SDA: InterSystems Clinical Data Format

Version 2024.1
2024-05-02

InterSystems Corporation 1 Memorial Drive Cambridge MA 02142 www.intersystems.com

SDA: InterSystems Clinical Data Format
InterSystems Version 2024.1 2024-05-02

For Support questions about any InterSystems products, contact:

InterSystems Worldwide Response Center (WRC)
Tel: +1-617-621-0700
Tel: +44 (0) 844 854 2917
Email: support@InterSystems.com

Table of Contents

List of Tables

1

About SDA

SDA (Summary Document Architecture) is an InterSystems format used to represent patient data. SDA is the intermediary format when transforming healthcare data from one format to another. For example, if you need to convert patient data from CDA to FHIR®, InterSystems products use XSLTs to convert the data from CDA to SDA, then use DTLs to convert the data from SDA to FHIR. For more information about how InterSystems IRIS for Health and HealthShare Health Connect use SDA to transform data, see Data Transformations in InterSystems Healthcare Products.

The SDA format can be customized and extended as needed.

2

SDA Documents

Information in SDA format is represented by an object that consists of instances of **HS.SDA3.Patient**, **HS.SDA3.Encounter**, and other classes. These classes are XML-enabled.

This topic describes:

- SDA document structure
- The Patient SDA object
- The Encounter SDA object
- Where to find further information on the SDA classes
- How to create an SDA stream from a collection of SDA objects

A separate chapter, "Customizing the SDA", describes the options available to customize and extend the SDA.

Note: SDA, when used in this document, refers to the **HS.SDA3** classes. HS.SDA classes are also present in your instance, but these are for backward compatibility only.

2.1 The Basic XML Structure of an SDA Document

The major sections in an SDA document are as follows:

XML

```
<Container>
  <Patient/>
  <Encounters/>
  <AdvanceDirectives/>
  <Alerts/>
  <Allergies/>
  <Appointments/>
  <Problems/>
  <Diagnoses/>
  <Documents/>
  <LabOrders/>
  <RadOrders/>
  <OtherOrders/>
  <Medications/>
  <Vaccinations/>
  <Observations/>
  <PhysicalExams/>
  <Procedures/>
  <FamilyHistories/>
  <IllnessHistories/>
  <SocialHistories/>
```

```
  <CustomObjects/>
  <Referrals/>
  <ClinicalRelationships/>
  <ProgramMemberships/>
  <MemberEnrollments/>
  <MedicalClaims/>
  <GenomicsOrders/>
  <CarePlans/>
  <HealthConcerns/>
  <Goals/>
  <SocialDeterminants/>
</Container>
```

The following rules apply to an SDA document:

1. There is a single <Patient>.
2. <Patient> is required and must be the first section that appears (after properties of the <Container> such as <Action>, <EventDescription>, or <SendingFacility>, see the class reference for specific details).
3. All sections other than <Patient> are optional.
4. All sections other than <Patient> may contain multiple entries, for example:

   XML

   ```
   <Procedures>
      <Procedure>
            ...
      </Procedure>
      <Procedure>
            ...
      </Procedure>
   </Procedures>
   ```

5. The <Encounters> section, if included, must appear directly after the <Patient> section.
6. All other sections may appear in any order.
7. Any entry may optionally reference an encounter number. In this case, <Encounters> must be included and must contain an <Encounter> with that <EncounterNumber>.
8. Some sections may also include an <ActionCode> or <ActionScope> to indicate that some action relative to persistence should be performed. In HealthShare Unified Care Record, these elements direct the system to perform some action when it stores the SDA, while in InterSystems IRIS for Health, you can direct that some action be taken against persisted SDA. For example, an <ActionCode> of `D` deletes a matching entry from the Unified Care Record. See the class reference for specific details relevant to each SDA section.

2.1.1 The Patient in SDA

The **HS.SDA3.Patient** class represents the patient. This class contains properties that store information such as the following:

- Demographic information and other basics, for example: name, a list of addresses, gender, marital status, race, religion, and so on
- Patient numbers

The following table shows the properties of **HS.SDA3.Patient**. Some of these properties are simple (for example, strings), some are complex objects which contain properties of their own, and some are lists of complex objects:

Table 2–1: Properties in HS.SDA3.Patient

Simple Property	Complex Object	List of Complex Objects
ActionCode*	BirthGender	Addresses
BirthOrder	BirthPlace	Aliases
BirthTime	Citizenship	Organizations
BlankNameReason	ContactInfo	OtherLanguages
Comments	CreatedBy	PatientNumbers
CommunicationPreference	DeathDeclaredBy	PriorPatientNumbers
CreatedOn	EnteredAt	Providers
DeathLocation	EnteredBy	Races
DeathTime	EthnicGroup	SupportContacts
EnteredOn	Extension	(CustomPairs)
ImmunizationRegistryStatus	FamilyDoctor	
ImmunizationRegistryStatusEffectiveDate	Gender	
InactiveMRNs	MaritalStatus	
IsDead	MothersFullName	
IsProtected	Name	
MothersMaidenName	Occupation	
MPIID	PrimaryLanguage	
ProtectedEffectiveDate	PublicityCode	
PublicityEffectiveDate	Race	
UpdatedOn	Religion	

*A <Patient> SDA section may include an <ActionCode>. An <ActionCode> of `D` instructs HealthShare Unified Care Record to delete a patient with the matching MRN. An <ActionCode> of `R` instructs Unified Care Record to replace an existing patient with this patient, based on MRN. See the class reference for specific details.

2.1.2 Encounters in SDA

An encounter encompasses all of the medical information related to a specific medical incident. The **HS.SDA3.Encounter** class represents a medical encounter of a patient. Other sections in the SDA, representing orders, procedures, exams and the like can refer to the encounter number, and thus be tied together.

