Science and Engineering Research Council
**Rutherford Appleton Laboratory**
Chilton DIDCOT Oxon OX11 0QX

RAL-93-061



# GKS-9x: A Specification of the Framework

**L B Damnjanovic and D A Duce**



August 1993

# GKS-9x: A Specification of the Framework

*L.B. Damnjanovic and D.A. Duce*

Informatics Department, Rutherford Appleton Laboratory, Chilton, Didcot, Oxon OX11 0QX, UK

## 1. Introduction

The Graphical Kernel System, GKS, was published as an International Standard in 1985,[7] and is now being revised.[1,4] Prior to exploring approaches to implementing some of the new functionality, the authors prepared a formal description of the key parts of the framework of GKS-9x which were relevant to the implementation issues which were later studied. The implementation studies are reported in the papers by Damnjanovic, Duce and Robinson.[2,3]

The formal description used the OBJ notation and follows the approach described in Duce and Damnjanovic.[5] The content proposed for the revised GKS, GKS-9x, has changed since the latter paper was written, but the specification follows very similar lines.

In order that this paper can be read independently from the implementation studies paper[2] the next section repeats much of the description of the current draft of GKS-9x contained in that paper.

## 2. An Overview of GKS-9x

### 2.1. The NDC Picture

In GKS:1985, graphical output primitives are defined by the application in a world coordinate system, are transformed to Normalized Device Coordinates (NDC) by a normalization transformation and are then passed to workstations for display. Each workstation then applies its own workstation transformation which maps a region of NDC space onto a specified region of the workstation's display space. The normalization transformation can be thought of as creating a scene in NDC space from components defined in different world coordinate systems and a workstation can be thought of as a camera viewing the scene.

The model as presented above is not actually present in GKS:1985 in such a pure form and this has led to a number of difficulties with the design of the first generation graphics standards, in particular in defining the relationship between GKS:1985 and the Computer Graphics Metafile (CGM).[4]

The first major new concept to be introduced in GKS-9x was the idea of an NDC picture, which is manipulated in well-defined ways and whose contents are always well-defined. The NDC picture is organized as a sequence of output primitives.

The overall architecture of GKS-9x is summarized in figure 1.

### 2.2. Namesets and Selection Criteria

The Disley workshop report recommendations listed in section 1 referred to the need for more generality in naming primitives based on the nameset concept of PHIGS.[8] In PHIGS, a set of names (normally represented in language bindings by an array or list of integers) may be associated with an output primitive. This nameset attribute is used to control whether the primitive is visible, highlighted or detectable by a particular pick device on a particular workstation. Control is exercised by filters. A filter consists of two sets of names, an inclusion set and an exclusion set. A primitive has a particular property such as highlighting if its nameset attribute has at least one name in common with the inclusion set of the filter and no names in common with its exclusion set.

In GKS-9x, the filters have been replaced by application-specifiable selection criteria. A primitive is selected for some purpose if its nameset attribute satisfies the corresponding selection criterion. Selection

criteria are constructed from set-theoretic comparison operators (equality with a specified set of names, subset and superset of specified sets of names) and the logical operators (and, or, not). The comparison operators are:

| | |
|---|---|
| contains(*ns*) | $ns \subseteq pns$ |
| isin(*ns*) | $pns \subseteq ns$ |
| equals(*ns*) | $ns = pns$ |
| SELECTALL | True (for any nameset) |
| REJECTALL | False (for any nameset) |

In the table above, *pns* denotes the nameset of the primitive to which the criterion is being applied and *ns* is a nameset supplied as parameter to the operator.

As an example, the criterion:

contains({SWAN, LAKE})

selects all primitives tagged with 'SWAN' and 'LAKE', while:

not contains({SWAN, LAKE})

would select all other primitives. Selection criteria are explained in more detail in a later section.

The main difference between selection criteria and PHIGS filters is that for filters the function applied to namesets is fixed, whereas for selection criteria the function is specified by the application. The range of uses to which namesets and selection criteria are put is much broader than the uses of namesets and filters in PHIGS, as will become evident later in this section.

### 2.3. Picture Part Store

To support the composition of NDC pictures, a separate picture part store has been provided. A picture part is a named sequence of output primitives. Picture parts can be copied from the picture part store to the NDC picture and a subsequence of primitives in the NDC picture which satisfy a specified selection criterion, can be copied into the picture part store.

The function to copy picture parts into the NDC picture also uses a selection criterion to select a subsequence of the primitives in the picture part for the copy operation. In addition, a specified set of names can be added to the nameset attribute of each new primitive created in the NDC picture. This enables primitives created from different copy operations on the same picture part to be distinguished.

It is envisaged that the picture part store will take over the functions of the old GKS segment store while producing a much richer storage facility.

### 2.4. Transformations

All primitives in GKS-9x have two transformation matrices associated with them, the global transformation and the local transformation. These attributes have been introduced in order to provide a flexible mechanism for constructing NDC pictures from picture parts. The local transformation matrix enables primitives to be composed to construct objects, or subparts of pictures. If the global transformation attributes of all the primitives making up an object are set to the same value, the effect is to control the overall positioning of the object in the NDC picture. Thus the local transformation is used to construct objects and the global transformation to control the positioning of an instance of an object in the NDC picture. Both local and global transformations can be changed when picture parts are copied into the NDC picture. In consequence, some of the modelling capabilities of PHIGS are available in GKS-9x.

