



Fundamentals for Accessing Transmittals

<http://www.sedris.org/>

SEDRIIS™ Technology Conference
Lake Buena Vista, FL
06 January 2004

Gregory Hull
SAIC
greg.a.hull@saic.com

Kevin Wertman
SAIC
kevin.m.wertman@saic.com



About This Tutorial

DESCRIPTION

This tutorial covers the fundamentals of accessing SEDRI transmittals through the use of the SEDRI transmittal access C and C++ APIs. Traversal strategies, object management, and API behaviors are discussed applicable to both the C and C++ APIs. Pseudo code and algorithms for performing common actions across a range of applications are provided. The mechanics of the APIs are provided with an introduction to the C functions and data structures, as well as the C++ classes and methods. Coding samples in both languages are provided.

WHO SHOULD ATTEND

Software engineers who intend to design and implement SEDRI-based applications, or those interested in learning the basic functionality of the SEDRI APIs.

PREREQUISITE

A working knowledge of C/C++, or other programming languages is strongly recommended. Prior attendance at either the "Introduction to SEDRI for Managers" or "SEDRI - The Technology Components" tutorial is recommended.

WHAT TO EXPECT

Attendees gain a fundamental understanding of the issues to access SEDRI transmittals. The attendee also gains basic insight into the functionality and mechanics of the SEDRI APIs.



Agenda

SEDRIIS C API Background and
Fundamentals

C API Extraction component

C API Insertion component

Introduction to the SEDRIIS C++ API

Summary



Purpose and Scope

Purpose

- Understand the capabilities of the SEDRIIS API.
- Introduce the key functions and data structures.

Scope

- Highlight API capabilities.
- Demonstrate API functionality through small applications and routines.
- Introduce and explain how to use API functions and data types.

Does NOT Cover

- Issues of good transmittal design or specific strategies of consumption or production.
- Every API call, refer to SEDRIIS Reference Manual and source code header files for complete description.
- EDCS, SRM or DRM API functions.



SEDRIIS C API Background and Fundamentals



Topics

Basic Definitions
API Components
API Conventions
API Data Types
Basic Tools
Reference Material



Basic Definitions

Application Program Interface (API)

- An encapsulation of functionalities common to many applications into reusable modules.

API Implementation

- The instantiation of an API's functionality in software that is bound to a software language.

SEDRIIS Transmittal

- A collection of SEDRIIS objects conforming to the DRM and accessible thru the SEDRIIS API.



SEDRIIS API Components

Transmittal Access API

- C and C++ versions
- Level 0
 - Data Extraction functions provide methods to search and access DRM objects in a SEDRIIS Transmittal.
 - Data Insertion functions provide ability to create or remove DRM objects and relationships.
- Level 1 – Common convenience functions built on top of the Level 0 API functions.

DRM - Provides access to meta-data describing DRM classes, data types and relationships.

EDCS API

SRM API



SEDRIS C API Conventions

Function and data type naming: SE_ followed by function mnemonic, first letter of each word capitalized.

(SE_CreateObject, SE_Object)

SEDRIS constants and enum values are in ALL_CAPS

(SE_DRM_CLS_TRANSMITTAL_ROOT)

Function return status: The return type of most functions is SE_Return_Code which is either SE_RET_CODE_SUCCESS or SE_RET_CODE_FAILURE.

Function returned data:

- Most data returned from the API is done via pointers to an opaque type. (SE_Object)
- All data returned through opaque types is freed using a corresponding API function. (SE_FreeObject)



SEDRIIS C API Data Types

Basic Data Types:

- Promote machine/compiler independence
- Examples: `SE_Integer_Unsigned`, `SE_Long_Float`, `SE_Boolean`, `SE_Short_Integer_Positive`

Opaque Types:

- Promote implementation independence through the hiding of underlying structures
- Commonly referred to as handles
- *Examples: `SE_Transmittal`, `SE_Object`, `SE_Iterator`, `SE_Store`, `SE_Search_Filter`, `SE_Search_Boundary`*



SEDRIIS C API Data Types

SE_Fields is a Tagged Union

- Contains all field information for every DRM object.
- Tag of type SE_DRM_Class specifies the DRM class of the object.
- The tag is used to select the correct union member to access.
- Example for <Transmittal Root>.
 - fields.tag = SE_DRM_CLS_TRANSMITTAL_ROOT
 - fields.u.Transmittal_Root

SE_String

- The characters field stores the string as an array of SE_Characters.
- The length field stores the length of the array.
- SE_Locale gives language and country information.



Basic Tools

Core applications

- Depth
- Syntax Checker, Rules Checker
- Transmittal Browser
- stf_test, itr_test
- Focus
- Model Viewer
- EDCS Query Tool

SEDRIS 3.1 to 4.0 Converter

- Convert STF transmittals created from the SEDRIS version 3.1 DRM and STF Implementation to the 4.0 version.



Reference Material

www.sedris.org/api.htm

email reflectors

- help@sedris.org
- se-coders@sedris.org (primarily for associates)

The Data Dictionary at

www.sedris.org/pro2trpl.htm

- Defines all functions and arguments for the API.
- Defines all data types used by the API.
- Listed by DRM, EDCS, Level 0, Level 1, and SRM.



The Extraction Component of the SEDRIIS C API



Topics

Extraction capabilities

Example 1: Reading the fields of the sample Transmittal's <Description> object

- Illustrates simple traversal methods and the basics of opening a Transmittal and accessing objects.

Example 2: Traversing with Iterators

- Retrieving all <Polygons> in the Transmittal.

Example 3: Retrieving elevation data from a <Property Grid>

- Learn how to extract data from a <Data Table> object.

Advanced features of component Iterators

- Witness the power of the SEDRIIS API.

Miscellaneous Extraction Features



Extraction Capabilities

The 3 Examples will cover the following topics:

- Retrieving the Root Object

- Traversing component relationships

- Retrieving the fields for objects

- Retrieving cells from <Data Tables>

- Selection and filtering of objects via Search Filters and Spatial Boundaries

- Automatic traversal through a Transmittal via branching criteria



Example 1: Reading the <Description>

Read the <Description> from the sample Lake Eola Transmittal

The description is stored in the fields of a SE_DRM_CLS_DESCRIPTION object which the DRM requires to be a component of the <Transmittal Root> object.

To do this we need to know:

- How to open and close a Transmittal for reading.
- How the C API handles status codes and error descriptions.
- About the API's methods of memory management.
 - SE_Store's used to get fields and other object data.
 - Memory allocated by the API through opaque data types.
- How to access objects by traversing relationships.
- How to retrieve fields from an object.



Example 1: Reading the <Description> [2of 3]

Steps to Retrieve the <Description>

- Step 1: Open the Transmittal.
- Step 2: Check the status of step 1.
- Step 3: Get the Root Object.
- Step 4: Get the <Description> as a component of the <Transmittal Root>.
- Step 5: Create an SE_Store to manage the memory for the <Description> object fields.
- Step 6: Get the fields from the object.
- Step 7: Free memory.
- Step 8: Close the Transmittal.



Example 1: Reading the <Description> [3 of 3]

Functions needed to retrieve the
<Description> object

SE_CreateStore()
SE_OpenTransmittalByFile()
SE_GetLastFunctionStatus()
SE_GetRootObject()
SE_GetComponent3()
SE_GetFields()
SE_FreeStore()
SE_FreeObject()
SE_CloseTransmittal()



Example 1: Managing Memory with Stores

A Store is an API data type (SE_Store) created by an application to manage memory allocated by the API.

A Store manages the memory allocated for:

- An description string for the status of the last function called.
- Object's Field data
- Data Table's cell data
- Image's image data

A Store may be reused for better efficiency.

Data managed by a Store is passed back from the API functions in a separate pointer.

Data is valid until the Store is freed or the Store is reused in another API call.

IMPORTANT: Make sure not to reuse the data after the Store has been reused or freed!



Example 1: Creating a Store

```
SE_Return_Code  SE_CreateStore
(
    const char    implementation_identifier[],
    SE_Store *new_store_ptr
);
```

SE_CreateStore() creates a Store for the given API implementation.

The API allocates the SE_Store and so must be freed using SE_FreeStore().

```
SE_Return_Code  SE_FreeStore
(
    SE_Store to_free
);
```



Example 1: Opening the Transmittal

```
SE_Return_Code  SE_OpenTransmittalByFile
(
    const char          file_name[],
    const char          implementation_identifier[],
    SE_Access_Mode      access_mode,
    SE_Transmittal      *transmittal_out_ptr
);
```

The 'root' file of the STF transmittal is passed as the file_name.

Specify "stf" as the implementation_identifier.

Specify access mode of SE_AC_MODE_READ_ONLY.

Assigns the Transmittal handle in the SE_Transmittal.

Returns SE_RET_CODE_SUCCESS or
SE_RET_CODE_FAILURE.



