Connecting the Systems that Power Education

How to Use the Mappings Class

SIFWorks® ADK for Java
Developer Note #104

Edustructures LLC
891 W Baxter Drive
South Jordan, UT 84095
1.877.790.1261

www.edustructures.com

Introduction

The SIFWorks® Agent Developer Kit (ADK) for Java includes a powerful data mapping facility to convert records from an application's database to SIF Data Objects, and vice versa. This document demonstrates how to use the `Mappings` class from the `com.edustructures.sifworks.tools.Mappings` package.

An Overview

The ultimate goal of any SIF Agent is to exchange data between the application's local data repositories and the world of SIF. When publishing data, an agent prepares one or more `SIFDataObject` instances to describe the changed or requested data. This involves reading data from a local database and transforming, or "mapping", it into a `SIFDataObject` that can be sent to the zone by the ADK. Conversely, when an agent receives a SIF Event or SIF Response message, it must convert the `SIFDataObject` payload back into a form that can be stored in the application database.

The ADK's Mappings class provides a simple, configurable method of transforming any set of field/value pairs into a `SIFDataObject` instance and vice-versa. The class's public interface is simple and straightforward:

To transform a Java HashMap of field/value pairs into a SIFDataObject:

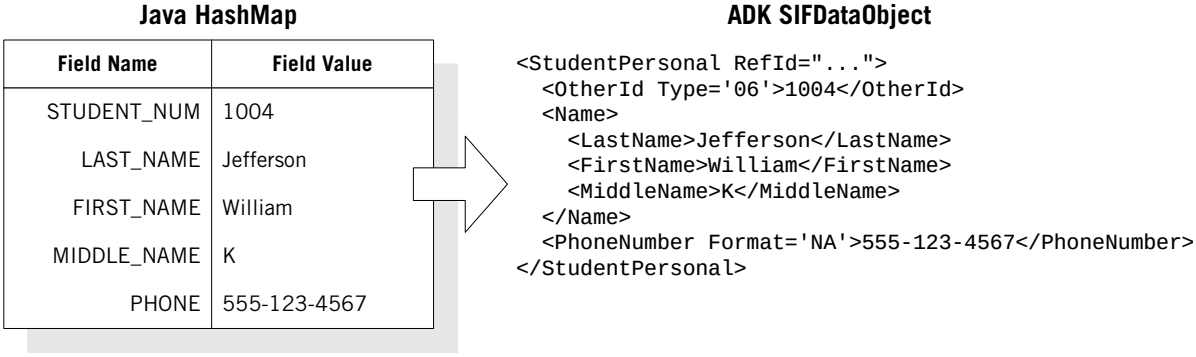
```
public void map( Map values, SIFDataObject object );
```

To transform a SIFDataObject back into a HashMap of field/value pairs:

```
public void map( SIFDataObject object, Map values );
```

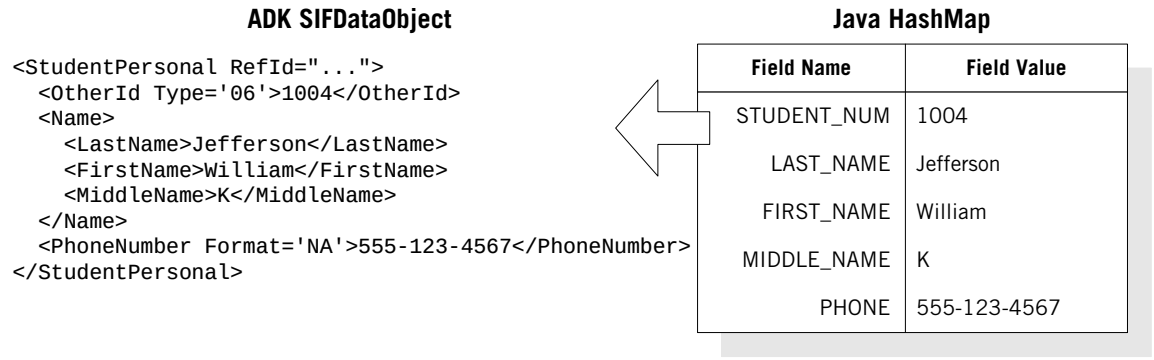
For example, suppose you have populated a `HashMap` with the fields of a student record from your database, and want to transform it into a StudentPersonal object. The first form of the `map` method accomplishes this task in one step:

```
public void map( Map values, SIFDataObject object )
```



Moving the other direction—that is, populating a `HashMap` from a `SIFData-Object`—is achieved by calling the map method with the parameters reversed:

```
public void map( SIFDataObject object, Map data )
```