The following table shows the properties of **HS.SDA3.Encounter**, some of which are simple (for example, strings), some of which are complex objects (containing properties of their own), and some of which are lists of complex objects:

Table 2–2: Properties in HS.SDA3.Encounter

Simple Property	Complex Object	List of Complex Objects
ActionCode*	AdmissionSource	AttendingClinicians
AccountNumber	AdmissionType	ConsultingClinicians
AssignedBed	AdmitReason	Guarantors
AssignedRoom	AdmittingClinician	HealthFunds
AssignedWard	DiagnosisRelatedGroup	RecommendationsProvided
EmergencyAdmitDateTime	DischargeLocation	Specialties
EncounterMRN	EncounterCodedType	(CustomPairs)
EncounterMRNAA	Extension	
EncounterNumber	HealthCareFacility	
EncounterType	EnteredAt	
EndTime	EnteredBy	
EnteredOn	Priority	
ExpectedAdmitTime	PublicityCode	
ExpectedDischargeTime	ReferringClinician	
ExpectedLOAReturnTime	SeparationMode	
ExternalId		
FromTime		
PreAdmissionNumber		
PriorBed		
PriorRoom		
PriorWard		
PriorVisitNumber		
ToTime		
TransferredFromED		
UpdatedOn		
VisitDescription		

*An encounter may also include an <ActionCode> directing HealthShare Unified Care Record to take an action. Possible values include:

- D = Delete
- E = Delete if Empty
- C = Clear All
- R = Replace

See the class reference for specific details of the <ActionCodes> for an encounter.

2.1.3 For More Information on SDA Classes

For complete details on the SDA format, see the class reference for **HS.SDA3.Container** and the other classes in the **HS.SDA3** package.

To access the class reference for the SDA classes:

1. Either click on the "Class Reference" link in the InterSystems launcher (Windows only), or navigate to the following URL in your browser, using the <baseURL> for your instance:
   `https://<baseURL>/csp/documatic/%25CSP.Documatic.cls`.
2. Log in, if required.
3. Using the drop-down list at the top of the left-hand pane, change to the HSLIB namespace.
4. Clear the **Percent Classes** check box to reduce visual clutter.
5. Drill down the hierarchy of package names to **HS**, then **SDA3** and then to the class of interest.

You may also view the XML document structure for any SDA class:

1. In the Management Portal, select an Edge Gateway namespace and navigate to **Home** > **Interoperability** > **Interoperate** > **XML** > **XML Schema Structures**.
2. In the **HS.SDA3** schema, select the name of the class whose structure you want to view.

Tip: To navigate through the HS.SDA3 XML schema, you may find it easiest to select Container, and then drill down from there.

The "CDA and SDA Annotations" feature provides an alternate view into the SDA class structure.

2.2 Creating an SDA Stream from SDA Objects

Note: This section describes a methodology that will accommodate an SDA of any size. Rather than instantiate an entire patient record as an SDA object (which in the context of HealthShare Unified Care Record may contain thousands of encounters), the approach described here instantiates only the individual sections, and then constructs the SDA stream manually. If you are certain that your SDA is small, then you can instantiate the entire SDA as an object and write the stream directly.

If you have a set of SDA objects and you wish to combine them to produce an SDA stream, use the following methodology:

1. Instantiate a new stream.
2. Write the <Container> opening tag to the stream.
3. For the Patient object, which must come first, use the **ToQuickXML()** method of the SDA object to write the object to the stream as XML.
4. For subsequent objects, which should be grouped by type (Encounter, LabOrder etc.), use the **StartXMLList()** method of the SDA object to write the opening tag for the collection, for example, `<Encounters>`. Remember that <Encounters>, if included, must come directly after <Patient>.
5. Use the **ToQuickXML()** method to write each object of that type to the stream.
6. When finished with a collection of a particular type, use the **EndXMLList()** method to write the closing tag for the collection.

7. Repeat for other collections.
8. When finished, write the </Container> closing tag.

2.3 Streamlet Matching

Each streamlet class has distinct matching criteria defined in that class's `MATCHINGS` parameter, which is inherited from HS.SDA3.Streamlet.Abstract. Streamlet matching logic allows HealthShare to determine whether or not a streamlet is new, or an update to existing data. Whether or not two streamlets will be compared for matching and deduplication depends on their `EncounterNumber` properties:

- In the event that two streamlets have differing `EncounterNumber`s, or one has a value for `EncounterNumber` and the other does not, the streamlets are not eligible for comparison.
- In the event that both streamlets have the same `EncounterNumber`, they are eligible for comparison.
- In the event that neither streamlet has a value for `EncounterNumber`, they are eligible for comparison.

Given eligible streamlets, the matching process happens in three phases:

1. First, when a streamlet is ingested by an Edge Gateway, its data is compared to streamlets of the same type with the same MRN. In a system with multiple Edge Gateways, comparisons are handled by the Access Gateway's aggregation cache. For more information on this process, consult the class reference for the **HS.SDA3.Streamlet.Abstract.Aggregate()** method.
2. If it contains an external identifier, this is used to match it to other streamlets of the same type with that external identifier.
3. If neither of the above checks yield a match, HealthShare checks each streamlet-specific `MATCHINGS` key-value pair against other records of the same streamlet type.

In the event that none of these checks yield a match, the streamlet is recognized as new. If any of the checks yield a match, the streamlet is recognized as an update of an existing streamlet and aggregated together with it. This is called deduplication. The `MATCHINGS` parameter may be modified, as described in "Customizing the SDA" later in this book.

The `MATCHINGS` parameter takes the form `MatchList1 || MatchList2` where each `MatchList` item takes the form `MatchType/MatchProp1,MatchProp2,.../NullProp1,NullProp2,...`.