Example 1: Checking the Function's Status

```
SE_Return_Code SE_GetLastFunctionStatus  
(  
    SE_Status_Code    *last_function_status,  
    SE_Store           store_in,  
    char               **status_description  
);
```

Assigns `last_function_status` a status code for the function call last made to the C API.

- `SE_STAT_CODE_SUCCESS` usually set when the return code of the function was `SE_RET_CODE_SUCCESS`.
- Other possible codes depend on the function called. For example `SE_STAT_CODE_TRANSMITTAL_UNACCESSIBLE` may be set for `SE_OpenTransmittal()`.

The `status_description`, if not `NULL`, is managed by the store passed in and will contain extra information about the kind of failure.



Example 1: Getting the Root Object

```
SE_Return_Code  SE_GetRootObject
(
    SE_Transmittal  transmittal_in,
    SE_Object       *root_object_out_ptr
);
```

The DRM specifies that all Transmittals must have a Transmittal Root object which is at the top of the Transmittal's hierarchy.

SE_GetRootObject() returns the root object as an SE_Object for the opened Transmittal passed in to the function.

The SE_Object is allocated by the API, and so must be freed when no longer used.



Example 1: Getting Components

```
SE_Return_Code  SE_GetComponent3  
(  
    SE_Object      object_in,  
    SE_DRM_Class  drm_class,  
    SE_Object      *object_out  
);
```

This is a three-parameter MACRO which calls the API function SE_GetComponent() with default values passed for more advanced options.

Simple to use.

Returns a component of the object passed in.

Can only return the first component of a given type, if more than one exists.



Example 1: Reading Object Fields

```
SE_Return_Code SE_GetFields
(
    SE_Object      object_in,
    SE_Store       store_in,
    SE_FIELDS_PTR  *fields_out_ptr
);
```

Field data is returned in a fields_out_ptr, which is allocated and managed by store_in.

The fields_out_ptr is valid until store_in is either freed or reused.



Example 1: Freeing Objects

```
SE_Return_Code SE_FreeObject  
(  
    SE_Object  old_object  
);
```

Called to free memory resources associated with SE_Objects returned by the API.

Does NOT remove the object from the Transmittal.

Needs to be called for every SE_Object that has been returned.

- Even if it is the same object retrieved previously.



Example 1: Closing Transmittals

```
SE_Return_Code  SE_CloseTransmittal  
(  
    SE_Transmittal transmittal  
);
```

SE_CloseTransmittal() frees memory associated with the opened Transmittal, and closes it.

Only pass in Transmittals that have been retrieved with the SE_OpenTransmittal() calls.



Example 1: Example Code

```
main() {
SE_Ret_Code  ret;
SE_Transmittal  xmittal;
SE_Object      root_obj, desc_obj;
SE_Store       store;
SE_FIELDS_PTR  desc_flds;

ret = SE_CreateStore( "stf", &store );
ret = SE_OpenTransmittalByFile( "Lake_Eola.stf", "stf",
                               SE_AC_MODE_READ_ONLY, &xmittal );
ret = SE_GetRootObject( xmittal, &root_obj );

ret = SE_GetComponent3( root_obj, SE_DRM_CLS_DESCRIPTION, &desc_obj );

ret = SE_GetFields( desc_obj, store, &desc_flds );

printf( "Transmittal Description %s\n", desc_flds->u.Description.abstract.characters );

ret = SE_FreeStore( store ); // this must be done after the printf above!
ret = SE_FreeObject( root_obj );
ret = SE_FreeObject( desc_obj );

ret = SE_CloseTransmittal( xmittal );
}
```



Example 2: Traversing with Iterators

Retrieve all of the <Polygons> in the sample Lake Eola Transmittal

We could do this by iteratively retrieving components from the <Transmittal Root> on down.

- This would be VERY tedious and inefficient.

This example will introduce:

- Iterators
- How to create and use Iterators
- How to define search rules, and create search filters to pass to our Iterator



Example 2: Traversing with Iterators

Steps to Retrieve <Polygons>

- Step 1: Open the transmittal and get the Root Object.
- Step 2: Specify SE_Search_Rule for matching
<Polygon> objects.
- Step 3: Use the Search Rules to create a SE_Search_Filter.
- Step 4: Use the Search Filter and Root Object to create a
component Iterator.
- Step 5: Call SE_GetNextObject() to traverse through matching
objects (<Polygons>).
- Step 6: ... process the <Polygons> ...
- Step 7: Free the memory for the object, Search Filter and Iterator.
- Step 8: Close the transmittal.



Example 2: Traversing with Iterators

New functions needed to retrieve
<Polygons>

SE_CreateSearchFilter()
SE_InitializeComponentIterator()
SE_IsIteratorEmpty()
SE_GetNextObject()
SE_FreeIterator()
SE_FreeSearchFilter()



Example 2: About Iterators

An Iterator provides sequential access to a set of objects matching a given criteria.

Iterators are represented by the opaque data type `SE_Iterator`. Since these are allocated by the API, they must be freed with the API call `SE_FreeIterator`.

The API function `SE_IsIteratorEmpty()` provides a way to determine if there are any more objects in the iterator.

The API function `SE_GetNextObject()` provides a way to step through the set.

There are 3 types of Iterators: component, associate and aggregate, which traverse the 3 different types of relationships.

More advanced features provide extensive control over selection process and objects returned via parameters to:

- Simplify access to more complex structures
- Apply conversions to object field data
- Control traversal sequence & branching



Example 2: Specifying Search Rules

Boolean expressions built using operators SE_AND, SE_OR and SE_NOT, and the following set of matching criteria:

- Object type match. (Abstract or concrete classes are valid.)
- Exact field value match and field value range match.
- Hierarchy depth. Matches objects within a given number of levels from the base object.
- Component object type match.
- Component exact field value or field value range match.
- Predicates: application functions “called back” by the user to evaluate the object.

Implemented as arrays which can be initialized with convenient C Macros.

```
SE_Search_Rule polygon_rules[] =
{
    SE_AND
    (
        SE_DRM_CLASS_MATCH( POLYGON ),
        SE_MAX_SEARCH_DEPTH( 5 )
    )
    SE_END
};
```



Example 2: Creating Search Filters

```
SE_Return_Code  SE_CreateSearchFilter
(
    SE_Transmittal    transmittal,
    const SE_Search_Rule  rules[],
    SE_Search_Filter *search_filter_out_ptr
);
```

Search Filters can be reused by more than one Iterator.
They are allocated by the API and must be freed using
`SE_FreeSearchFilter()`.
May be freed before the Iterator is freed.

```
SE_Return_Code  SE_FreeSearchFilter
(
    SE_Search_Filter  search_filter
);
```



Example 2: Component Iterators

```
SE_Return_Code  SE_InitializeComponentIterator3
(
    SE_Object      start_object,
    SE_Search_Filter  filter,
    SE_Iterator     *Iterator_out_ptr
);
```

`SE_InitializeComponentIterator3()` creates a component `SE_Iterator`.

The iterator will traverse down from `start_object` and find all objects matching the criteria stored in the search filter.

This is a C language macro that resolves to `SE_InitializeComponentIterator()` with default parameters passed for more advanced functionality of iterators.

`SE_Iterator` is an opaque type that is allocated by the API, and so must be freed with `SE_FreeIterator()`.

```
SE_Return_Code  SE_FreeIterator
(
    SE_Iterator  Iterator
);
```



Example 2: Retrieving Objects from Iterators

```
SE_Return_Code SE_GetNextObject
(
    SE_Iterator  iterator,
    SE_Object    *next_object,
    SE_Object    *link_object
);
```

Objects can be sequentially retrieved from Iterators using the API function `SE_GetNextObject()`.

`SE_Objects` are allocated by the API and must be freed.

If the component relationship has a link object, the object is returned in the third argument.

`SE_IsIteratorEmpty()` returns `SE_TRUE` if there are no objects to be returned by the iterator.

```
SE_Boolean SE_IsIteratorEmpty
(
    SE_Iterator  iterator
);
```



Example 2: Example Code

```
void FindPolygons( SE_Transmittal xmittal, SE_Object root_obj )
{
    SE_Return_Code      ret;
    SE_Search_Rule      polygon_search_rules[] =
    {
        SE_DRM_CLASS_MATCH( POLYGON ) /* default to Infinite depth */
    }
    SE_Search_Filter      search_filter;
    SE_Iterator           iterator;
    SE_Object             polygon_obj;

    ret = SE_CreateSearchFilter( xmittal, polygon_search_rules, &search_filter );

    ret = SE_InitializeComponentIterator3( root_obj, search_filter, &iterator );

    while( SE_IsIteratorEmpty ( iterator) == SE_FALSE )
    {
        ret = SE_GetNextObject( iterator, &polygon_obj, NULL ;
        /* process the polygon object */

        ret = SE_FreeObject( polygon_obj );
    }
    ret = SE_FreeIterator( iterator );
    ret = SE_FreeSearchFilter( search_filter );
}
```



Example 3: Retrieving Gridded Elevation Data

We want to retrieve elevation data from a <Property Grid>

In the DRM, a <Property Grid> is a sub class of the abstract class <Data Table>.