The Mappings engine knows how to perform its transformations by consulting a set of object and field "mapping rules", usually specified in an XML configuration file. Here's the set of mapping rules that produced the StudentPersonal object above:

```
<object name="StudentPersonal">
      <field name="STUDENT_NUM">OtherId[@Type='06']</field>
      <field name="LAST_NAME">Name/LastName</field>
      <field name="FIRST_NAME">Name/FirstName</field>
      <field name="MIDDLE_NAME">Name/MiddleName</field>
      <field name="PHONE">PhoneNumber[@Format='NA']</field>
</object>
```

Mapping rules are written in a XPath-like style, where each `<field>` describes the path to a SIF element or attribute relative to a parent object. For instance, the rule for the `STUDENT_NUM` field is "`OtherId[@Type='06']`". This rule translates as follows: "Assign the value of the `STUDENT_NUM` field to a StudentPersonal element named OtherId and having a Type attribute with a value of '06'".

```
<OtherId Type='06'>1004</OtherId>
```

Similarly, the rule for the `LAST_NAME` field instructs the Mappings engine to create a Name element of StudentPersonal with a LastName child element:

```
<Name>
      <LastName>Jefferson</LastName>
</Name>
```

The field names that are used for each entry, such as `STUDENT_NUM` and `LAST_NAME`, are chosen by you; they have no significance to the ADK. However, the names must be unique within a group because they're used to identify values in the `HashMap` when performing a transformation.

Advanced Features

There are several other features of the Mappings class worth pointing out.

- **Mappings Hierarchy**

  Mappings rules can be grouped and organized in a hierarchy. At runtime, you can select the appropriate set of rules from the hierarchy based on the zone you're working with, the version of SIF the data object conforms to, and/or the SourceId of the agent that published the object. Furthermore, groups of mapping rules can be nested and the Mappings engine will inherit rules from parent groups. This makes it easy to "customize" the rules for individual zones or different versions of SIF.

- **Configuration File**

  Mappings rules can be read from an XML configuration file. This makes it easy to customize the behavior of your agent in the field. Customization is important for ensuring interoperability among agents that may not always ship with compatible default settings.

- **Call Java Methods at Runtime**

  A special convention to the rule syntax lets you embed calls to any static Java method in a field mapping rule. For example, you could write a Java method to pad a value with zeros, and then reference that method using the @ character to post-process the value of the `STUDENT_NUM` field. The value returned by the Java method is pasted into the SIF Data Object:

  ```
  <field name="STUDENT_NUM">
  ```

```
      OtherId[@Type='06']=@pad( $(STUDENT_NUM), 0, 8 )
</field>
```

Mappings Configuration

The two forms of the **map** method that we've seen so far handle the work of transforming data from a Java `HashMap` to a `SIFDataObject` and vice-versa. These functions are so straightforward that they don't require much more explanation. The complexity of the Mappings class lies in proper configuration of the mapping rules introduced in the previous section.

The first concept to understand when it comes to Mappings configuration is that groups of mapping rules are organized in a *hierarchy*. Each node in the hierarchy is represented by an instance of the Mappings class. The root of the hierarchy is for organizational purposes only; it cannot contain any rules and can't be used to perform transformations.

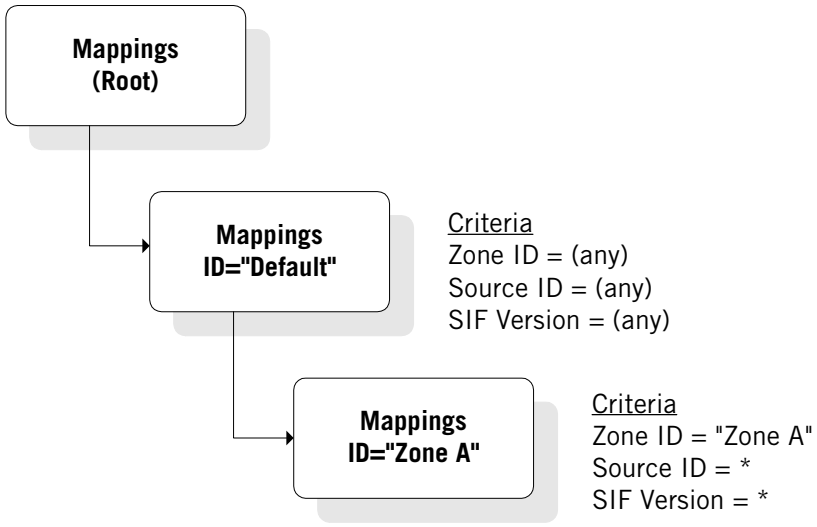
At runtime, you ask the Mappings class to "select" the most appropriate child object based on the criteria you provide it. This is done by calling the `select` method:

```
public Mappings select(
      String zoneId,
      String sourceId,
      SIFVersion version );
```

