one-in-wins strategy*, for System & Geometry fields. This strategy is made evident in cases where two versions change the locations to the same fabric points, but in two different versions. The result is that the last version to reconcile has those changes applied, and no conflict dialog is presented.
3. When conflicts are presented they refer, by design, only to the Public fields as described above.

Based on this, when the user interface presents conflicts in the Reconcile dialog, the “by Object” or “by Attribute” option is not applied and is ignored for the fabric data, and would only apply to the non-fabric data that is in conflict.

The “Target Version” or “Edit Version” option is only applied to the “Public” category of fields, described above. It would also only apply to the non-fabric data that is in conflict.

More Information: Last-One-In-Wins

Consider multiple users, each editing in different versions, doing parcel fabric adjustments in the same area. During a reconcile the fabric does not present the conflicts from different coordinate/geometry results, as this would not make sense since each of these adjustments are based on processing the record attributes on *lines.* Choosing between geometry results without the corresponding line information breaks the concept. For this reason, when making programmatic changes to point geometry, no parcel edit locks are required, the latest adjustment to be posted to default is the result that is stored in the fabric points table. The reconcile process also runs the parcel fabric regenerator to ensure that the new geometry point changes are propagated to the rest of the lines and parcel polygon geometries, and also the line point geometries. The parcel regenerator and how it can be accessed and used through ArcObjects is presented in the section called Regenerating the Fabric.

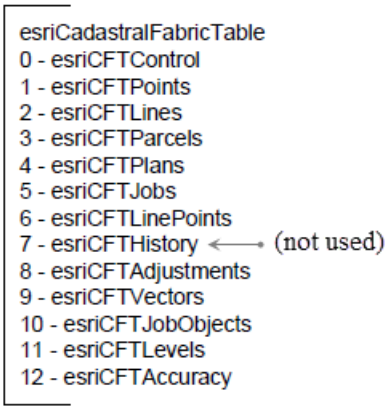
Opening the Fabric Tables for Edits

To edit the fabric tables you first need to get a reference to the target fabric. There are a number of methods to do this; examples of these are shown in the snippets of code available on ArcGIS Online (AGOL) item Parcel Editor Code Snippets, but the easiest approach is to get the fabric from the Cadastral editor extension, and then get the required table using the `esriCadastralFabricTable` enumeration, as shown in the following lines of code:

```
ICadastralEditor pCadEd =
(ICadastralEditor)ArcMap.Application.FindExtensionByName("esriCadastralUI.CadastralEditorExtension");

IFeatureClass pFabricPointsFC =
(IFeatureClass)pCadEd.CadastralFabric.get_CadastralTable(esriCadastralFabricTable.esriCFTPoints);
```

Note that in this case the fabric class table referenced is the fabric points table, and since it has a geometry, it can be declared as a feature class. The other members of the esriCadastralFabricTable enumeration are:



Once the feature class object or table has been accessed, the next step is to start an edit operation, turn off the read-only flag, and then create a database cursor on these tables. This is achieved using the ICadastralFabricSchemaEdit2 interface, and is demonstrated in the code that follows.

```
//1. start an edit operation
ArcMap.Editor.StartOperation();


//2. turn off the read-only flag on the cadastral fabric tables that are to be edited. Make sure that
// this step precedes the creation of the cursor.
ICadastralFabricSchemaEdit2 pSchemaEd = (ICadastralFabricSchemaEdit2)pCadEd.CadastralFabric;
pSchemaEd.ReleaseReadOnlyFields((ITable)pFabricPointsFC, esriCadastralFabricTable.esriCFTPoints);

//3. Create the Cursor
ICursor pFabricPointsFC
IFeatureBuffer featureBuffer = pPointFeatClass.CreateFeatureBuffer();

featureBuffer.set_Value(indxX, X);
//4. Do My Edits on the fabric tables using geodatabase ArcObjects
```

After changes to the table has been made, reset the read-only flag, and then stop the edit operation, as shown in the following code:

```
//After the edits have been made, reset the read-only fields as follows:
pSchemaEd.ResetReadOnlyFields(esriCadastralFabricTable.esriCFTPoints);
ArcMap.Editor.StopOperation("My Edits");
```

When the fields have been made available for editing, the same coding practices can be used as would be used for standard geodatabase feature classes and tables. You can use standard ArcObjects for the geodatabase as you would for standard feature classes/tables to insert/update table records.

Since the fabric edits often require accessing multiple tables at once, it can be tempting to use multiple nested cursors. This should be avoided since it makes management of the editable switch on fields complicated, and it also makes garbage collection difficult. Instead, use one cursor on a table in a loop at a time, and store the required information in a data structures such as lists, arrays, or dictionaries. Then use this stored information in subsequent loops that use another new cursor on a different fabric table.