To retrieve data from a <Data Table> we need to know:

- What is a <Data Table's> Signature and Extents.
- What functions exist for retrieving data from a <Data Table>.
- What the format of retrieved data can be.



Example 3: Retrieving Gridded Elevation Data

Steps to retrieve gridded elevation data

Step 1: Open the Transmittal and get the Root Object.

Step 2: Get the <Property Grid>.

Step 3: Create a Store.

Step 4: Get the <Property Grid> extents.

Step 5: Get the <Property Grid> signature.

Step 6: Get the <Property Grid> data.

Step 7: Process the data.

Step 8: Free the memory associated with the extents and signature.

Step 9: Free the <Property Grid>, Store, and Root Object.

Step 10: Close the Transmittal.



Example 3: Retrieving Gridded Elevation Data

New functions needed to
retrieve elevation data

SE_GetDataTableSubExtent () (level 1)

SE_GetDataTableSignature() (level 1)

SE_GetDataTableData()

SE_FreeDataTableSubExtent() (level 1)

SE_FreeDataTableSignature() (level 1)



Example 3: <Data Table> Signatures

A <Data Table> consists of Cells.

Cells consist of 1 or more elements (types of data).

The <Data Table>'s signature determines the elements for its cells.

The signature is the set of ordered <Table Property Description> components under the <Data Table> object.

EDCS codes in the <Table Property Description> fields give the meaning of an element.

A Level 1 API function exists as a convenience to get a <Data Table>'s signature.



Example 3: <Data Table> Signatures

```
SE_Status_Code  SE_GetDataTableSignature
(
    SE_Object                data_table,
    SE_Integer_Unsigned      *element_count_out_ptr,
    SE_Table_Property_Description_Fields **prop_array_out_ptr,
    SE_Integer_Positive      **tbl_prop_descr_ptr
);
```

Returns the number of elements for the data_table.

Allocates an array of indices for the ordered <Table Property Description>'s that are components of data_table.

Optionally creates an array of the fields for these <Table Property Description>s.

These lists should be freed with the Level 1 function SE_FreeDataTableSignature().

```
void  SE_FreeDataTableSignature
(
    SE_Table_Property_Description_Fields *prop_array_ptr,
    SE_Integer_Positive                  *tbl_prop_descr_ptr
);
```



Example 3: The <Data Table> Extents

<Data Table>s may have 1 or more spatial or non-spatial dimensions.

The dimensional extents of <Data Table> is determined by its ordered <Axis> components.

A Level 1 API function exists to get a <Data Table>'s extents.

```
SE_Status_Code  SE_GetDataTableSubExtent
{
    SE_Object          data_table,
    SE_Data_Table_Sub_Extent *extents_ptr,
    SE_Integer_Unsigned *cell_count_ptr
};
```

The number of cells in the <Data Table> is returned.

SE_Data_Table_Sub_Extent stores the number of <Axis> dimensions and the starting and stopping value for each <Axis>.

The extents should be freed with the Level 1 function below:

```
SE_Status_Code  SE_FreeDataTableSubExtent
{
    SE_Data_Table_Sub_Extent *extents_ptr
};
```



Example 3: <Data Table> Conceptual Layout

Illustration of small data table with 4 cells
and 2 elements per cell.

The types of the elements are
EDCS_Long_Float and EDCS_Integer which
are given in the 2 Table Property Descriptions
There is 1 Axis of size 4.

	<i>Cell 0</i>	<i>Cell 1</i>	<i>Cell 2</i>	<i>Cell 3</i>
<i>element 1</i>	456.65	433.15	388.61	453.09
<i>element 2</i>	1024	998	110	879



Example 3: <Data Table> Extraction Functions

```
SE_Return_Code SE_GetDataTableData  
(  
    SE_Object                data_table,  
    const SE_Data_Table_Sub_Extent *extents_ptr,  
    SE_Integer_Positive       element_count,  
    const SE_Integer_Positive  tbl_prop_descr_num[],  
    SE_Store                  store_in,  
    SE_Data_Table_Data        **dt_data_p  
);
```

Returns data as an array of SE_Data_Table_Data structs which is managed by the store. This data struct contains a union with pointers (arrays) for each data type that a data table may hold.

The full extents of the Data Table may be retrieved by setting extents_ptr to the value from SE_GetDataTableSubExtent. Or the user may specify any arbitrary sub-extent of the data table.

All elements in a Data Table may be retrieved by setting element_count and the tbl_prop_descr_num array to values set from SE_GetDataTableSignature. Or any subset of elements from the list of elements may be retrieved.

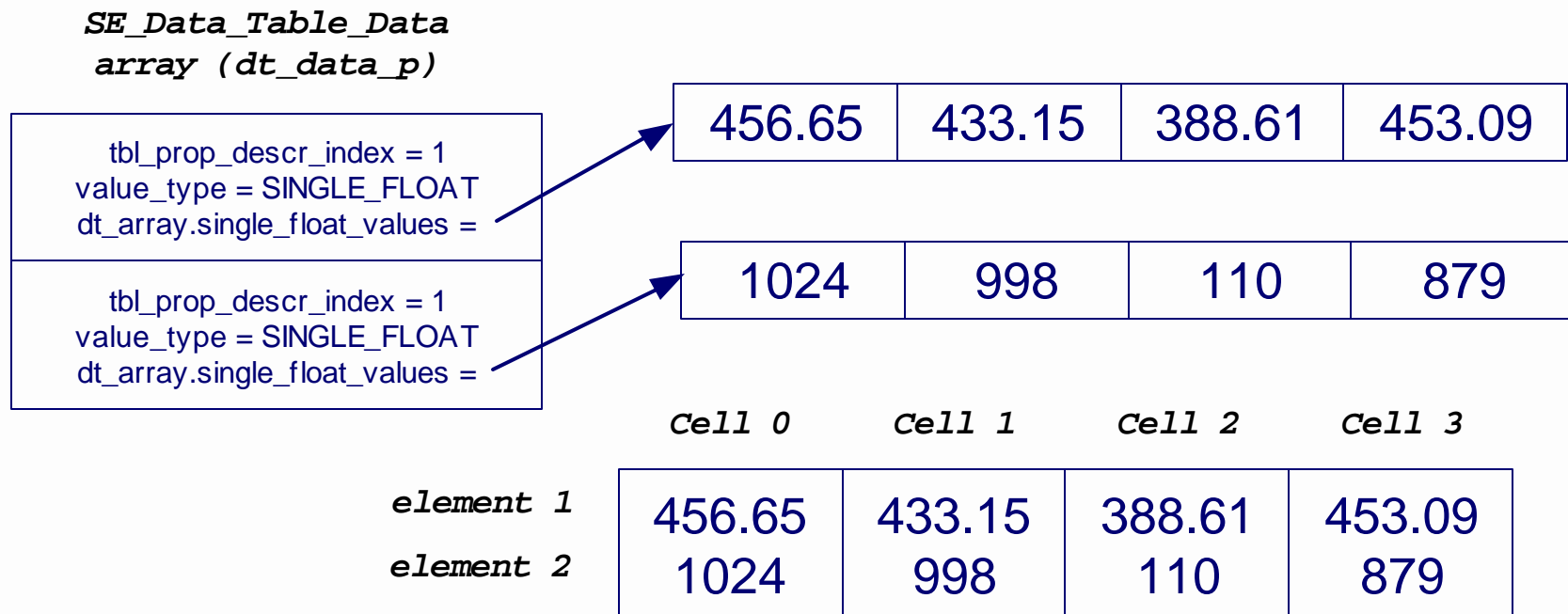


Example 3: <Data Table> Cell Data Memory Organization

Illustration of how retrieved data table cell data is accessed.