These elements are defined as follows:

- `MatchType` is an arbitrary name for the matching type.
- `MatchProp` is the name of a property in the SDA streamlet class used for matching. In some complex cases, this can be a transient or computed property. Where there are two or more `MatchProp` elements, matching must be performed on *all* of them.
- `NullProp` is the name of a property used for matching which is optional. An empty `NullProp` counts as matching anything. Where there are two or more `NullProp` elements, matching is performed on any combination of them.
- An asterisk after either a `MatchProp` or `NullProp` indicates that the element is a `CodeTableDetail` to be matched on `Code` and `SDACodingStandard`.

Consider the following code from a hypothetical custom Allergy streamlet:

```
Parameter MATCHINGS = "ALG/Allergy*/FromTime,AllergyCategory* ||
ZATC/ATCCode*/FromTime";
```

In this case, Unified Care Record will attempt to match according to the following criteria in order:

1. Match on `Allergy`, which is a `CodeTableDetail`. Optionally, match on `FromTime` or `AllergyCategory`–also a `CodeTableDetail` if either exists.
2. Match on `ATCCode`, which is a `CodeTableDetail`. Optionally, match on `FromTime` if it exists.

3

Customizing the SDA

In most cases, the SDA structure is sufficient to handle all of the data coming through the system. However, if you do need to capture additional data, extending the SDA is very straightforward.

There are two *recommended* ways to extend the SDA, depending on your needs:

- Extend an existing SDA object by using the provided extension class. This option is available in all products. Each SDA class has a corresponding extension class, which by default has no properties. You can extend the capabilities of the SDA by declaring properties on the extension classes.
- Create a custom SDA container and a corresponding custom streamlet. This option is available only in HealthShare Unified Care Record. You can define a custom SDA container that adds extra sections to the SDA. This definition includes custom SDA streamlets. Choose this option if you are running Unified Care Record and you want to create new streamlet types to handle new data types.

An additional option, available in all products, is to add a custom name/value pair to an existing SDA section or a new custom SDA section. You might prefer this approach if the data you want to capture is structured as simple pairs (for example, a questionnaire with a set of questions and answers). However, this approach has numerous limitations:

- It is complicated when storing complex objects or collections of data with 0:* cardinality.
- It cannot store custom data on serial properties of streamlets.
- This data cannot easily be propagated to Health Insight's database.
- There is no way to easily implement datatype validation on custom data.

Using name/value pairs is not recommended because of the above limitations. If you believe that this approach may be appropriate for your situation, review the "Extending the SDA with Name/Value Pairs" section of this guide and consult InterSystems support if needed.

3.1 Using the Extension Classes to Customize the SDA

Each SDA data class (with some exceptions, described below) has a corresponding extension class and a property that refers to it. For example, class **HS.SDA3.Allergy** has property Extension of type **HS.Local.SDA3.AllergyExtension**. By default, these extension classes have no properties. You can customize your SDA by adding properties to these classes.

All of the extension classes can be found in the HSCUSTOM database, which was automatically created on installation or upgrade.

To use the extension classes:

1. Add properties to the extension classes as needed.
2. Recompile your classes.
3. Populate your extension properties.

If you are running HealthShare Unified Care Record, you can also customize the streamlet class associated with an SDA extension class.

After you have added properties to an extension class, you can use them in code and, in Unified Care Record, display them in the Clinical Viewer.

Important: If you are also using Health Insight for analytics, you must propagate your SDA extensions to Health Insight and also modify the HSAA.Local.<*SDAType*>Extension classes in Health Insight. See "SDA Extensions and Health Insight" in the *Health Insight Installation and Configuration Guide*.

If you are viewing this documentation in InterSystems IRIS for Health™, click here to link to the HealthShare Health Insight documentation on extending the SDA.

General Notes on Extension Classes

All code table extension classes extend **HS.SDA3.CodeTableExtension**; all other extension classes extend **HS.SDA3.DataType**.

The following classes do not have associated extension classes:

- **HS.SDA3.SuperClass** and **HS.SDA3.CodeTableDetail**, because they are only used as superclasses.
- **HS.SDA3.Container**, because there is a separate mechanism for extending this class, as described later in this chapter.
- **HS.SDA3.CustomObject**
- **HS.SDA3.ClinicalRelationship** and **HS.SDA3.ProgramMembership**, because they contain registry data and are not part of the streamlet architecture

The Patient class is of special significance, as the main parent class for all patient data. The Patient SDA object has a corresponding registry class **HS.Registry.Patient**, and associated message classes. Each of these contains a property called Extension, which is a reference to the **HS.Local.SDA3.PatientExtension** class. Therefore, any extension properties that you create are available to those associated classes as well as to the Patient class itself.

3.1.1 Adding Properties to the Extension Class

To customize the SDA, add properties to the extension classes in the HS.Local package in the HSCUSTOM namespace.

In your IDE, switch to the appropriate namespace and open the desired extension classes. Add new properties as needed.

You can add properties of the following types:

- %String
- Any HS.SDA3.<*DataType*> such as Boolean or Numeric
- Any existing HS.SDA3 serial class
- Any custom serial class that you have created, which must extend HS.SDA3.DataType. Any custom serial classes should not be defined in HealthShare packages that are available out of the box, like HS.Local or HS.SDA3. This ensures that your class names will not conflict with any future standard SDA3 class names. For example, you should name your custom serial class something like ZUser.HS.MySDASection, rather than HS.SDA3.MySDASection. You can add custom package mappings to HSCUSTOM.

- Any new code tables that you have created, which must extend HS.SDA3.CodeTableTranslated; the following code tables are *not* translatable:

Table 3–1: Code Tables that are Excluded from Translation

HealthCareFacility	CareProviderType	Country
Organization	City	County
User	State	Trust
CareProvider	Zip	

In the following example, the Sneeziness and FlowerType properties have been added to the **HS.Local.SDA3.AllergyExtension** class. Sneeziness is a string, and FlowerType is a custom type.