Feature Adjustments and Vectors

As described in the Overview section of the SDK documentation for the *GeoDatabaseExtensions Library*, there is a system that is part of the parcel fabric that tracks and records the changes to fabric points. Whenever the position of a fabric point is changed, the coordinate x and y from and to values are recorded to a table and grouped with all the other fabric point changes that were part of the same edit. This grouping is called the adjustment area, and is also stored as an adjustment level. The information from these tables may be used to adjust the geometry of standard feature classes, to match the changing position of the fabric points. When updating the positions of points there are some recommended Parcel Editing ArcObjects to use, and also some additional information about the data model needed, as described in the following section about fabric points, coordinates, and associated control points.

Fabric Points, Coordinates, and Associated Control Points

When working with fabric points and control, and line points, a few other rules apply when making changes to these tables:

- Fabric point's x and y attribute values must match the shape geometry location of the fabric point.
- Control point x and y attribute values must match the shape geometry location of the control point.
- Line-point geometry must be at the same geometry location as the fabric point that it references.
- There are cases where a fabric point that is a curve center, can also be used as a regular boundary point. In these cases the regular fabric point must be set to trump the center point flag setting.
- a line that is a circular arc must set the CenterPointID flag to the id of the center point that its radial lines reference.
- vectors should not be generated for curve center point movements.

A New Interface available with 10.4 - ICadastralFabricUpdate

If a fabric point is going to have its geometry updated via code, it is important to recognize the other expected effects, such as the vector creation, and updating the geometry for the connected lines, associated line points, and parcel geometry.

To ensure that all the requisite work is done in the data, the recommended way to make this type of change is via the *ICadastralFabricUpdate* interface. This interface is public for the 10.4 (and higher) SDK release, and the code below shows how it is used.

```
ICadastralFabricUpdate pFabricPointUpdate = (ICadastralFabricUpdate)TheTargetFabric;
while (pPointFeat != null)
{

   :
   :
//make an updated point location for a particular fabric point that you have the objects ID for
// add the fabric point objects ids to the CadastralFabricUpdate object, and add the new
// x and y coordinates, and whether or not the point is a center point for a curve
  pFabricPointUpdate.AddAdjustedPoint(pPointFeat.OID, pPt.X, pPt.Y, bIsCenterPoint);
  Marshal.ReleaseComObject(pPointFeat);
  pPointFeat = pFeatCurs.NextFeature();
}

ArcMap.Editor.StartOperation();
pFabricPointUpdate.AdjustFabric(pTrkCan);
pFabricPointUpdate.ClearAdjustedPoints();
ArcMap.Editor.StopOperation("Move fabric points");
```

Control Point to Fabric point association

The following relationships between fabric points associated with the control points should be noted, to avoid data corruption:

1. The fabric point's [Name] field must have the control point's Name in it.
2. The control point's [POINTID] field must have the objected of the associated fabric point.

Note that the Parcel Editor's Regenerate command (see programmatic use in the next Section called Regenerating the Fabric) will change the *geometry* of the fabric points to match the X, Y attributes. This means that running the regenerate command after changing only the geometry of a fabric point will have the effect of "undoing" the geometry change.

Regenerating the Fabric

The parcel fabric has a lot of redundancy built into it. In general, the concept of updating and propagating changes involves giving highest weight to the attribute information, and then using the attributes to update missing or out-of-date geometry information. The parcel fabric regenerator is used for things such as updating or adding missing radial lines of a circular arc, or updating the lines and polygon geometry after point geometry changes have been made.

The code snippet below shows how it can be used on a subset of parcels in the parcel fabric. It includes the ability to only perform certain types of regeneration, using the bitwise operator. In the code below the first three member of the enumeration are used, and so the value passed in for the *RegeneratorBitmask* property is 7 (1+2+4).

```
ICadastralFabricRegeneration pRegenFabric = new CadastralFabricRegenerator();
#region regenerator enum
// enum esriCadastralRegeneratorSetting
// esriCadastralRegenRegenerateGeometries         =   1
// esriCadastralRegenRegenerateMissingRadials     =   2,
// esriCadastralRegenRegenerateMissingPoints      =   4,
// esriCadastralRegenRemoveOrphanPoints           =   8,
// esriCadastralRegenRemoveInvalidLinePoints      =   16,
// esriCadastralRegenSnapLinePoints               =   32,
// esriCadastralRegenRepairLineSequencing         =   64,
// esriCadastralRegenRepairPartConnectors         =   128

// By default, the bitmask member is 0 which will only regenerate geometries.
// (equivalent to passing in regeneratorBitmask = 1)
#endregion
pRegenFabric.CadastralFabric = TheCadastralEditorExtension.CadastralFabric;
pRegenFabric.RegeneratorBitmask = 7;
pRegenFabric.RegenerateParcels(pFIDSetForParcelRegen, false, pTrackCancel);

//15 (enum values of 8 means remove orphan points; this only works when doing entire fabric)
pStepProgressor.MinRange = 0;
pStepProgressor.MaxRange = iChangeCount;
pStepProgressor.StepValue = 1;
pProgressorDialog.Animation = ESRI.ArcGIS.Framework.esriProgressAnimationTypes.esriProgressSpiral;
        pRegenFabric.RegenerateParcels(pFIDSetForParcelRegen, false, pTrackCancel);
```