The Mappings class consults its hierarchy and selects from among its children the best available fit for the Zone ID, Source ID, and SIF Version criteria supplied. These three values generally come from the SIF message that's being processed as part of the transformation. If you're preparing an unsolicited SIF Data Object to send to a zone (e.g. reporting a SIF Event), you can pass the *SourceId* parameter a null value because it is not applicable in this case.

Mappings Hierarchy

To understand how Mappings are arranged in a hierarchy, let's look at an example. The following Mappings hierarchy defines two groups of rules: one that is applicable to all Zone IDs, Source IDs, and versions of SIF; and one that contains rules specifically for the "Zone A" zone.



The XML configuration file that defines this hierarchy might look something like this:

```
<mappings id=”Default”>

   <object name=”StudentPersonal”>
      <field name=”STUDENT_NUM”>OtherId[@Type=’06’]</field>
      <field name=”LAST_NAME”>Name/LastName</field>
      <field name=”FIRST_NAME”>Name/FirstName</field>
      <field name=”MIDDLE_NAME”>Name/MiddleName</field>
      <field name=”PHONE”>PhoneNumber[@Format=’NA’]</field>
   </object>

   <object name=”StaffPersonal”>
      ...
   </object>

   <mappings id=”Zone A” zoneIds=”Zone A”>

      <object name=”StudentPersonal”>
         <field name=”STUDENT_NUM”>OtherId[@Type=’08’]</field>
         <field name=”SSN”>OtherId[@Type=’SY’]</field>
      </object>

   </mappings>

</mappings>
```

Note the following about this configuration file:

- Each `<mappings>` node has a unique ID (“Default” and “Zone A”)
- The diagram depicts a hierarchy comprised of three Mappings nodes, but in the configuration file there are only two `<mappings>` elements. This is because the

root of the hierarchy is implicitly created by the Mappings engine. It has no corresponding element in the configuration file.

- The "Default" Mappings node defines a default set of rules that are not constrained by Zone ID, Source ID, or SIF Version. The "Zone A" Mappings node specifies the *zoneIds* attribute to restrict its rules to that zone (you can specify a comma-delimited list of zone IDs in this attribute.) This means that the Mappings engine will only choose this group of rules at runtime if you were to pass a value of "Zone A" to the first parameter of the `select` method.

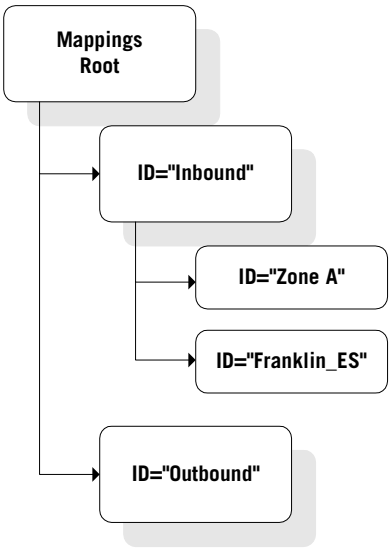
- Because the Zone A Mappings node is a *child* of the Default node, it will inherit all rules from that parent. In this example, we've overridden the rule for the `STUDENT_NUM` field to use an OtherId Type of 08 instead of 06, and we've added the SSN field to StudentPersonal objects in this zone. The remainder of the fields—`LAST_NAME`, `FIRST_NAME`, `MIDDLE_NAME`, and `PHONE`—will be inherited from the parent node.

Top-Level Nodes

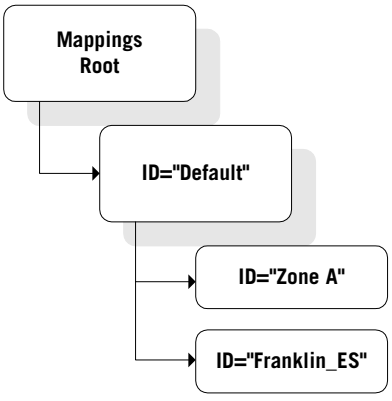
The Mappings hierarchy is intended to have one or more "top-level" nodes with well-defined names like "Default", "Inbound", "Outbound", etc. In this document we define a top-level node as an immediate child of the root Mappings object. Thus, a Mappings hierarchy must *always* be at least two levels deep: the root node plus one or more top-level nodes. Children of top-level nodes are optional.