Class Definition

```
Class HS.Local.SDA3.AllergyExtension Extends HS.SDA3.DataType
{
Parameter STREAMLETCLASS = "HS.SDA3.Streamlet.Allergy";
Property Sneeziness As %String;
Property FlowerType As My.Local.Type.FT;
}
```

3.1.2 Recompiling The Classes

After you have made your changes, you must recompile the relevant classes. For greatest efficiency, InterSystems recommends that you compile the entire HS.SDA3 package.

If you have any additional instances, make sure that your new code (including associated mappings) is deployed to each instance.

Note: If you are deploying in a mirror:

1. Apply code changes in HSCUSTOM to the backup member first.
2. Fail over.
3. Apply the changes to the new backup member.

3.1.3 Populating The Extension Properties

Now that your extension properties are in place, you will need to ensure that they will be populated by incoming data from source systems.

Depending on your needs and your system setup, you can accomplish this in several ways:

- With HealthShare Unified Care Record, you can create a pipeline. See the chapter "Creating Pipelines and Inbound Processes" in *Configuring Unified Care Record for Data Feeds* for details.
- With InterSystems IRIS for Health or Health Connect, you can create a DTL transformation. See *Developing DTL Transformations* for details. To ensure that the desired extension properties will be available to you when creating your DTL, you will need to export the XML schema from HSCUSTOM and import it to the desired namespace, as follows:

  1. In Terminal in the HSCUSTOM namespace, run the following:

     ```
     do ##class(HS.SDA3.Container).ExportXMLSchema()
     ```

You will be prompted to enter the desired filename for the export file. Specify the full path and filename as desired, with an .xsd extension.

2. In the Management Portal, in the namespace where you intend to create a DTL, select **Interoperability** > **Interoperate** > **XML** > **XML Schema Structures**.
3. Select the **Import** button and navigate to your export file. The exported schema is imported to the desired namespace.
4. You can now open a new DTL transformation and use the extension properties. If your extension class has a single property, that custom property does not appear under the **Extension** property of the DTL diagram. For example, if your extension class has two properties, BloodType and Sneeziness, you can expand the **Extension** property in the DTL diagram to display the two custom properties. However, if the extension class has only one property, BloodType, the DTL diagram does not display **BloodType** under the **Extension** property; the BloodType property gets "rolled up" into the parent Extension property.

- With any product: If you are using the HS.Gateway.HL7.HL7ToSDA3 class to convert HL7 messages to SDA:
  1. Create a new class extending **HS.Gateway.HL7.HL7ToSDA3**.
  2. In the new class, implement callback methods to handle the SDA extensions you have added. The callback methods are named in the form **On<StreamletName>()**, for example `OnAllergy()`.
  3. Edit the HS.Gateway.HL7.InboundProcess operation and change its HL7ToSDA3Class property to refer to your new class.

The following lists specific cases and considerations for populating SDA extensions:

Transforming between HL7™ FHIR® DSTU2 and Customized SDA

For each SDA class for which you have created extensions, you will need to create a custom copy of the corresponding DTLs in order to use your new extensions with FHIR DSTU2. Please refer to *FHIR Support in InterSystems Products* for information on how to customize the FHIR/SDA DSTU2 DTLs.

Unrecognized SDA Extensions

In order to accommodate raw SDA shared between environments with disparate sets of extensions, all SDA3 classes use the XML processing directive, `XMLIGNOREINVALIDTAG = 1`. If a system attempts to load an SDA stream with unrecognized Extension properties into an object, the unrecognized properties will be ignored. If TraceOperations is enabled, a warning trace will be logged when this occurs.

Patient Extension

To ensure that data stored in HS.Local.SDA3.PatientExtension is available and used consistently across a HealthShare deployment, the HS.Types.PatientInfo class supports a property named Extension which is of the type HS.Local.SDA3.PatientExtension. As a result, all sub-classes of HS.Types.PatientInfo also inherit the extensions; those classes are listed in the table below. Additionally, HS.Registry.Patient has an Extension property of type HS.Local.SDA3.PatientExtension and the code in the **AddUpdateHub()** method in HS.Hub.MPI.Manager handles the patient extension classes and stores them in the HS_Registry.Patient table.

Important: Any persistent class such as HS.Registry.Patient *must* be recompiled after modifications are made to HS.Local.SDA3.PatientExtension.

Table 3–2: Classes Inheriting from or Using HS.Types.PatientInfo

HS.Audit.Criteria	HS.Message.PatientSearchRequest
HS.Gateway.Access.QueryProcess	HS.Message.QueueForFetchRequest
HS.Hub.HSWS.WebServices.Containers.Patient	HS.Message.RemovePatientRequest
HS.Hub.MPI.FetchStreamlet	HS.MPI.Initiate.Operations
HS.Hub.MPI.Manager	HS.MPI.Native.PatientRecord
HS.Message.AddPatientRequest	HS.MPI.SecondaryMPI
HS.Message.AddUpdateHubRequest	HS.MPI.SureScripts.Operations
HS.Message.FindAutoLinkMatchRequest	HS.Types.PatientInfo
HS.Message.GetCompositeRecordResponse	HS.Types.PatientSerial
HS.Message.MedicationHistoryRequest	HS.UI.ClinicianPortal
HS.Message.PatientBatchFetchRequestAsync	HS.UI.PatientSearch
HS.Message.PatientMPIMatch	HS.UI.PatientSearchUtil
HS.Message.PatientSearchMatch	

3.1.4 Customizing Streamlet Classes

Important: This section applies to HealthShare Unified Care Record only.

Customizing streamlet classes is an advanced task. Carefully review the limitations described in this section, and contact InterSystems customer support if needed.

In addition to adding properties to SDA objects as described in the preceding sections, you can also customize some of the behavior of the associated streamlet classes. This section describes that process and the key limitations that you should keep in mind.