Working with Fabric Events

Events are useful for coding customized behavior when certain actions occur as a result of user activity. For example, when an editor enters information into the lines grid of the parcel details window, events are fired that return the information entered into the grid cell. This information can then be used for customized behavior. There are also events fired for actions that relate to database edits.

The events are initiated and wired into an Addin that as an editor extension. When the application starts up the OnStartup function is run, and is the place where the event listeners are created:

```
protected override void OnStartup()
{
  IEditor theEditor = ArcMap.Editor;
  //get the cadastral extension
  ICadastralExtensionManager2 pCadExtMan =
(ICadastralExtensionManager2)ArcMap.Application.FindExtensionByName("esriCadastralUI.CadastralEditorExten
sion");
  m_pParcEditorMan = (IParcelEditManager)pCadExtMan;
  m_pParcelEditEvents = pCadExtMan as IParcelEditEvents_Event;
  WireParcelFabricEvents();
}
```

The code for capturing the line grid cell events is shown in the function below:

```
void WireParcelFabricEvents()
{
  ParcelEditorEvents.OnGridCellEdit += delegate(int row, int col, object inValue)
  { return OnGridCellEdit(ref row, ref col, ref inValue); };
}
```

The code for the desired custom behavior for the line grid cell entry value is shown in the function below. In this case the goal is to allow the 0 key to be used like a Delete key. When the incoming value is detected as a “0” string, the event returns an empty string; this has the same effect as deleting the default value that appears in this field when it gets focus. The delete key is used often in this field, when the row is intended to be a circular arc, and so allowing use of the “0” key on the 10-key pad is more efficient from the user entry stand-point.

```
object OnGridCellEdit(ref int row, ref int col, ref object inValue)
{
  object OutValue = inValue;
  IGSLine pGSLine = null;
  IParcelConstruction pCourses = (IParcelConstruction)m_pParcEditorMan.ParcelConstruction;
  if (col == 3)//this is the distance field
  {//this code uses a value of 0 on the distance field to over-ride
    string sDistance = Convert.ToString(inValue);
    if (sDistance.Trim() == "0")
    {
      bool IsCompleteLine = (pCourses.GetLine(row, ref pGSLine));
      //true means it's a complete line, false means it's a partial line
      if (!IsCompleteLine)
      {
        pGSLine.Radius = pGSLine.Distance; //default the radius to be the same as the default distance
        return String.Empty;
      }
      //note that the type of the inValue must be honoured when returning the value from this function.
    }
  }
  return OutValue;
}
```

Events for database edits are implemented by event handlers using the IObjectClassEvents interface, as shown in the code below:

```
public void WireFabricTableEditEvents()
{
  if (_objectClassEventList == null)
    _objectClassEventList = new List<ESRI.ArcGIS.Geodatabase.IObjectClassEvents_Event>();

  //create event handler for each fabric class in the edit workspace
  try
  {
    _objectClassEventList.Clear();
    for (int i = 0; i < _fabricObjectClassIds.Count; i++)
    {
      IObjectClass pObjClass = (IObjectClass)_fabricObjectClasses.get_Element(i);
      //Create event handler.
      ESRI.ArcGIS.Geodatabase.IObjectClassEvents_Event ev =
(ESRI.ArcGIS.Geodatabase.IObjectClassEvents_Event)pObjClass;
      ev.OnChange += new ESRI.ArcGIS.Geodatabase.IObjectClassEvents_OnChangeEventHandler(FabricRowChange);
      ev.OnChange += new ESRI.ArcGIS.Geodatabase.IObjectClassEvents_OnChangeEventHandler(FabricGeometryRowChange);
      //ev.OnCreate += new ESRI.ArcGIS.Geodatabase.IObjectClassEvents_OnCreateEventHandler(FabricRowCreate);
      _objectClassEventList.Add(ev);
     }
   }
   catch (Exception ex)
   {
     MessageBox.Show(ex.Message + " in Wire Fabric Events");
   }
  }
```