To access 1st element 4th cell (value 453.09):

`dt_data_p[0].dt_array.single_float_values[3];`





Example 3: Example Code

```
void ExtractDataTableCells( SE_Object data_table, SE_Store store )
{
    SE_Data_Table_Sub_Extent      extents;
    SE_Integer_Unsigned           cell_count, element_count;
    SE_Integer_Positive           *tbl_prop_descr_num;
    SE_Table_Property_Description_Fields *prop_descr_flds;
    SE_Data_Table_Data            *cell_data;

    status = SE_GetDataTableSubExtent( data_table, &extents, &cell_count );

    status = SE_GetDataTableSignature( data_table, &element_count,
                                       prop_descr_flds, tbl_prop_descr_num );

    ret = SE_GetDataTableData ( data_table, &extents, element_count,
                               tbl_prop_descr_num, store, &cell_data );

    /* process the cell data */

    SE_FreeDataTableSubExtent( &extents );
    SE_FreeDataTableSignature( prop_descr_flds, tbl_prop_descr_num );
    /* cell_data is freed when the Store is freed */
}
```



Advanced Component Iterator Features

```
SE_Return_Code SE_InitializeComponentIterator(  
    SE_Object                start_object,  
    SE_Search_Boundary       boundary,  
    SE_Search_Filter          filter,  
    SE_Boolean               directly_attach_table_components,  
    SE_Boolean               process_inheritance,  
    SE_Boolean               transform_locations,  
    SE_Boolean               follow_model_instances,  
    SE_Boolean               evaluate_static_control_links,  
    const SE_Hierarchy_Select_Parameters *select_parameters_ptr,  
    const SE_Hierarchy_Order_Parameters *traversal_order_parameters_ptr,  
    SE_Traversal_Order        general_traversal_pattern,  
    SE_ITR_Behaviour          itr_traversal,  
    SE_Iterator               *iterator_out_ptr);
```

This is the complete definition of
`SE_InitializeComponentIterator()`.

Ten parameters we have not discussed before.



Advanced Component Iterator Features

Component Iterators contain extensive features to:

Filter objects

- Search Rules
- Search Boundaries
- Inter-Transmittal Referencing (ITR)

Provide control over sequence and branching of traversal

- Traversal method
- Hierarchy selection

Add objects or field data to the Iterator based on DRM
information

- Directly attach table components
- Process inheritance
- Transform locations
- Follow model instances
- Evaluate static <Control Links>



Advanced Component Iterator Features: Filtering

Objects can be included or excluded based on:

- Search Rules:
 - Selection determined from user defined rules based on objects' fields or relationships. Search Rules are used to create Search Filters.
- Search Boundary:
 - Selection is based on whether the spatial location or extent of an object falls within a user defined search bounds.
 - The search bounds is given by the C API data struct SE_Search_Bounds.
 - The search boundary information is given with the creation of opaque C API type SE_Search_Boundary using SE_CreateSpatialSearchBoundary()
- Inter-Transmittal Referencing (ITR):
 - An ITR reference is a component or associate relationship from an object in one Transmittal to an object in another. The DRM is not ITR aware. All normal rules for relationships apply.
 - Iterators may be configured include objects in others transmittals or to ignore them. If including, the iterator can either resolve (access) the ITR object or it may simply return an 'unresolved' SE_Object. The Status Code set by SE_GetNextObject() will give an indication when an ITR object is returned and whether it was resolved.



Advanced Component Iterator Features: Sequencing and Branching Control

Objects can be ordered according to a Depth-first or a Breadth-first Traversal.

Hierarchy order parameters can be set to affect the order in which components of hierarchy objects are returned.

Hierarchy selection parameters can be set to choose the set of components that are returned.



Advanced Component Iterator Features: DRM Related Features

Directly attach table components:

- Indexed components are replaced with directly attached instances.
- Examples: <Attribute Set Index> objects, <Color Index> , ...

Inherited components:

- The DRM specifies rules to allow objects to inherit components of objects found higher in the aggregation tree. Inherited components are semantically the same as directly attached components.
- Component Iterators can be configured to return inherited components as real objects.

Follow model instances:

- When enabled, this feature causes an Iterator to return the <Model> associated with a model instance as the model instance's component.



Advanced Component Iterator Features: DRM Related Features

Transform locations:

- For <Models>, coordinates are transformed from the <Model's> SRF to the World SRF.

Evaluate static <Control Link>:

- <Control Link> objects in the DRM provide a mechanism for specifying how field values of objects can be modified.
- Used to implement instance-specific values for object fields.
- The component iterator can process static <Control Link> expressions and return fields containing the modified values.



Miscellaneous Extraction Features and functions

Object ID Strings

- SE_GetObjectFromIDString(), SE_GetObjectIDString().
- Every object in a transmittal has an ID String unique to the transmittal so that they may be extracted directly without having to traverse from the root object.

Associate and aggregate Iterators

- SE_InitializeAssociateIterator(), SE_InitializeAggregateIterator()
- Similar to component iterators, but with only search filters and ITR as options and limited to a 'depth' of 1.

Extracting a hierarchy of objects

- SE_GetPackedHierarchy()
- Extracts an entire hierarchy of the transmittal at one time.

Extracting <Image> data

- SE_GetRearrangedImageData(), SE_GetImageData()



Miscellaneous Extraction Features and Functions

Extracting <Image> data:

- The API has a specialized function to extract <Image> data.
- Organization and word order of returned data is determined by the <Image> object's field values.
- Only one mip level can be extracted at a time.
- The start and stop texel parameters must correspond to the start and stop texels for the Image.
- Stores are used to manage the memory for the <Image> data.
- Level 1 functions exist to convert <Images> between differing representations (color model, scan direction, pixel depth, etc.).

```
SE_Return_Code SE_GetImageData(  
    SE_Object          image,  
    SE_Integer_Unsigned start_texel_horizontal,  
    SE_Integer_Unsigned start_texel_vertical,  
    SE_Integer_Unsigned start_texel_z,  
    SE_Integer_Unsigned stop_texel_horizontal,  
    SE_Integer_Unsigned stop_texel_vertical,  
    SE_Integer_Unsigned stop_texel_z,  
    SE_Short_Integer_Unsigned mip_level,  
    SE_Store           store_in,  
    unsigned char      **data_out_ptr);
```



The Insertion Component of the SEDRIIS C API



Topics

Insertion Capabilities

Example 4: Build a geometry representation of the Lake Eola transmittal.

Example 5: Build a feature representation of the Lake Eola transmittal with elevation data.

Example 6: Edit the feature representation by editing the <Description> fields, removing some objects and adding other objects.

Example 7: Combine the feature representation and geometry representations of Lake Eola by using ITR.

Miscellaneous Insertion Topics



Insertion Capabilities

Creating Transmittals by adding objects and relationships.

Adding cell data to <Data Tables>.

Adding objects and relationships, to existing Transmittals.

Editing the fields of objects in existing Transmittals.

Removing objects and relationships from existing Transmittals.

Publishing objects for use in ITR references.

Make ITR references from one Transmittal to another.



Example 4: Geometry Representation

We want to create a geometry representation of Lake Eola

To do this we need to know:

- How to open a Transmittal for creation.
- How to create SEDRIIS objects.
- How to set the fields of an object.
- How to create relationships between objects.
- How to set the Root Object in the Transmittal.



Example 4: Geometry Representation

Steps to Create a Transmittal

- Step 1: Open the Transmittal.
- Step 2: Create a <Transmittal Root> and set its fields.
- Step 3: Add the <Transmittal Root> to the transmittal.
- Step 4: Set the <Transmittal Root> as the root object.
- Step 5: Create a new object.
- Step 6: Set field data for new object - Optional.
- Step 7: Create a component relationship between the new object and its parent.
- Step 8: Free the new object.
- Step 9: Repeat Steps 5 - 8 until done.
- Step 10: Free the Root Object.
- Step 11: Close the Transmittal.



Example 4: Geometry Representation

Functions needed to create Lake Eola Transmittal using geometry

SE_OpenTranmittalByFile()

SE_CreateObject()

SE_SetFieldsToDefaults()

SE_PutFields()

SE_AddComponentRelationship()

SE_SetRootObject()

SE_FreeObject()

SE_CloseTransmittal()



Example 4: Creating a New Transmittal

```
SE_Return_Code  SE_OpenTransmittalByFile
(
    const char          file_location[],
    const char          implementation_identifier[],
    SE_Access_Mode      access_mode,
    SE_Transmittal      *transmittal_out_ptr
);
```

With access_mode of SE_AC_MODE_CREATE, an SE_Transmittal is created and passed back in transmittal_out_ptr.

Implementation_identifier is the implementation of the API; usually “stf”.



Example 4: Creating New Objects

```
SE_Return_Code  SE_CreateObject
(
    SE_Transmittal  transmittal,
    SE_DRM_Class    drm_class,
    SE_Object       *new_object_out_ptr
);
```

Creates a SE_Object in memory.

The “drm_class” parameter specifies the object’s DRM class.

Transmittal is the Transmittal to which this object will belong.

The created object is allocated by the API and must be freed by calling SE_FreeObject().



Example 4: Setting Field Values

```
SE_Return_Code SE_PutFields  
(  
    SE_Object      object,  
    SE_FIELDS_PTR  new_fields  
);
```

Sets field values of an SE_Object.

The `drm_class` member of the fields structure must match the class of the object.

The object is created with fields that are initialized to the default for its class. If the defaults are desired, then `SE_PutFields()` is not necessary.



Example 4: Creating Object Relationships

```
SE_Return_Code  SE_AddComponentRelationship  
(  
    SE_Object aggregate_object,  
    SE_Object component_object,  
    SE_Object link_object  
);
```

Establishes the relationship between an aggregate and its component.