You might choose to customize streamlet classes if you want to:

- Change the matching logic for the associated SDA class
- Change the validation logic for the associated SDA class
- Change which fields are and are not treated as translated code
- Override the behavior of callback methods

3.1.4.1 Procedure

The general approach is as follows:

1. In the HSCUSTOM namespace, extend the streamlet class that corresponds to the desired SDA object. You should use a class package name or subpackage name of your own creation. If you choose to save your custom class in

HS.Local, make sure that you are using a subpackage name that begins with Z, such as HS.Local.ZMyPackage. You can add custom package mappings to HSCUSTOM.

2. Adjust the *STREAMLETCLASS* parameter on the SDA extension class accordingly. For example, **HS.Local.SDA3.AllergyExtension** will already have the following:

   ```
   Parameter STREAMLETCLASS = "HS.SDA3.Streamlet.Allergy";
   ```

   If you extend this streamlet class and create new class MyPackage.SDA3.Streamlet.ZAllergy, you would want to modify this parameter to:

   ```
   Parameter STREAMLETCLASS = "MyPackage.SDA3.Streamlet.ZAllergy";
   ```

3. Customize callback methods, or add new methods, or add new transient properties, as desired. Please review the limitations described below before you begin.

The following example defines a new Allergy streamlet with custom matchings and validation for a new Sneeziness property.

Class Definition

```
Class ZHS.SDA3.Streamlet.ZAllergy Extends HS.SDA3.Streamlet.Allergy
{
/// Adding a fallback match
/// Previously, it matched based on the Allergy code table,
/// with nullable matches on FromTime and AllergyCategory code table
/// If that doesn't match, we want it to also try using the ATCCode code table
/// (with nullable match on FromTime)
Parameter MATCHINGS = "ALG/Allergy*/FromTime,AllergyCategory*|| ZATC/ATCCode*/FromTime";
/// Adding some validation for our extension property
Method OnValidate() As %Status
 {
  // We use IsDefined to avoid instantiating (swizzling) a null serial property, such as Extension
  // If we know that we'll always have data under Extension, this isn't needed
  If ..SDA.IsDefined("Extension") {
  Set tSneez=..SDA.Extension.Sneeziness
  If tSneez'?1.N1" Tissue".E, tSneez'?1.N1" Hankie".E {
  Quit $$$ERROR($$$GeneralError, "Sneeziness must be in terms of Tissues or Hankies.")
}
}
Quit ##super()
}
}
```

3.1.4.2 Restrictions

When you are customizing a streamlet class, keep in mind the following limitations:

- Do *not* create a customized streamlet class in any package name starting with "HS", unless it is a subpackage of HS.Local that begins with Z, such as HS.Local.ZMyPackage.
- Streamlet customizations apply in-memory only and will *not* affect the storage of the streamlet. When a streamlet is stored, its type will always be HS.SDA3.Streamlet.<*type*> when stored.
- The only methods that you may extend in a custom streamlet class are the event callback methods listed below; all other event handlers are marked as `FINAL` in the superclass and may not be modified.
  - **OnInactivate()**
  - **OnMatchActionScope()**
  - **OnBeforeMatch()**
  - **OnValidate()**
  - **OnBeforeSave()**
  - **OnAfterSave()**

- You cannot customize the `OnDeleteSQL` or `OnDeleteHandler` callback methods, or any non-callback methods.
- When customizing other callback methods (OnXXX), you should always invoke `##super`.
- InterSystems recommends that any new methods or properties you create begin with the letter Z to avoid future conflicts.
- You can customize only the following parameters: *DATEPROPERTY*, *MATCHINGS*, *TRANSLATIONS*, and *ACTIONSCOPES*. You cannot customize any other parameters.
- If you modify the *MATCHINGS* parameter, any new match type that you create should begin with the letter Z. Do not modify any existing match types. You can delete existing match types.
- The *DATEPROPERTY* and *MATCHINGS* parameters can access properties on the extension class by specifying `Extension.<property>`
- If you modify the *MATCHINGS* parameter, you must perform the additional steps described in Recalculating Matches below.
- You cannot add or change streamlet metadata:
  - persistent properties
  - relationships
  - indexes
- The Patient streamlet includes a callback method called **OnAggregateExtensionImpl()** which controls how patient data in extension classes is aggregated. By default, this method takes the extension data from the patient record that is determined to be the best record among those being aggregated. If desired, you can override this method to provide customized aggregation behavior.
- There is a transient multidimensional property available on streamlet classes named Stash that can be used to pass data from the OnBeforeSave() method to the OnAfterSave() method.

After customizing a streamlet class, you should recompile the HS.SDA3 package as described previously.

3.1.4.3 Recalculating Matches

If you have modified the *MATCHINGS* parameter for your custom streamlet, you must reevaluate your data to check for new matches and reconcile them if needed. Three utilities are provided for this purpose, as described below.

Each of these utilities takes a streamlet type as its argument. For standard streamlets or extensions of standard streamlets, this is the standard type, such as `Allergy`; for a custom streamlet, you should use the full classname, such as `HSCustom.SDA3.Streamlet.ZAllergy`.

1. First, use **RecalculateMatches** to recalculate existing matches. For example:

   ObjectScript

   ```
   Do ##class(HS.SDA3.Streamlet.Utility.RecalculateMatches).Start("HSCustom.SDA3.Streamlet.ZAllergy")
   ```

2. Next, use **FindMatches** to identify any duplicates. For example:

   ObjectScript

   ```
   Do ##class(HS.SDA3.Streamlet.Utility.FindMatches).Start("HSCustom.SDA3.Streamlet.ZAllergy")
   ```

   You can view the contents of the `^HS.SDA3.Streamlet.MatchGroups` global to see whether this utility found anything.