You can choose any organization that best meets the requirements of your agent. Two are commonly used:

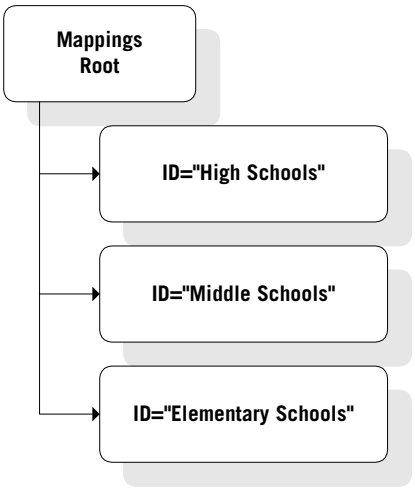
1. If you want your agent to have a unique set of rules for inbound messages versus outbound messages, create two top-level nodes: one named "Inbound" and the other named "Outbound". You would use the Inbound rules when processing incoming SIF Event and SIF Response messages, and the Outbound rules when reporting events and publishing data in response to SIF Requests.



2. While the above approach is the most flexible, it has one significant drawback: you must duplicate your set of Mapping rules in the configuration file. This can become a headache, particularly when developing GUI editors for editing mapping rules. A simpler technique is to define a single top-level Mappings node named "Default". The rules defined in this group will be used for SIF Data Object transformations on both inbound and outbound messages.



3. Yet another possibility is to create top-level nodes for "types" of zones. Here we're created a unique set of mapping rules for High School zones versus Elementary School zones:



Populating the Mappings Hierarchy

Before you can make use of the Mappings class, the rules from the XML configuration file need to be read into memory when your agent starts up. The following code illustrates how to accomplish this.

Note the Mappings class is integrated with another of the ADK's tool classes from the `com.edustructures.sifworks.tools.cfg` package: `AgentConfig`. `AgentConfig` defines a configuration file format and API that you can use to read and write agent settings from Java streams. We'll use this class to read the Mappings configuration from a file on disk:

```
import com.edustructures.sifworks.*;
import com.edustructures.sifworks.tools.cfg.*;
import com.edustructures.sifworks.tools.mapping.*;

...

// Initialize the ADK
ADK.initialize();

// Parse the configuration file
AgentConfig config = new AgentConfig();
config.read( “mappings.cfg”, false /* validate */ );

// Get the root Mappings node
Mappings root = config.getMappings();
```

That's all there is to it.

We've used the `AgentConfig` class to read and populate a Mappings hierarchy from an XML configuration file on disk, then called its `getMappings` method to return the root of the hierarchy. (Of course, you can programmatically construct the Mappings hierarchy, too, but because the `AgentConfig` class provides this service we won't cover that in this document.)

The next chapter will show you how to make use of this Mappings object at runtime.

Using the Mappings Class

The Mappings class is typically employed in the *Publisher*, *Subscriber*, and *QueryResults* message handlers of an agent when processing SIF Request, SIF Event, and SIF Response messages.

Once you've populated the Mappings hierarchy from a configuration file as shown in the previous section, follow these three steps whenever a mapping operation needs to take place:

1. Lookup a top-level Mappings node by name by calling the `getMappings` method on the root. (Note: In this document, the immediately children of the root node are considered top-level nodes.) We suggest defining a single top-level node with a name of "Default", but you could also have multiple nodes for different purposes (e.g. Inbound and Outbound nodes)

   ```
   // Lookup the "Default" mappings object
   Mappings m = root.getMappings( "Default" );
   ```

2. Once you have a reference to a top-level Mappings instance, call its **`select`** method to obtain a child Mappings that's most appropriate for the context in which it's being used. If your Mappings object has no children, the select method simply returns a reference to it. In any case, `Mappings.select` is always guaranteed to return a value.

   ```
   // Select the most appropriate child Mappings for the
   // operation at hand
   m = m.select( zoneId, sourceId, sifVersion );
   ```

3. Finally, call either form of the **`map`** method to transform your application's data to a `SIFDataObject` or vice-versa

   ```
   // Transform a HashMap of field/value pairs to a
   // SIFDataObject
   m.map( values, object );

   // Populate a HashMap from a SIFDataObject
   m.map( object, values );
   ```

Global Reference to the Mappings Root

The root Mappings node should be treated as a global resource of your agent. After reading it from a configuration file, it's a good idea to keep a reference to this object in your Agent class or some other high-level class where it is easily accessed. Note if you have selected an organization in which only one top-level node will ever exist (e.g. "Default"), it makes sense to store a reference to that node instead.