If the DRM specifies a 2-way relationship, then the component to aggregate relationship is made too.

Supports specification of link objects (use NULL if no link object is required).



Example 4: The Root Object

```
SE_Return_Code  SE_SetRootObject  
(  
    SE_Transmittal transmittal,  
    SE_Object  new_root_object,  
    SE_Object *old_root_object  
);
```

This sets the object that will be returned from a call to `SE_GetRootObject()`.

The Root Object must be a Transmittal Root Object.

If the root object was set previously, then the old root object is returned in `old_root_object`.

NULL may be passed in for `old_root_object`.



Example 4: Example Code

```
main()
{
    SE_Return_Code ret;
    SE_Transmittal  xmittal;
    SE_Object       root_obj, description_obj;
    SE_Fields       root_flds, description_flds;

    ret = SE_OpenTransmittalFile( "Lake_Eola.stf", "stf",
                                   SE_AC_MODE_CREATE, &xmittal );

    ret = SE_CreateObject( xmittal, SE_DRM_CLS_TRANSMITTAL_ROOT,
                           &root_obj );
    ret = SE_SetRootObject( xmittal, root_obj, NULL);

    ret = SE_CreateObject( xmittal, SE_DRM_CLS_DESCRIPTION,
                           &description_obj );
    ret = SE_SetFieldsToDefault( SE_DRM_CLS_DESCRIPTION, &description_flds );
}
```



Example 4: Example Code (continued)

```
strcpy( description_flds.u.Description.abstract.characters,  
                                               "Lake Eola, Geometry" );  
description_flds.u.Description.abstract.length =  
                                               strlen("Lake Eola, Geometry " );  
  
ret = SE_PutFields( description_obj, &description_flds );  
  
ret = SE_AddComponentRelationship( root_obj, description_obj, NULL );  
  
/* create the Geometry Objects under the Environment Root. */  
create_environment_root_objects( xmittal, root_obj );  
  
ret = SE_FreeObject( root_obj );  
ret = SE_FreeObject( description_obj );  
  
ret = SE_CloseTransmittal( xmittal );  
}
```



Example 5: Feature Representation and Elevation

We want to create a feature representation of Lake Eola complete with elevation data

A feature representation simply uses different DRM objects, such as Point_Feature and Areal_Feature to represent trees, lakes, fountains, footpaths etc.

In this Transmittal, elevation data is stored in a <Property Grid>, which is a <Data Table>.

To do this we need to know:

- What we previously learned about insertion.
- How to create and add cells to a <Data Table>.



Example 5: Feature Representation and Elevation

Steps to create our feature representation of Lake Eola

- Step 1: Open the Transmittal and create the Root Object.
- Step 2: Use the same functions introduced in the geometry example to create the feature objects.
- Step 3: Create the <Property Grid> object and all its <Axis> and <Table Property Description> objects.
- Step 4: Add the elevation data to the <Property Grid>.



Example 5: Feature Representation and Elevation [3 of 3]

**New functions needed to create our
feature representation of Lake Eola**

SE_AllocDataTableData()
SE_PutDataTableData()
SE_GetCellCountForSubExtent()



Example 5: Inserting data into the <Data Table>

Data is organized the same as for SE_GetDataTableData(); as an array of SE_Data_Table_Data structures which each point to an array of data for the element.

As a convenience for allocating these arrays, the function SE_AllocDataTableData is provided.

The dt_data_ptr is set with the memory managed by the Store passed in.

```
SE_Return_Code SE_AllocDataTable
(
    SE_Object                data_table,
    SE_Data_Table_Sub_Extent *sub_extent,
    SE_Integer_Unsigned      element_count,
    const SE_Integer_Positive tbl_prop_descr_num[],
    SE_Store                 *store,
    const SE_Data_Table_Data **dt_data_ptr
);
```



Example 5: Inserting data into the <Data Table>

All the Axis and Table Property Description objects must be created as components of the data table before the cell data is added.

Once the array's of data in SE_Data_Table_Data have been set with the element data to be put into the data table, the function SE_PutDataTableData is called.

```
SE_Return_Code SE_PutDataTableData
(
    SE_Object          data_table,
    SE_Data_Table_Sub_Extent *sub_extent,
    SE_Integer_Unsigned element_count,
    const SE_Integer_Positive tbl_prop_descr_num[],
    const SE_Data_Table_Data *dt_data_in_ptr
);
```



Example 5: Code to insert data into the <Data Table>

```
void AddDataTableCells( SE_Object data_table_obj,
                        SE_Data_Table_Sub_Extent *sub_extents,
                        SE_Store store,
                        EDCS_Long_Float *elev_data
{
    SE_Return_Code      ret;
    SE_Integer_Unsigned elev_tpd_indx=1; // data table has only 1 element for elevation
    SE_Data_Table_Data *dt_data;

    ret = SE_AllocDataTableData( data_table_obj, sub_extents,
                                1, // 1 element to allocate
                                &elev_tpd_indx, // element is 1st Table Prop Description
                                store, &dt_data );
    int cell_cnt = SE_GetCellCountForSubExtent( sub_extents );

    for( int i=0 ; i < cell_cnt; i++ )
    {
        dt_data[0].value_type = SE_DTD_VT_SINGLE_FLOAT;
        gridData[0].u.single_float_values[i] = elev_data[i];
    }

    ret = SE_PutDataTableData( data_table_obj, sub_extents,
                              1, &elev_tpd_indx, dt_data );
    // dt_data is freed when the store is freed or reused.
}
```



Example 6: Editing Transmittals

We want to edit our feature Transmittal. We want to change the <Description>, and Replace a Tree with a Building.

Our building will be a <Feature Model Instance> (FMI) of a <Feature Model> that is already in our <Model Library>.

To do this we need to know:

- How to open a Transmittal for update.
- How to change the fields of an existing object.
- How to delete relationships.
- How to remove objects from a Transmittal.
- How to add new objects to an existing Transmittal.
- How to create associate relationships.



Example 6: Editing Transmittals [2 of 3]

Steps to editing our feature Transmittal

- Step 1: Open the Transmittal in update mode.
- Step 2: Get the <Description> object.
- Step 3: Put the new fields in the same manner as example 1.
- Step 4: Get the <Point Feature> representing the tree we are chopping down.
- Step 5: Remove the component relationship between the <Point Feature> and its parent.
- Step 6: Remove the <Point Feature>.
- Step 7: Create a <FMI> object to represent the building we are putting up.
- Step 8: Add the <FMI> to the Transmittal.
- Step 9: Create an association relationship from the <FMI> to the <Feature Model>.



Example 6: Editing Transmittals [3 of 3]

Functions needed to edit our feature Transmittal

SE_OpenTransmittalByFile()

SE_PutFields()

SE_RemoveFromTransmittal()

SE_RemoveComponentRelationship()

SE_RemoveObjectTree()

SE_CreateObject()

SE_AddAssociateRelationship()



Example 6: Editing Object Fields

Pass `SE_AC_MODE_UPDATE` as the access mode to `SE_OpenTransmittalByFile()`.

Using functions from the Extraction API, obtain the `SE_Object` for the <Description>.

Set new field values in a `SE_Fields` type, and call `SE_PutFields` just the same as in create mode.



Example 6: Removing Relationships

```
SE_Return_Code  SE_RemoveComponentRelationship
(
    SE_Object    aggregate_object,
    SE_Object    component_object,
    SE_Object    link_object
);
```

Obtain the aggregate and component SE_Objects with the relationship to break.

Call SE_RemoveComponentRelationship().

- Note: SE_RemoveAssociateRelationship() exists for removing associate relationships.

This does NOT remove objects from the Transmittal. If this is the only relationship to the component object and it is not removed from the transmittal, then the object will become an orphaned object that exists in the transmittal with no way to access it.



Example 6: Removing Objects

```
SE_Return_Code  SE_RemoveFromTransmittal  
(  
    SE_Object      old_object,  
    SE_Transmittal transmittal  
);
```

Obtain the SE_Object to remove.

Call SE_RemoveFromTransmittal().

This does NOT remove existing relationships to the object.

- Be careful not to leave dangling objects or references.
- If we want to remove the whole sub-tree below the object, then instead call the level 1 function SE_RemoveObjectTree().

```
SE_Status_Code  SE_RemoveObjectTree( SE_Object start_object );
```



Example 6: Adding New Objects

Pass SE_AC_MODE_UPDATE as the access mode to SE_OpenTransmittalByFile().

Using functions from the Extraction API, obtain the SE_Object that will be the parent of the new <FMI> object.

Call SE_CreateObject() to create the FMI just the same as in Create mode.

Set field values of the <FMI>, and call SE_PutFields just the same as in Create mode.

Call SE_AddComponentRelationship() from the <FMI's> parent to the <FMI>.