3. If **FindMatches** identifies any duplicates, you may wish to run **ReconcileMatches**, which reconciles all such matches:

ObjectScript

```
 Do
##class(HS.SDA3.Streamlet.Utility.ReconcileMatches).ReconcileMatches("HSCustom.SDA3.Streamlet.ZAllergy")
```

The **ReconcileMatches()** method achieves this by accepting the most recent streamlet, and marking all others that matched it as deleted. If you want to apply different criteria for determining which streamlet to accept from a group of matches, you can extend the **ReconcileMatches()** method or create your own custom method for this purpose.

FindMatches and **RecalculateMatches** can optionally take a second parameter, which is a boolean that specifies whether the processing should resume from the method's last run or start over from the beginning. For example:

ObjectScript

```
Do ##class(HS.SDA3.Streamlet.Utility.FindMatches).Start("HSCustom.SDA3.Streamlet.ZAllergy", 1)
```

If this parameter is set to 0 or omitted, the method will process all streamlet IDs for the specified type. If it is set to 1, the method will resume from the last successfully processed streamlet ID, which is stored in the global `^ISC.HS.Streamlet.Loader("Last")`.

3.1.5 Using Custom SDA Properties

You can now use your custom properties in ObjectScript code. The custom extension classes and their properties behave the same as any other class.

For example, if you have added a new Sneeziness property to the Allergy extension class, you can use it as follows:

ObjectScript

```
Set tAllergy=##class(HS.SDA3.Allergy).%New()
Set tAllergy.Extension.Sneeziness="3 Tissues"
```

You can also display your new property in the Clinical Viewer. For detailed instructions, see Adding SDA Fields to the Clinical Viewer in the *Customizing the Clinical User Interfaces* guide.

Custom SDA extensions can also be accessed and used by HealthShare Patient Index and Health Insight. Consult the product documentation for details and additional configuration.

3.2 Customizing the SDA by Creating a Custom SDA Container

Important: This functionality is available only in HealthShare Unified Care Record.

A custom SDA container allows you to create custom SDA sections that contain complex properties and match keys. In order to create a custom SDA container you must perform the following steps in the correct order:

1. For each custom SDA section, define a custom SDA data class
2. For each custom SDA data class, define an SDA streamlet class to store the data
3. Define a custom SDA container class that includes each of your custom sections as a property
4. Register your custom SDA container in the configuration registry

Once you complete these steps, SDA processing will accept, store, and aggregate your custom SDA container. Your custom SDA data will be available in the aggregation cache for custom display in the Clinical Viewer and for inclusion in custom reports.

3.2.1 Creating a Custom SDA Data Class

The first step in creating a custom SDA container is to define a custom SDA data class for each of your custom SDA sections:

1. Create a new class that extends **HS.SDA3.SuperClass**.

   If you define a custom SDA data class for one of your custom SDA sections and want Health Insight to be able to interpret and store the data, review the following section before proceeding.

2. Name your class something like: User.ZMySection. It is good practice to prefix your short class names with a "Z" to ensure that they do not conflict with any future standard SDA3 class names.

   Your class will inherit the standard set of SDA properties:

ActionCode	EncounterNumber	EnteredOn	ToTime
ActionScope	EnteredAt	ExternalId	UpdatedOn
CustomPairs	EnteredBy	FromTime	

3. You may add your own SDA properties. If you use data types other than strings, then use the existing SDA3 datatype classes such as Blob, Boolean, Numeric, or TimeStamp. This ensures support for the delete mechanism, where a pair of double quotes triggers a delete.

   Double quote deletion does *not* apply to elements in collections. In order to delete such elements, you must assign `ActionCode = 'R'` to the streamlet in question, triggering a replacement of the streamlet which clears all existing collections.

   You can add properties of the following types:

   - %String
   - Any HS.SDA3.<*DataType*> such as Boolean or Numeric
   - Any existing HS.SDA3 serial class
   - Any custom serial class that you have created, which must extend HS.SDA3.DataType. Any custom serial classes should not be defined in HealthShare packages that are available out of the box, like HS.Local or HS.SDA3. This ensures that your class names will not conflict with any future standard SDA3 class names. For example, you should name your custom serial class something like ZUser.HS.MySDASection, rather than HS.SDA3.MySDASection.
   - Any new code tables that you have created, which must extend HS.SDA3.CodeTableTranslated; the following code tables are *not* translatable:

     Table 3–3: Code Tables that are Excluded from Translation

HealthCareFacility	CareProviderType	Country
Organization	City	County
User	State	Trust
CareProvider	Zip	

4. The standard naming convention for SDA classes is singular, like "Medication", where the container has a list with the standard plural, like "Medications", as illustrated below in XML:

   XML

   ```
   <Medications>
      <Medication>
      </Medication>
      <Medication>
      </Medication>
   </Medications>
   ```

   If adding an "s" to the end of your custom section does not work (for example, Diagnosis/Diagnoses), then implement the methods **StartXMLList()** and **EndXMLList()** in your SDA data class, to output the open and close collection tags on the container:

   Class Member

   ```
   ClassMethod StartXMLList()
   {
       Quit "<Diagnoses>"
   }
   ```

   Class Member

   ```
   ClassMethod EndXMLList()
   {
       Quit "</Diagnoses>"
   }
   ```

5. Compile your class.

   Note: If you receive a <PROTECT> error, temporarily make the HSLIB database writable, recompile your class, then make HSLIB read-only again.