One good technique is to declare a static data member in your Agent class. You can then reference this variable whenever you need to perform a mapping operation. Note the Mappings class is thread-safe, so there is no need to synchronize it.

```
public MyAgent extends Agent
{
   // Global reference to the root Mappings instance
   public static Mappings sMappings;

   ...


   public void initialize()
      throws ADKException
   {
      // Get the root Mappings object
      Mappings root = myConfig.getMappings();

      // Store a reference to the "Default" top-level
      // Mappings node because this is the one this agent
      // will use for all mapping operations
      sMappings = root.getMappings( "Default" );
```

Calling Mappings.select

Recall that the **`select`** method is used to choose a Mappings instance that is most appropriate given the Zone ID, Source ID, and SIF Version associated with the task at hand. This method can *only* be called on a top-level Mappings instance. If you attempt to call it on the root node or a child of a top-level node an exception is thrown.

The three parameters to the **`select`** method usually come from the `SIFMessageInfo` object passed to all ADK message handlers.

For example,

```
public void onRequest(
   DataObjectOutputStream out,
   Query query,
   Zone zone,
   MessageInfo info )
      throws ADKException
```

```
{
   SIFMessageInfo sifInfo = (SIFMessageInfo)info;

   // Select the Mappings to use...
   Mappings m = MyAgent.sMappings.select(
      sifInfo.getZoneId(),
      sifInfo.getSourceId(),
      sifInfo.getSIFRequestVersion() );

   ...
```

Calling Mappings.map

Depending on what you're trying to accomplish, the **`map`** method can either populate a `HashMap` with a set of field/value pairs by transforming the elements and attributes of a `SIFDataObject`, or it can do the reverse: produce a `SIFDataObject` from a set of field/value pairs in the `HashMap`. The example above is continued to illustrate how this might be done to respond to SIF_Request messages by reading records from the application database and mapping them into StudentPersonal objects:

```
public void onRequest(
   DataObjectOutputStream out,
   Query query,
   Zone zone,
   MessageInfo info )
      throws ADKException
{
   SIFMessageInfo sifInfo = (SIFMessageInfo)info;

   // Select the Mappings to use...
   Mappings m = MyAgent.sMappings.select(
      sifInfo.getZoneId(),
      sifInfo.getSourceId(),
      sifInfo.getSIFRequestVersion() );

   // Read students from the database
   StudentRecord[] dbStudents = execQuery(query);

   for( int i = 0; I < dbStudents.length; i++ )
   {
      StudentPersonal sp = new StudentPersonal();

      // It is not recommended that RefIds be “mapped”;
      // instead, assign/lookup the object’s RefId in the
      // database and call the setRefId method directly

      sp.setRefId( RefIds.lookup( dbStudents[i].getID() ) );

      // Populate a HashMap with field values. Most agents
      // would take a data-driven approach by enumerating the
      // fields of the database record, but for this example
      // the hard-coded way will suffice.
```

```
        HashMap values = new HashMap();
        values.put( “STUDENT_NUM”, dbStudents[i].getID() );
        values.put( “SSN”, dbStudents[i].getSocSec() );
        values.put( “LAST_NAME”, dbStudents[i].getLName() );
        values.put( “FIRST_NAME”, dbStudents[i].getFName() );
        values.put( “MIDDLE_NAME”, dbStudents[i].getMName() );
        values.put( “PHONE”, dbStudents[i].getHomePhone() );

        // Now ask the Mappings object to populate the Student-
        // Personal with these values

        m.map( values, sp );

        // Note you can also set values on the StudentPersonal
        // object directly using the ADK’s methods. The Mappings
        // class is flexible; it can be used to whatever extent
        // you want to allow users to be able to customize the
        // way your agent produces SIF Data Objects.

        Demographics d = sp.getDemographics();
        if( d == null ) {
           d = new Demographics();
           sp.setDemographics( d );
        }
        sp.setGender( dbStudents[i].getGender() );

        // Now write this StudentPersonal to the zone
        out.write( sp );
    }
}
```

FAQ

1. How do I map the same field to more than one SIF element?

A group of mapping rules must be comprised of field names that are unique within the group, and each field name may only be specified once. This limitation stems from the fact that the Java HashMap does not allow duplicate entries. Each entry in a HashMap is identified by a unique key, which happens to be the field name specified in the *name* attribute of a mapping rule:

```
<field name=”MY_FIELD_NAME”>
```