Example 6: Adding Associate Relationships

```
SE_Return_Code  SE_AddAssociateRelationship
(
    SE_Object    from_object,
    SE_Object    to_object,
    SE_Object    link_object,
    SE_Boolean   make_two_way
);
```

Obtain <Feature Model> of the building in the model library.

Call SE_AddAssociateRelationship().

Pass NULL for the link object, since there is no link object for a <FMI> => <Feature Model> associate relationship.

Pass SE_FALSE for make_two_way, since this will only be a one-way associate relationship.



Example 6: Example Code

```
void ChopDownTree( SE_Transmittal xmittle, SE_Object tree_parent_obj,
                  SE_Object tree_obj, SE_Object bldg_model_obj )
{
    SE_Return_Code ret;
    SE_Object      bldg_fmi_obj;

    ret = SE_RemoveComponentRelationship( tree_parent_obj, tree_obj, NULL );

    ret = SE_RemoveFromTransmittal( tree_obj, xmittle );

    // We could replace the previous 2 lines of code with 1 call to SE_RemoveObjectTree( tree_obj );
    //
    ret = SE_CreateObject( xmittle, SE_DRM_CLS_FEATURE_MODEL_INSTANCE,
                          &bldg_fmi_obj );

    ret = SE_AddComponentRelationship( tree_parent_obj, bldg_fmi_obj, NULL );

    ret = SE_AddAssociateRelationship( bldg_fmi_obj, bldg_model_obj, NULL, SE_FALSE );
}
```



Example 7: Using ITR

We want the geometry representation to include the elevation data from the feature representation

There are 2 ways of creating an ITR reference.

- The referenced object is accessible.
- The referenced object is NOT accessible.

We will need to know how to give a transmittal a URN name and how to publish objects in the transmittal.

If the reference transmittal is not accessible, then we will need to know:

- How to create an unresolved object used to create a component relationship.



Example 7: Creating an ITR Reference

Steps to create an ITR reference if the referenced object is accessible

- Step1 : Open the geometry representation Transmittal for update.
- Step 2: Open the feature representation Transmittal for update.
- Step 3: Give the feature Transmittal a URN Name.
- Step 4: Create and add a <Property Grid Hook Point> (PGHP) to the geometry representation.
- Step 5: Use an Iterator to find the <Property Grid> (PG) in the feature representation.
- Step 6: Publish the PG object.
- Step 7: Call SE_AddComponentRelationship() with the <PGHP> and the <PG> objects to make the relationship just as if the objects were in the same Transmittal. We don't need to specify ITR; it is automatic.



Example 5: Creating an ITR Reference

**New functions needed to make our
<Property Grid> available to other
transmittals**

SE_SetTransmittalName()

SE_PublishObject()

SE_GetUnresolvedObjectFromPublishedLabel()



Example 7: Publishing Objects

```
SE_Return_Code  SE_SetTransmittalName
(
    SE_Transmittal    transmittal,
    const char *      transmittal_name
);
```

In order for one Transmittal to be referenced from another, it must have a URN name..

SE_SetTransmittalName() gives an open Transmittal a valid URN name.

- Example URN: urn:x-sedris:saic:LakeEola.stf:1.

In order to reference an object from a different Transmittal, an object must be published with a label.

- The function SE_PublishObject() does this.

```
SE_Return_Code  SE_PublishObject
(
    SE_Object      object_in,
    const char *   *label_in
);
```



Example 7: Creating an ITR Reference

Steps to create an ITR reference if the referenced object is NOT accessible.

Step 1: Open the geometry representation Transmittal for update.

Step 2: Create and add a <PGHP> to the geometry representation.

Step 3: Call SE_GetUnresolvedObjectFromPublishedLabel() to create an unresolved SE_Object with the <PG's> object label and the feature representation Transmittal's URN name.

Step 4: Call SE_AddComponentRelationship() with the <PGHP> and the <PG> SE_Objects.

Note: The feature representation at some point must still be assigned the URN and have the PG published with the same label.

```
SE_Return_Code  SE_GetUnresolvedObjectFromPublishedLabel
(
    const char      *transmittal_name,
    const char      *object_label,
    const char      implementation_identifier[],
    SE_Object *object_out_ptr
);
```



Miscellaneous Insertion Topics: Inserting Image Data

```
SE_Return_Code SE_PutImageData  
(  
    SE_Object                image,  
    SE_Integer_Unsigned      start_texel_horizontal,  
    SE_Integer_Unsigned      start_texel_vertical,  
    SE_Integer_Unsigned      start_texel_z,  
    SE_Integer_Unsigned      stop_texel_horizontal,  
    SE_Integer_Unsigned      stop_texel_vertical,  
    SE_Integer_Unsigned      stop_texel_z,  
    SE_Short_Integer_Unsigned mip_level,  
    SE_Integer_Unsigned      byte_count,  
    const unsigned char      *data_in_ptr  
);
```

The <Image> object must have first been created and committed to the transmittal.

Memory for data_in_ptr is managed by the user application.

Word order of the <Image> data is determined by the image object's field values.



Introduction to the SEDRIIS C++ API



Topics

C++ API History

C++ API Goals

C++ API Overview

C++ API Benefits

C++ API Classes

Code comparisons with C API

- Data extraction example
- Data insertion example



C++ API History

A C++ prototype was shown at STC 2002 in Vancouver.

- Prototype generated a lot of interest.
- In parallel, existing programs using SEDRIIS requested an object-oriented API.

Requirements gathering and planning started in late 2002 and continued through 2003.

A working prototype based on the current SEDRIIS 3.1.2 SDK was used to finish fleshing out the C++ API and used for internal testing in order make sure we got it right.

Completed C++ API will be released as part of the SEDRIIS 4.0 SDK package.



C++ API Goals

To meet current and future customer demand for object-oriented (OO) implementations.

To expose the object-oriented aspects of the DRM at the interface level. For example, using C++ class hierarchies for DRM objects.

To balance the use of OO facilities vs. providing consistency with the current interfaces and capabilities.

To minimize the economic impact (maintainability, development) of multiple baseline implementations.

To provide a portable interface/implementation to other languages (e.g. Java) and access through COM interfaces

To ease the learning curve for new users and users of OO languages.



C++ API Goals (cont'd)

To at least maintain the level of robustness and performance of the current implementation.

To provide for an efficient and direct access to field data by using DRM class-specific accessor methods.

To provide an automated and more efficient memory management system.

To provide a more consistent error handling and logging mechanism.

To allow for reduced user code complexity by exploiting features of the implementation language.

To provide extensibility where appropriate, such as specialized iterators and search filters.



C++ API Overview

Similarly to the C API, the OOI C++ API is a set of header files and shared/static libraries for all supported platforms.

All C++ classes are accessed through the “sedris” namespace to avoid name collisions with other libraries.

All C++ API functions are documented in the header files, and also contain sample code that demonstrate their use.

- HTML documentation is generated from the header files for easier browsing using the doxygen tool.

Reads and writes the same SEDRIS Transmittal Format that the C API does.

- All STF transmittals can be accessed with either API, provided they were created with the same SDK version (ie, the 4.0 C++ API will not read version 3.1.2 transmittals.)



C++ API Benefits

Allows user access to an object-oriented programming interface.

Memory management is taken care of by the C++ API.

- No worrying about SE_Stores.

Simplified methods for accessing fields, rather than C structure based access.

Error handling advantages

- Can always trap API exceptions without a lot of code.
- Exception mechanism handles clean up on errors:
 - Outstanding object references
 - Closing transmittals

Possible to extend API class functionality:

- Class inheritance
- Operator overloading



C++ API Classes

seWorkspace

- Groups access to transmittals in a common setup.

seTransmittal

- Access to transmittal data, creates DRM objects.

seObject

- Manipulates DRM object instances. The DRM hierarchy of classes is derived from *seDRMBase*, which is in turn derived from *seObject*.

seIterator

- Sequential access to DRM objects.

seException

- Exception handler class thrown from all API functions when an error occurs.



seWorkspace (1 of 2)

Used to manage transmittals under a common “theme”.

- Opens transmittals
 - `seWorkspace::openTransmittalByFile`
 - `seWorkspace::createTransmittal`
 - `seWorkspace::editTransmittal`
 - `seWorkspace::openTransmittalByURN`
- Specify API memory model
 - `seWorkspace::setMemoryModel`
- Manages transmittals
 - `seWorkspace::getOpenedTransmittalCount`
 - `seWorkspace::getOpenedTransmittal`



seWorkspace (2 of 2)

Specifies ITR behaviour for all transmittals in workspace

- `seWorkspace::setITRResolverPath`
 - This fulfills the same purpose as the `SEDRIIS_RESOLVER_PATH` environment variable in the C API.
- `seWorkspace::setITRBehaviour`
 - Same as the C API.
- `seWorkspace::setAccessModelInheritance`
 - Sets if transmittals reached via an ITR link should have the same access mode as the transmittal originally opened by the user.