The example below illustrates a custom data class for transplant information:

Class Definition

```
Class User.ZTransplant Extends HS.SDA3.SuperClass
{

Property OrganType As %String;

Property TransplantPhysician As HS.SDA3.CodeTableDetail.CareProvider;

Storage Default
{
<Data name="SuperClassState">
     <Subscript>"SuperClass"</Subscript>
     <Value name="1"><Value>ActionCode</Value></Value>
     <Value name="2"><Value>ActionScope</Value></Value>
     <Value name="3"><Value>EnteredBy</Value></Value>
     <Value name="4"><Value>EnteredAt</Value></Value>
     <Value name="5"><Value>EnteredOn</Value></Value>
     <Value name="6"><Value>ExternalId</Value></Value>
     <Value name="7"><Value>EncounterNumber</Value></Value>
     <Value name="8"><Value>FromTime</Value></Value>
     <Value name="9"><Value>ToTime</Value></Value>
     <Value name="10"><Value>Deleted</Value></Value>
     <Value name="11"><Value>UpdatedOn</Value></Value>
     <Value name="12"><Value>CustomPairs</Value></Value>
     <Value name="13"><Value>OrganType</Value></Value>
     <Value name="14"><Value>TransplantPhysician</Value></Value>
</Data>
<SequenceNumber>5</SequenceNumber>
<Type>%Library.SerialState</Type>
}
}
```

3.2.1.1 Making an Analogous Custom Data Class in Health Insight

If you define a custom SDA data class for one of your custom SDA sections and want Health Insight to be able to interpret and store the data, you should make an analogous custom data class in Health Insight.

In this case, both your custom SDA streamlet class and your custom SDA data class should exist in the HSCUSTOM namespace and be package mapped to your Analytics namespace. These classes should be compiled in the Analytics namespace as well.

In order to create your custom SDA data class in Health Insight, do the following:

1. Create a new class that extends HSAA.IndexCommonData. Ensure that the class name avoids the HSAA package.
2. Name your class something like: User.ZMyHISection. It is good practice to prefix your short class names with a "Z" to ensure that they do not conflict with any future standard SDA3 class names. Ensure that your Health Insight custom data class does not have the exact same name as your custom SDA data class.

   Your class will inherit the standard set of SDA properties:

ActionCode	EncounterNumber	EnteredOn	ToTime
ActionScope	EnteredAt	ExternalId	UpdatedOn
CustomPairs	EnteredBy	FromTime	

3. You may add your own SDA properties. If you use data types other than strings, then use the existing HSAA.Internal.Boolean, HSAA.Internal.Numeric, or HSAA.TimeStamp datatype classes.

   You can add properties of the following types:

   - %String
   - Any HSAA.Internal.<*DataType*> such as Boolean or Numeric
   - Any existing HSAA serial class
   - Any custom serial class that you have created, which must extend HSAA.Internal.DataType. Any custom serial classes should not be defined in Health Insight packages that are available out of the box, like HSAA. This ensures that your class names will not conflict with any future standard Health Insight class names. For example, you should name your custom serial class something like ZUser.HSAA.MySDASection, rather than HSAA.MySDASection.
   - Any new code tables that you have created, which must extend HSAA.Internal.Interface.CodeTableTranslated; the following code tables are *not* translatable:

     Table 3–4: Code Tables that are Excluded from Translation

HealthCareFacility	CareProviderType	Country
Organization	City	County
User	State	Trust
CareProvider	Zip	

4. If the SDA custom data class uses an HS.SDA3.CodeTableDetail class, use the analogous HSAA.Interface.CodeTableDetail class, or, if none exists, the analogous HSAA.Internal.Interface.CodeTableDetail class.
5. If you want to use a Tag or other fields in Health Insight, add them to the class rather than extending any other HSAA class.

6. Compile your class.

Once you have compiled the custom Health Insight data class, register it. For instructions on how to register your custom Health Insight data class, see "Registering Custom Container Classes" in the Health Insight Installation and Configuration Guide. Follow the procedure described in that section, but enter your custom SDA data class name for the **SDA Source Class** field and your analogous Health Insight custom data class for the **Health Insight Class Name** field.

Note: If you are viewing this documentation in InterSystems IRIS for Health™, click here to link to the HealthShare Health Insight documentation on extending the SDA.

3.2.2 Creating a Custom SDA Streamlet Class

The second step in creating a custom SDA container is to define a custom SDA streamlet class to store the data for each of your custom data classes:

1. Create a new class that extends both **HS.SDA3.Streamlet.Abstract** and %Persistent.
2. Name your class something like: User.Streamlet.ZMySection. It is good practice to prefix your short class names with "Z" to ensure that they will not conflict with any future standard SDA3 class names.

   The class name above illustrates the standard naming convention for streamlet classes: <*PackageName*>.streamlet.<*SDADataClassname*>. If you do not follow this naming convention, then you must implement the **GetStreamletClass()** method of **HS.SDA3.SuperClass** on your SDA data class to output the name of your streamlet class. The default code is:

```
// for a standard SDA class, we get the streamlet class from the extension class,
// to let the extension override it.  Classes that don't allow extension
// can override this method
If pType'["." Quit $Parameter("HS.Local.SDA3."_pType_"Extension","STREAMLETCLASS")
// For custom ones, if we have a parameter, use it
If $Parameter(,"STREAMLETCLASS")'="" Quit $Parameter(,"STREAMLETCLASS")
//The code below is for backwards compatibility - it predates the parameter
//and can be overridden if need be, but really one should just use the parameter
Quit $P(pType,".",1,$L(pType,".")-1)_".Streamlet."_$P(pType,".",$L(pType,"."))
```

3. Enter values for the following parameters.

Parameter	Description	Required?
INFOTYPE	Either a predefined information type, or a custom information type.	Required
SDACLASS	The name of your SDA data class.	Required
DATEPROPERTY	The name of the property to use for filtering by date. If this is missing, defaults to ToTime.	Optional
MATCHINGS	Describes how to perform matching for this section in HealthShare Unified Care Record. Does not apply to Health Connect. For a description of the matchings, see the class reference for **HS.SDA3.Streamlet.Abstract**. For examples, see the class reference for any of the SDA3 streamlet classes.	Required
ACTIONSCOPES	If this Streamlet class has a limited set of valid *ACTIONSCOPES* values, this parameter should be a comma-delimited list (with leading and trailing commas) of those values. For HealthShare Unified Care Record only; does not apply to Health Connect.	Optional

4. Optionally enter code for the callbacks you want to implement in HealthShare Unified Care Record (not applicable to Health Connect or InterSystems IRIS for Health):
   - **OnInactivate()**
   - **OnMatchActionScope()**
   - **OnBeforeMatch()**
   - **OnValidate()**
   - **OnBeforeSave()** — You can use the transient multidimensional property Stash to pass data from OnBeforeSave() to OnAfterSave()
   - **OnAfterSave()**
5. Compile your class.