To get around this problem, you can define one or more *aliases* for any field by assigning a unique alias name to the *alias* attribute. The field name of the alias is not significant; when the transformation takes place, the Mappings engine will lookup the value of the field identified by the *alias* attribute and use its value for the SIF element. For example,

```
<object object="StaffPersonal">
   <field name="TEACHER_NUM">OtherId[@Type='06']</field>
   <field name="ALT_TEACHER_NUM" alias="TEACHER_NUM">
      OtherId[@Type='ZZ']
   </field>
   ....
```

At runtime, this would result in two SIF elements mapped to the `TEACHER_NUM` field:

```
<OtherId Type='06'>105P</OtherId>
<OtherId Type='ZZ'>105P</OtherId>
```

2. I want to use literal text within a SIF element. How do I do this?

Normally, the value of a field is automatically assigned to the SIF element that is specified by a mapping rule. If you want to include additional text in the SIF element's value, append the rule with an equal sign. The Mappings engine will use the text following the equals sign as the value for the SIF element. To include the value of a field in your element, surround the field name with parenthesis and preceded it with a dollar sign (e.g. "`$(FIELD_NAME)`"):

```
<object object="StaffPersonal">
   <field name="ALT_TEACHER_NUM" alias="TEACHER_NUM">
      OtherId[@Type='ZZ']=AltID:$(TEACHER_NUM)
   </field>
   ....
```

This rule would result in the following SIF element:

```
<OtherId Type='ZZ'>AltID:105P</OtherId>
```

3. When I define several rules for the same SIF element, only the last one is included in my SIF Data Object. How do I get around this?

Whenever you specify the same SIF element more than once, and with the same set of attribute values, you must use a special "+" notation to signal the Mappings engine that there is more than one entry with the same unique signature. For example, these three rules all assign a value to "`<OtherId Type='ZZ'>`":

```
<field name="F1">OtherId[@Type='ZZ']=1:$(F1)</field>
<field name="F2">OtherId[@Type='ZZ']=2:$(F2)</field>
<field name="F3">OtherId[@Type='ZZ']=3:$(F3)</field>
```

However, at runtime the Mappings engine will produce only the last one:

```
<OtherId Type='ZZ'>3:Yellow</OtherId>
```

This happens because the Mappings engine "matches" each rule to the same element—all three have an identical *signature.* Since it processes field rules in order

from top to bottom, it replaces each with the one that follows it. To get around this problem, include a plus sign immediately before the closing bracket. This signals to the Mappings engine that there will be more than one rule with the same signature, and that it should create a unique SIF element for each one:

```
<field name="F1">OtherId[@Type='ZZ'+]=1:$(F1)</field>
<field name="F2">OtherId[@Type='ZZ'+]=2:$(F2)</field>
<field name="F3">OtherId[@Type='ZZ'+]=3:$(F3)</field>
```

Now you'll get the desired result:

```
<OtherId Type='ZZ'>1:Apple</OtherId>
<OtherId Type='ZZ'>2:Crow</OtherId>
<OtherId Type='ZZ'>3:Yellow</OtherId>
```

4. Can I combine more than one field value into a single SIF element?

Yes. Use the "`$(FIELD_NAME)`" notation to reference any field in the `HashMap`:

```
<field name="FULLNAME">
   Name/FullName=$(LAST_NAME), $(FIRST_NAME) $(MIDDLE_NAME)
/field>
```

NOTE: For the above example to work, the `HashMap` must contain a "`FULLNAME`" entry. The Mappings class does not produce a SIF element for a rule if the corresponding field is not defined in the `HashMap`.