Creates unresolved objects to use to reference other transmittals via ITR

- `seWorkspace::createUnresolvedObject`
 - Creates a special `seObject` that represents an ITR reference into another transmittal without having the other transmittal present.



seTransmittal (1 of 2)

Used to manipulate SEDRIS transmittals, and to create/access the DRM objects in the transmittal hierarchy

Accesses and sets the transmittal's root object

- `seTransmittal::setRootObject`
- `seTransmittal::getRootObject`

Creates and deletes DRM objects

- `seTransmittal::createObject`
- `seTransmittal::removeObject`

Accesses objects via object ID

- `seTransmittal::getObjectFromID`



seTransmittal (2 of 2)

Manages various ITR information at the transmittal level.

- Accesses all published objects in the transmittal.
 - **seTransmittal::getPublishedObject**
 - **seTransmittal::getPublishedObjectsIterator**
- Manages the transmittal's URN
 - **seTransmittal::getURN**
 - **seTransmittal::setURN**
- Accesses all ITR reference (to other transmittals) information
 - **seTransmittal::getITRReferenceCount**
 - **seTransmittal::getITRReference**
 - **seTransmittal::getITRReferenceLabelCount**
 - **seTransmittal::getITRReferenceLabel**



seObject (1 of 3)

Encapsulates access to all fields and data contained in DRM transmittal objects, including relationships to other DRM transmittal objects.

DRM type and field information

- seObject::getDRMClass
- seObject::getFields
- seObject::setFields

DRM relationship information

- Components
 - seObject::hasComponents
 - seObject::getComponent
 - seObject::getComponentIterator
 - seObject::addComponent
 - seObject::removeComponent
- Same 5 calls for Associates and Aggregates



seObject (2 of 3)

Provides various utility methods for each object.

Access to Object IDs

- seObject::getID

User data mechanism

- seObject::setUserData
- seObject::getUserData

Other utility methods

- seObject::isSameAs
- seObject::getTransmittal



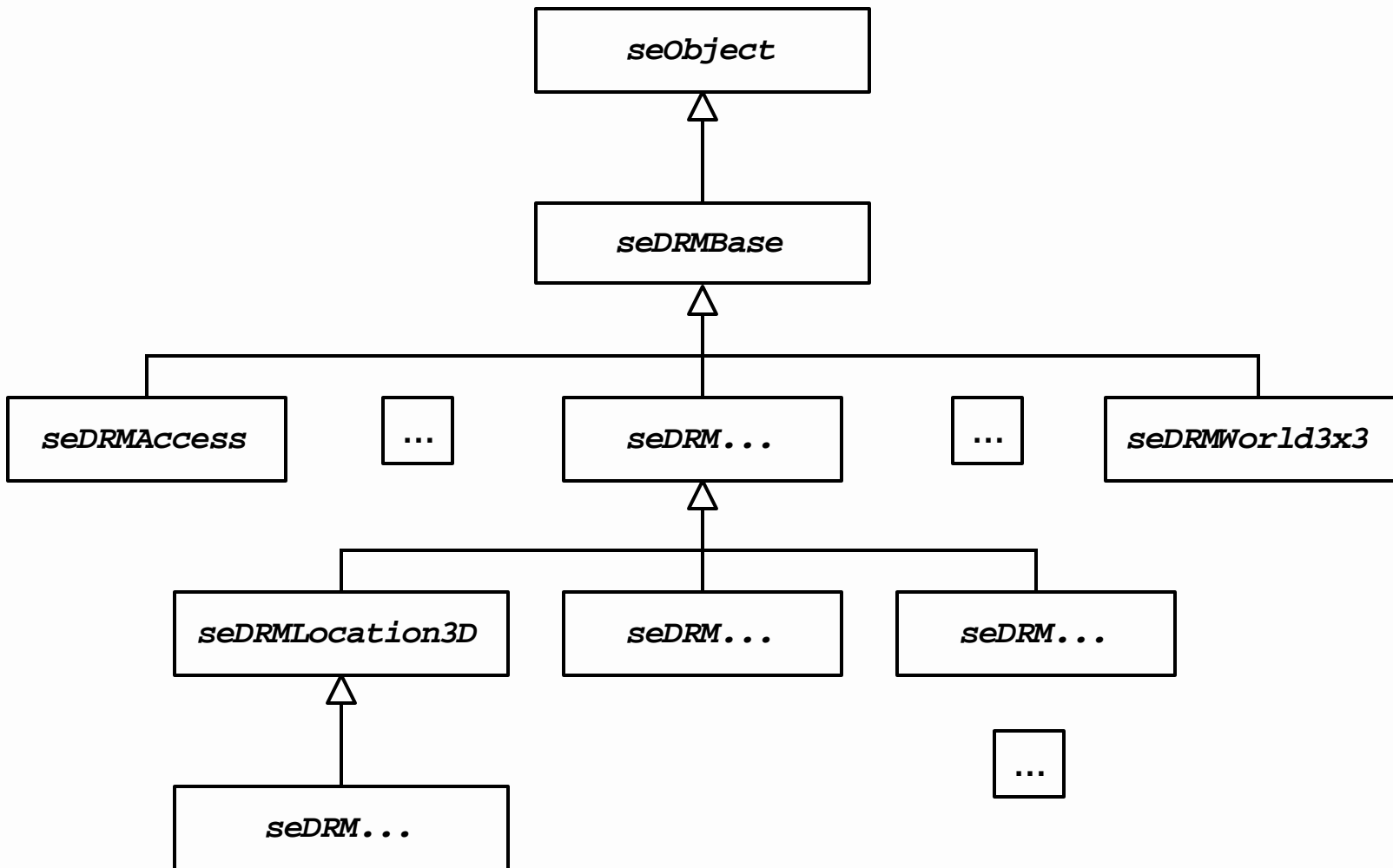
seObject (3 of 3)

Provides ITR related methods:

- seObject::isPublished
 - Check to see if an object is published so it can be referenced from other transmittals.
- seObject::publish
 - Publish the object so it can be referenced from other transmittals.
- seObject::unpublish
 - Remove the object from the list of objects that can be referenced from other transmittals.
- seObject::getPublishedLabelCount
 - Returns how many different labels the object is published as.
- seObject::getPublishedLabel
 - Gets each specific label the object was published as.
- seObject::isResolved
 - Determine if an object is resolved or not.
- seObject::resolve
 - If an object was retrieved via ITR, and the ITR behavior is not set to SE_ITR_BHVR_RESOLVE, then this function will resolve the object.
- seObject::getUnresolvedTransmittalURN
 - Gets the transmittal URN that the unresolved object references.
- seObject::getUnresolvedObjectLabel
 - Gets the object label that the unresolved object references.



seDRMBase Hierarchy (1 of 3)





seDRMBase Hierarchy (2 of 3)

All seDRMBase derived objects inherit from the seObject class.

- All seDRMBase objects share all seObject methods.

Each DRM class has it's own C++ API class under the seDRMBase hierarchy

- seDRMTransmittalRoot
- seDRMModelLibrary
- seDRMEnvironmentRoot
- etc.



seDRMBase Hierarchy (3 of 3)

Each seDRMBase derived class has direct access to individual field elements defined by the DRM class.

- For example, the seDRMTransmittalRoot has the following field methods:
 - get_name() / set_name()
 - get_major_DRM_version() / set_major_DRM_version()
 - get_minor_DRM_version() / set_minor_DRM_version()
 - get_interim_DRM_version() / set_interim_DRM_version()
 - etc...
- Can still access the SE_Fields structure via the inherited seObject method:
 - seObject::getFields / setObject::setFields



selerator

Generalized iterator for seObjects.

- Used to iterate through component objects, associate objects, and aggregate objects.

Much simpler than the C API iterator concepts:

- Only one level deep
- Can only filter by DRM class type
- ITR behaviour is set at the workspace level

Created by the seObject and one special case:

- seObject::getComponentIterator
- seTransmittal::getPublishedObjectsIterator

Functions

- selerator::isCompleted
- selerator::getCount
- selerator::getNext
 - With or without a link object parameter
- selerator::getNthNext
 - With or without a link object parameter



seException

The general exception class thrown from failed methods.

Exception contains two pieces of information:

- Error code (FILE_ERROR, TRANSMITTAL_UNACCESSIBLE, etc)
 - seException::getCode
- Description string (freeform)
 - seException::getWhat

Must be caught at the application/library level code, otherwise application will exit.

- Gives ability to clean up after the API encounters an error.

```
try
{
    seWorkspace wksp;
    seTransmittal xmtl;
    wksp::openTransmittalByFile( "bad_path.stf", xmtl )
}
catch( seException & e )
{
    cout << "Exception was thrown: " << e.getWhat << "(code: " << e.getCode() << ")" << endl;
    // clean up application memory and continue
}
```



Code Comparison: Example 1

Read the <Description> from the sample Lake Eola Transmittal

The description is stored in the fields of a SE_DRM_CLS_DESCRIPTION object which the DRM requires to be a component of the <Transmittal Root> object.