Resources

The following resource links provide help content on SDK for fabrics/parcel editor (note some older browser versions may display a blank screen for some of these hyper-links; if you experience this problem, please try an alternate browser)

GitHub Commits

Parcel Fabric desktop add-ins code samples

Parcel Fabric Quality Control add-in source code

Fabric Point Move To Feature add-in source code

Delete Fabric Records add-in source code

Attribute Assistant add-in source code

ArcGIS Online code snippets

How to access the fabric dataset a few different ways (from layer, extension, etc.) http://bit.ly/GCtfDB

Versioning Considerations for Fabric feature adjustment

https://desktop.arcgis.com/en/desktop/latest/manage-data/editing-parcels/versioning-considerations-for-the-fabric-feature-adjustment.htm

Appendix

Technical Implementation for the Cadastral Job's Role in Parcel Locks

This section will be informative for purposes of interpreting the contents of the tables. However, it is not a pre-requisite for parcel editing customizations. Parcel locks involve two tables in the Cadastral Fabric.
These non-versioned tables are called the *Jobs* table and the *JobObjects* table; they have the following fields:

***Jobs** table*		**JobObjects** table	
ObjectID		*ObjectID*	
Name	Name of the job	*JobID*	The *ObjectID* from the **Jobs** table.
Description		*ObjectType*	**0** = parcel **3** = Control point
Locked	Set to **1** when created Set to **0** when committed (reference only – not used by the code to infer locks)	*JobFeatureID*	ObjectID of the parcel or Control point being locked.
CreateDate	Date of job creation.	*Role (aka lock method)*	**-1** for edit parcels, **jobID** otherwise.
ModifiedDate	Date of most recent job modification.	*Edit*	**1** for edit parcels **0** for adjustment parcels (currently redundant)
CommitDate	Date of job commitment.		
Owner	User owning the job.		
Status	**0** = active **1** = committed		
SystemState	Unused		
Version	The name of the database version in which the job was created.		
LockMachine	Name of the machine.		
LockPID	Process ID containing the job.		

When a job is created, a new row is placed in the Jobs table. The *Locked* field is set to **1**, the *Status* field is set to **0**. If the database is versioned, the name of the current version is placed in the *Version* field. The *Owner* is set to the user creating the job. The *CreateDate* and *ModifiedDate* are set to the current date.

A particular job is locked if it has been opened for edits within the current edit session. This is indicated by entries in the *LockMachine* and *LockPID* fields for that particular job row. The job lock is released when the edit session is ended. (StopEditing). The *LockMachine* field is emptied and the *LockPID* field is set to **0**.

Each parcel within the job is added in the JobObjects table along with the *jobID* of the job containing it. The *Role (LockMethod)* is set to **-1** if the parcel is locked (edit parcel) and the jobID if the parcel is unlocked (adjustment parcel).

A parcel can be unlocked in more than one job (i.e. there can be multiple unlocked entries for the same parcel) but can only be locked by one job. The same parcel will not appear as both locked and unlocked in the same job (there will not be two rows for the same parcel with the same job ID, one locked and one unlocked)

If a parcel is not locked by any job, it can be upgraded to locked within a job holding it. This entails changing the *Role* from jobID to **-1** for the job currently being edited.

A lock can be transferred from one job to another as long as:

1. The jobs are in the same version.
2. The job containing the locked parcel is not itself locked.

These are referred to as *Soft Locks*.

In this case the locked parcel has its jobID changed to the new job and a new unlocked entry is added for the parcel for the old job.

To illustrate this, imagine two users working in the same version with jobs which contain the same parcels. To begin with, both user A and user B have a particular parcel as an unlocked parcel. User A edits that parcel and so obtains a lock on it. If user B tries to edit the same parcel, they will be blocked because it is already locked by A. If user A then saves the edits, user B will be able to obtain the lock and edit the parcel.

A lock cannot be transferred between versions

If a job is in the default version, the job can be *committed*, which deletes all entries for the job objects for that job (hence releasing the locks on all parcels in the job) and sets the *status* field to **1** for the corresponding row in the Jobs table.

If a job is not in the default version, the job will be committed (along with all other jobs in that version) when the version is posted back to default.

If a parcel appears in any job as locked, it can only be unlocked by committing the job or posting the version.

The parcel editing environment will keep the job lock information in sync with the undo/redo edit operation stack. Within the same edit session an undo will rewind the lock that was created. However, the physical lock will only be released once the edit session is aborted or ended. This is done to support the redo.