Appendix A

Sample Mappings Configuration

The following is from the Edustructures SIF Agent for SASIxp™ factory default configuration file. It illustrates the mapping rules that ship with a commercial SIF Agent developed with the SIFWorks ADK.

```
<mappings id="Default">

    <object object="StudentPersonal">

        <!-- Student Identifiers and OtherIds -->
        <field name="PERMNUM">OtherId[@Type='06']</field>
        <field name="SOCSECNUM">OtherId[@Type='SY']</field>
        <field name="SCHOOLNUM">OtherId[@Type='ZZ'+]=SCHOOL:$(SCHOOLNUM)</field>
        <field name="SCHOOLNUM2" alias="SCHOOLNUM">OtherId[@Type='ZS']</field>
        <field name="GRADE">OtherId[@Type='ZZ'+]=GRADE:$(GRADE)</field>
        <field name="HOMEROOM">OtherId[@Type='ZZ'+]=HOMEROOM:$(HOMEROOM)</field>

        <!-- Student Name -->
        <field name="LASTNAME">Name[@Type='02']/LastName</field>
        <field name="FIRSTNAME">Name[@Type='02']/FirstName</field>
        <field name="MIDDLENAME">Name[@Type='02']/MiddleName</field>

        <!-- Mailing Address -->
        <field name="MAILADDR">
            StudentAddress[@PickupOrDropoff='NA',@DayOfWeek='NA']/Address[@Type='M']/Street/Line1
        </field>
        <field name="CITY">
            StudentAddress[@PickupOrDropoff='NA',@DayOfWeek='NA']/Address[@Type='M']/City
        </field>
        <field name="STATE">
            StudentAddress[@PickupOrDropoff='NA',@DayOfWeek='NA']/Address[@Type='M']/StatePr/@Code
        </field>
        <field name="COUNTRY">
            StudentAddress[@PickupOrDropoff='NA',@DayOfWeek='NA']/Address[@Type='M']/Country[@Code='US']
        </field>
        <field name="ZIPCODE">
            StudentAddress[@PickupOrDropoff='NA',@DayOfWeek='NA']/Address[@Type='M']/PostalCode
        </field>

        <!-- Residence Address -->
        <field name="RESADDR">
            StudentAddress[@PickupOrDropoff='NA',@DayOfWeek='NA']/Address[@Type='P']/Street/Line1
        </field>
```

```
    <field name="RESCITY">
        StudentAddress[@PickupOrDropoff='NA',@DayOfWeek='NA']/Address[@Type='P']/City
    </field>
    <field name="RESSTATE">
        StudentAddress[@PickupOrDropoff='NA',@DayOfWeek='NA']/Address[@Type='P']/StatePr/@Code
    </field>
    <field name="RESCOUNTRY">
        StudentAddress[@PickupOrDropoff='NA',@DayOfWeek='NA']/Address[@Type='P']/Country[@Code='US']
    </field>
    <field name="RESZIPCODE">
        StudentAddress[@PickupOrDropoff='NA',@DayOfWeek='NA']/Address[@Type='P']/PostalCode
    </field>

    <!-- Telephone -->
    <field name="TELEPHONE">PhoneNumber[@Format='NA',@Type='HP']</field>

    <!—Demographics & Misc -->
    <field name="BIRTHDATE">Demographics/BirthDate</field>
    <field name="ETHNICCODE">Demographics/Ethnicity[@Type='NA']/@Code</field>
    <field name="ENGPROF">Demographics/EnglishProficiency/@Code</field>
    <field name="PRIMARYLNG">Demographics/Language/@Type</field>
    <field name="ORIGYRGRAD">GradYear[@Type='Original']</field>

</object>


<object object="StaffPersonal">

    <!-- These used for SASIxp Teacher records -->
    <field name="ATCH.TCHNUM">OtherId[@Type='06']</field>
    <field name="ATCH.SOCSECNUM">OtherId[@Type='SY']</field>
    <field name="ATCH.SCHOOLNUM">OtherId[@Type='ZZ'+]=SCHOOL:$(ATCH.SCHOOLNUM)</field>
    <field name="ATCH.SCHOOLNUM2" alias="ATCH.SCHOOLNUM">OtherId[@Type='ZS']</field>
    <field name="ATCH.HOMEROOM">OtherId[@Type='ZZ'+]=HOMEROOM:$(ATCH.HOMEROOM)</field>
    <field name="ATCH.LASTNAME">Name[@Type='02']/LastName</field>
    <field name="ATCH.FIRSTNAME">Name[@Type='02']/FirstName</field>
    <field name="ATCH.MIDDLENAME">Name[@Type='02']/MiddleName</field>
    <field name="ATCH.EMAILADDR">Email[@Type='Primary']</field>
    <field name="ATCH.TELEPHONE">PhoneNumber[@Format='NA',@Type='WP']</field>