To do this we need to know:

- How to open and close a Transmittal for reading.
- How the C++ API handles exceptions, status codes, and error descriptions.
- How to access objects by traversing relationships.
- How to access fields of an object.



Example 1: C API Code

```
main() {  
    SE_Return_Code  ret;  
    SE_Transmittal  xmittal;  
    SE_Object       root_obj, desc_obj;  
    SE_Store        store;  
    SE_FIELDS_PTR   desc_flds;  
  
    ret = SE_OpenTransmittalByFile( "Lake_Eola.stf", "stf",  
                                     SE_AC_MODE_READ_ONLY, &xmittal );  
    ret = SE_GetRootObject( xmittal, &root_obj );  
  
    ret = SE_GetComponent3( root_obj, SE_DRM_CLS_DESCRIPTION, &desc_obj );  
  
    ret = SE_CreateStore( "stf", &store );  
  
    ret = SE_GetFields( desc_obj, store, &desc_flds );  
  
    printf( "Transmittal Description %s\n", desc_flds->u.Description.abstract.characters );  
  
    ret = SE_FreeStore( store );  
    ret = SE_FreeObject( root_obj );  
    ret = SE_FreeObject( desc_obj );  
  
    ret = SE_CloseTransmittal( xmittal );  
}
```



Example 1: C++ API Code

```
main()
{
    seWorkspace          workspace;
    seTransmittal        xmittal;
    seDRMTransmittalRoot root_obj;
    seDRMDescription     desc_obj;

    try
    {
        workspace.openTransmittalByFile( "Lake_Eola.stf", xmittal );
        xmittal.getRootObject( root_obj );

        if( root_obj.getComponent( desc_obj ))
        {
            cout << "Transmittal Description " << desc_obj.get_description().characters << endl;
        }

        xmittal.close();
    }
    catch( seException & e )
    {
        cout << "An error occurred: " << e.getWhat() << endl;
    }
}
```



Example 1: Summary

No longer need to worry about memory management.

- Constructors and destructors handle it for you.
 - No more SE_FreeObject calls.

Didn't have to check the status code of every function. One try...catch block handles everything.

Didn't have to remember "SE_AC_MODE_READ_ONLY" everytime you want to open a transmittal.

Didn't have to specify SE_DRM_CLS_DESCRIPTION because a seDRMDescription object reference was passed into root_obj.GetComponent.



Code Comparison: Example 2

Retrieve all of the <Polygons> in the sample Lake Eola Transmittal

We do this by recursively retrieving components from the <Transmittal Root> on down.

This example will introduce:

- selIterators
- How to create and use selIterators recursively with help from the DRM.



Example 2: C API Code to retrieve <Polygons>

```
void FindPolygons( SE_Transmittal xmittal, SE_Object root_obj )
{
    SE_Return_Code      ret;
    SE_Search_Rule      polygon_search_rules[] =
    {
        SE_DRM_CLASS_MATCH( POLYGON ) /* Infinite depth */
    }
    SE_Search_Filter     search_filter;
    SE_Iterator          iterator;
    SE_Object            polygon_obj;

    ret = SE_CreateSearchFilter( xmittal, polygon_search_rules, &search_filter );

    ret = SE_InitializeComponentIterator3( root_obj, search_filter, &iterator );

    while( SE_IsIteratorEmpty ( iterator) == SE_FALSE )
    {
        ret = SE_GetNextObject( iterator, &polygon_obj, NULL ;
        /* process the polygon object */

        ret = SE_FreeObject( polygon_obj );
    }
    ret = SE_FreeIterator( iterator );
    ret = SE_FreeSearchFilter( search_filter );
}
```



Example 2: C++ API Code to Retrieve Polygons

```
void FindPolygons( seObject current_obj ){

    seIterator  iterator;
    seObject    child_obj;
    SE_DRM_Class drm_type;

    current_obj.getComponentIterator( iterator );

    while(!iterator.isCompleted()){

        iterator.getNext( child_obj );
        drm_type = child_obj.getDRMClass();

        if( drm_type == SE_DRM_CLS_POLYGON ){
            /* process the polygon object */
        }
        else{
            /* prune search paths that won't result in finding <Polygons> */
            if( SE_ShortestAggPath[drm_type][SE_DRM_CLS_POLYGON] != -1 )
            {
                FindPolygons( child_obj );    /* recurse down the transmittal tree */
            }
        }
    }
}
```



Example 2: Summary

No search rules or search filters.

Had to use recursion to produce same functionality as the C API search filter mechanism.

Also had to use SE_ShortestAggPath to limit wasteful recursion.

- C API uses SE_ShortestAggPath under the covers.
- Had to use getDRMClass to use in SE_ShortestAggPath.

Could utilize or extend the C++ selector class to create a new iterator class that goes many levels deep and/or provides more filtering mechanisms.



Code Comparison: Example 3

We want to create a geometry representation of Lake Eola

To do this we need to know:

- How to open a Transmittal for creation.
- How to create SEDRIIS objects.
- How to set the fields of an object.
- How to create relationships between objects.
- How to set the Root Object in the Transmittal.



Example 3: C API Code

```
main()
{
    SE_Return_Code ret;
    SE_Transmittal  xmittal;
    SE_Object       root_obj, description_obj;
    SE_Fields       root_flds, description_flds;

    ret = SE_OpenTransmittalFile( "Lake_Eola.stf", "stf",
                                  SE_AC_MODE_CREATE, &xmittal );

    ret = SE_CreateObject( xmittal, SE_DRM_CLS_TRANSMITTAL_ROOT,
                           &root_obj );
    ret = SE_SetRootObject( xmittal, root_obj, NULL);

    ret = SE_CreateObject( xmittal, SE_DRM_CLS_DESCRIPTION,
                           &description_obj );
    ret = SE_SetFieldsToDefault( SE_DRM_CLS_DESCRIPTION, &description_flds );
}
```



Example 3: C API Code [2 of 2]

```
strcpy( description_flds.u.Description.abstract.characters,  
                                               "Lake Eola, Geometry" );  
description_flds.u.Description.abstract.length =  
                                               strlen("Lake Eola, Geometry " );  
  
ret = SE_PutFields( description_obj, &description_flds );  
  
ret = SE_AddComponentRelationship( root_obj, description_obj, NULL );  
  
/* create the Geometry Objects under the Environment Root. */  
create_environment_root_objects( xmittal, root_obj );  
  
ret = SE_FreeObject( root_obj );  
ret = SE_FreeObject( description_obj );  
  
ret = SE_CloseTransmittal( xmittal );  
}
```



Example 3: C++ API Code to create objects

```
main()
{
    try
    {
        seWorkspace wksp;
        seTransmittal xmittal;
        seDRMTransmittalRoot root_obj;
        seDRMDescription description_obj;

        wksp.createTransmittal( "Lake_Eola.stf", xmittal );
        xmittal.createObject( root_obj );
        xmittal.setRootObject( root_obj );

        xmittal.createObject( description_obj );
        description_obj.setFields( SE_DESCRIPTION_FIELDS_DEFAULT );
        description_obj.set_abstract( "Lake Eola, Geometry " );

        root_obj.addComponent( description_obj );

        /* create geometry objects under <Environment Root> */
        create_environment_root_objects( xmittal, root_obj );
    }
    catch( seException & e )
    {
        cout << "An error occurred: " << e.getWhat() << endl;
    }
}
```



Example 3: Summary

Simple createTransmittal call replaces SE_OpenTransmittalByFile call with “SE_AC_MODE_CREATE”.

Instead of manipulating the SE_Fields object and then call SE_PutFields, we manipulate the fields via direct methods on the seDRMDescription object.

SE_AddComponentRelationship is replaced with addComponent

- Link object is now optional, no NULL parameter required.

Transmittal is closed when the workspace drops out of scope.



C++ Summary

We've covered the basics of the C++ API

- C++ API is very different than the C API, but DRM classes and structures are the same for both, so code will be structured very similarly.
- C++ API handles a lot of memory management issues that you have to worry about with the C API
- C++ API has uses exception handling rather than return codes for error conditions.

C++ API will be released with the upcoming SEDRIIS 4.0 SDK release.

Where to go from here:

- [*Advanced Use of the SEDRIIS SDK*](#) tutorial will be using the C++ API.



Summary

We have:

- Highlighted API capabilities.
- Demonstrated API functionality through small applications and routines.
- Introduced and explained how to use API functions and data types.
- Identified common mistakes in the use of the API.

You should:

- Understand the capabilities of the SEDRIIS API.
- Recognize the key functions and data structures.
- Know where to go for more information:
 - www.sedris.org
 - help@sedris.org
 - *How To Produce and Consume Transmittals* tutorial