The example below illustrates a custom streamlet class for transplant information:

Class Definition

```
Class User.Streamlet.ZTransplant Extends (HS.SDA3.Streamlet.Abstract, %Persistent)
{

Parameter INFOTYPE = "PRC";

Parameter SDACLASS = "User.ZTransplant";

Parameter DATEPROPERTY = "EnteredOn";

Parameter MATCHINGS = "PRC/EnteredOn";

Storage Default
{
<ExtentSize>100000</ExtentSize>
<SequenceNumber>5</SequenceNumber>
<Type>%Library.Storage</Type>
}
}
```

3.2.3 Creating a Custom SDA Container Class

The third step in creating a custom SDA container is to define a custom SDA container class that includes your custom SDA sections:

1. Create a new class that extends **HS.SDA3.Container**.
2. Name your container class something like: User.ZMyContainer. It is good practice to prefix your short class names with a "Z" to ensure that they do not conflict with any future standard SDA3 class names.
3. Add a property for each new SDA data section. Name the property ZMySections (plural) or, if you implemented the **StartXMLList()** and **EndXMLList()** methods in your data class, use that value as the property name. The property should be a list of User.ZMySection (singular).
4. Compile your class.

   Note: If you receive a <PROTECT> error, temporarily make the HSLIB database writable, recompile your class, then make HSLIB read-only again.

The example below illustrates a custom container class that includes the transplant section:

Class Definition

```
Class User.ZMyContainer Extends HS.SDA3.Container
{

Parameter XMLNAME = "Container";

Property ZTransplants As list Of User.ZTransplant;

}
```

3.2.4 Registering your Custom SDA Container in the Configuration Registry

The final step in creating a custom SDA container is to register your custom container in the configuration registry:

1. In the configuration registry, create an entry where:
   - the key is `\CustomSDA3Container`
   - the value is the name of your container class, without the ".cls" suffix

3.3 Extending the SDA with Name/Value Pairs

You can use either of the following methods to extend the SDA:

3.3.1 Creating a Custom SDA Name/Value Pair

Each section of the SDA includes a <CustomPairs> element. To add a name value/pair that captures an additional data item in an existing SDA section, simply submit an SDA document that includes a <CustomPairs> element in the appropriate section. Each name/value pair appears in a <NVPair> element. Enter the description of the data in the <CustomPairs><NVPair><Name> property, and include the data in the <CustomPairs><NVPair><Value> property.

The example below illustrates how to add a set of treatments to an allergy in SDA:

XML

```
<Container>
  ...
   <Allergies>
     <Allergy>
       ...
       <CustomPairs>
         <NVPair>
           <Name>Treatment</Name>
           <Value>Oral Corticosteroids</Value>
         </NVPair>
         <NVPair>
           <Name>Treatment</Name>
           <Value>Injected Steroids</Value>
         </NVPair>
       </CustomPairs>
     </Allergy>
   </Allergies>
 ...
</Container>
```

3.3.2 Creating a Custom SDA Object

The SDA contains a <CustomObjects> section that you can use to store data that is not relevant to any other SDA section. This section comprises one or more <CustomObject> elements.

Each <CustomObject> element *must* include one or more:

- <CustomType> — Identifies the type of entry.
- <CustomPairs> — A list of name/value pairs that contain the data, as described in the previous section.

Each <CustomObject> *may also* include:

- <ActionCode><ActionScope> — Directions to the HealthShare Unified Care Record product on an action to take. See the class reference for specific details.
- <CustomMatchKey> — Directions to the HealthShare Unified Care Record product on how to match entries of this type with objects already in the database. This is typically a concatenation of custom pair entries, for example `<NVPair3>|<NVPair1>`. Each value becomes required for a match, and is evaluated in the order specified.
- <ExternalID> — An identifier that may have meaning to an outside system. It is used as the primary match key, if present.
- <EnteredOn>, <EnteredAt>, <EnteredBy> — Details about the source.
- <FromTime>, <ToTime> — Details regarding when the data is valid.

The class reference for **HS.SDA3.CustomObject** provides additional details.

The example below contains data for two physiotherapy home care visits.

XML

```
<Container>
  ...
  <Patient>
    ...
  </Patient>
  ...
  <CustomObjects>
    <CustomObject>
      <CustomType>HomeCareEvent</CustomType>
      <EnteredOn>2012-02-05T13:00:00Z</EnteredOn>
      <CustomPairs>
        <NVPair>
          <Name>EventType</Name>
          <Value>Physiotherapy</Value>
        </NVPair>
        <NVPair>
          <Name>Comment</Name>
          <Value>Minimal progress.  Assistance required for many tasks.</Value>
        </NVPair>
      </CustomPairs>
    </CustomObject>
    <CustomObject>
      <CustomType>HomeCareEvent</CustomType>
      <EnteredOn>2012-02-01T12:55:00Z</EnteredOn>
      <CustomPairs>
        <NVPair>
          <Name>EventType</Name>
          <Value>Physiotherapy</Value>
        </NVPair>
        <NVPair>
          <Name>Comment</Name>
          <Value>Home evaluation.  Bathroom rails and detachable shower head required.</Value>
        </NVPair>
      </CustomPairs>
    </CustomObject>
  </CustomObjects>
 ...
</Container>
```