    <field name="ATCH.TELEXTN">PhoneNumber[@Format='NA',@Type='EX']</field>
    <field name="ATCH.ETHNIC">Demographics/Ethnicity[@Type='NA']/@Code</field>

    <!-- These used for SASIxp Staff records -->
    <field name="ASTF.STAFFNUM">OtherId[@Type='06']</field>
    <field name="ASTF.SOCSECNUM">OtherId[@Type='SY']</field>
    <field name="ASTF.SCHOOLNUM">OtherId[@Type='ZZ'+]=SCHOOL:$(ASTF.SCHOOLNUM)</field>
    <field name="ASTF.SCHOOLNUM2" alias="ASTF.SCHOOLNUM">OtherId[@Type='ZS']</field>
    <field name="ASTF.HOMEROOM">OtherId[@Type='ZZ'+]=HOMEROOM:$(ASTF.HOMEROOM)</field>
    <field name="ASTF.LASTNAME">Name[@Type='02']/LastName</field>
    <field name="ASTF.FIRSTNAME">Name[@Type='02']/FirstName</field>
    <field name="ASTF.MIDDLENAME">Name[@Type='02']/MiddleName</field>
    <field name="ASTF.ADDRESS">Address[@Type='M']/Street/Line1</field>
    <field name="ASTF.CITY">Address[@Type='M']/City</field>
    <field name="ASTF.STATE">Address[@Type='M']/StatePr/@Code</field>
```

```
    <field name="ASTF.COUNTRY">Address[@Type='M']/Country[@Code='US']</field>
    <field name="ASTF.ZIPCODE">Address[@Type='M']/PostalCode</field>
    <field name="ASTF.EMAILADDR">Email[@Type='Primary']</field>
    <field name="ASTF.SCHOOLTEL">PhoneNumber[@Format='NA',@Type='WP']</field>
    <field name="ASTF.TELEXTN">PhoneNumber[@Format='NA',@Type='EX']</field>
    <field name="ASTF.HOMETEL">PhoneNumber[@Format='NA',@Type='HP']</field>
    <field name="ASTF.ETHNICCODE">Demographics/Ethnicity[@Type='NA']/@Code</field>
    <field name="ASTF.BIRTHDATE">Demographics/BirthDate</field>

</object>


<object object="StudentContact">

    <field name="APRN.SOCSECNUM">OtherId[@Type='SY']</field>
    <field name="APRN.SCHOOLNUM">OtherId[@Type='ZZ']=SCHOOL:$(APRN.SCHOOLNUM)</field>
    <field name="APRN.SCHOOLNUM2" alias="APRN.SCHOOLNUM">OtherId[@Type='ZS']</field>
    <field name="APRN.LASTNAME">Name[@Type='02']/LastName</field>
    <field name="APRN.FIRSTNAME">Name[@Type='02']/FirstName</field>
    <field name="APRN.MIDDLENAME">Name[@Type='02']/MiddleName</field>
    <field name="APRN.ADDRESS">Address[@Type='M']/Street/Line1</field>
    <field name="APRN.CITY">Address[@Type='M']/City</field>
    <field name="APRN.STATE">Address[@Type='M']/StatePr/@Code</field>
    <field name="APRN.COUNTRY">Address[@Type='M']/Country[@Code='US']</field>
    <field name="APRN.ZIPCODE">Address[@Type='M']/PostalCode</field>
    <field name="APRN.WRKADDR">Address[@Type='O']/Street/Line1</field>
    <field name="APRN.WRKCITY">Address[@Type='O']/City</field>
    <field name="APRN.WRKSTATE">Address[@Type='O']/StatePr/@Code</field>
    <field name="APRN.WRKZIP">Address[@Type='O']/PostalCode</field>
    <field alias="APRN.COUNTRY" name="APRN.WRKCOUNTRY">Address[@Type='O']/Country[@Code='US']</field>
    <field name="APRN.EMAILADDR">Email[@Type='Primary']</field>
    <field name="APRN.TELEPHONE">PhoneNumber[@Format='NA',@Type='HP'+]</field>
    <field name="APRN.ALTTEL">PhoneNumber[@Format='NA',@Type='AP'+]=$(APRN.ALTTEL) $(APRN.ALTTELEXTN)</field>
    <field name="APRN.WRKTEL">PhoneNumber[@Format='NA',@Type='WP'+]</field>
    <field name="APRN.WRKEXTN">PhoneNumber[@Format='NA',@Type='EX'+]</field>
</object>


 <object object="SchoolInfo">

   <field name="SCHOOLNUM">IdentificationInfo[@Code='76']</field>

   <field name="ADDRESS">Address[@Type='SS']/Street/Line1</field>
   <field name="CITY">Address[@Type='SS']/City</field>
   <field name="STATE">Address[@Type='SS']/StatePr/@Code</field>
   <field name="COUNTRY">Address[@Type='SS']/Country[@Code='US']</field>
   <field name="ZIPCODE">Address[@Type='SS']/PostalCode</field>
   <field name="TELEPHONE">PhoneNumber[@Format='NA',@Type='TE']</field>
   <field name="FAXNUM">PhoneNumber[@Format='NA',@Type='FE']</field>

 </object>


</mappings>
```