### 2.5. Modification of the NDC Picture

The attributes of primitives in the NDC picture can be edited. The functionality provided is to set a specified attribute for every primitive in the NDC picture satisfying a specified selection criterion, to a new value. In addition, functions are provided to add or remove specified sets of names from the nameset attributes of a selected sequence of primitives in the NDC picture. These additional functions are often a more convenient way of manipulating nameset attributes than the editing function which just enables a nameset

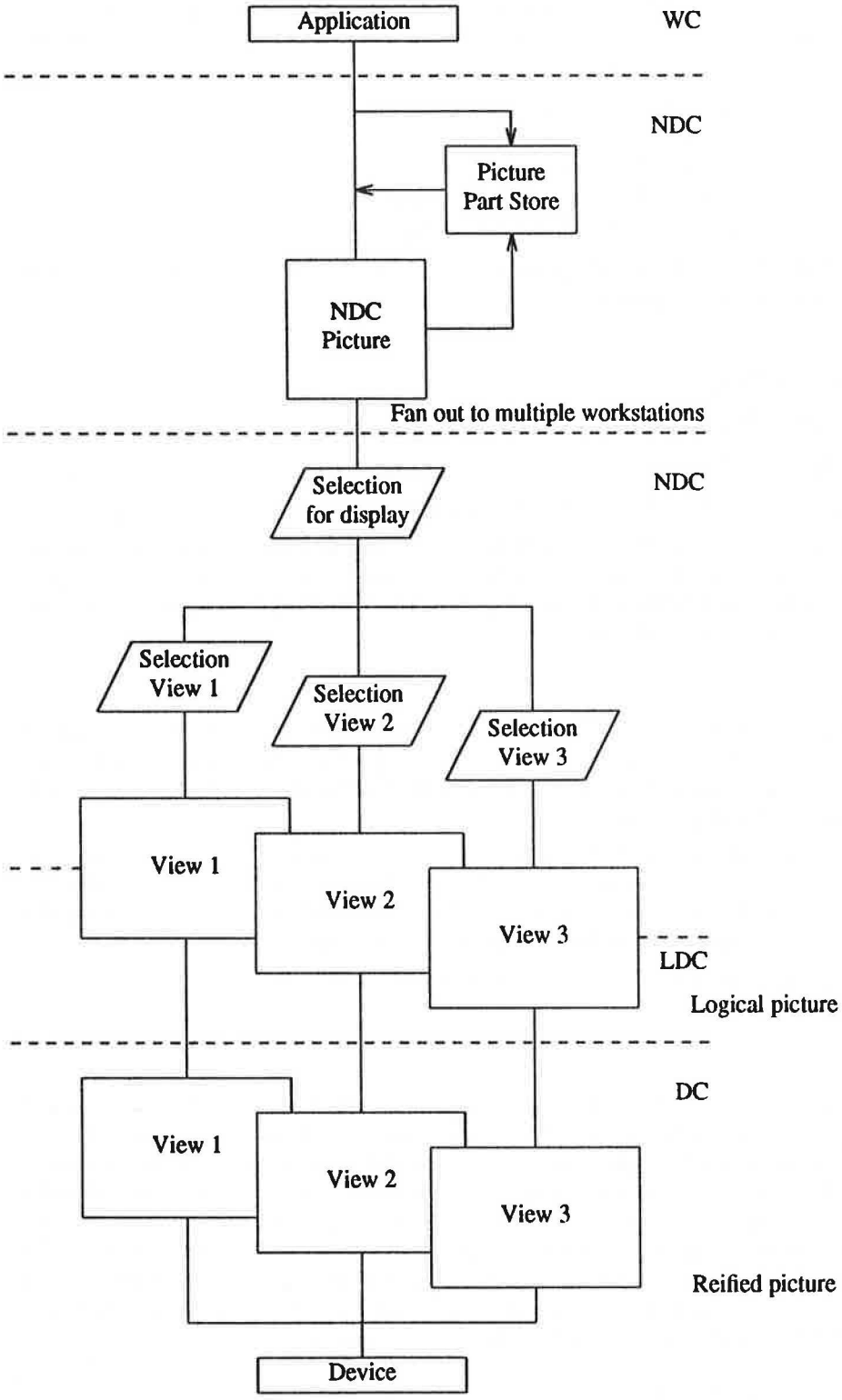


**Figure 1: Architecture of GKS-9x**

attribute to be replaced by a new value.

Primitives may be deleted from the NDC picture and again a selection criterion is used to specify which primitives the operation is to apply to.

As stated earlier, the NDC picture is a sequence of output primitives. This means that there is a well-defined order to the primitives in the NDC picture and this order is preserved by all the operations on the NDC picture. This order is also respected when the NDC picture is displayed on a workstation. It may sometimes be necessary to change the order of the primitives in the NDC picture and to do this, a function REORDER NDC PICTURE, is provided. This function moves the subsequence of primitives which satisfy a selection criterion to either the start or the end of the NDC picture sequence.

### 2.6. Display

The mechanism in GKS-9x for displaying the NDC picture on workstations is also more general than the mechanism in GKS:1985. The key idea is that each open workstation acts as a camera, viewing the NDC picture. As the NDC picture is changed, the picture displayed on the workstation is updated accordingly. The NDC picture is assumed to be always up-to-date. The function SET NDC VISUAL EFFECTS allows the continual refreshing of the workstation display to be suspended and resumed. Using the camera analogy, this is equivalent to closing and opening the shutter of the camera.

A selection criterion associated with each open workstation controls which primitives in the NDC picture are eligible for display on that workstation.

### 2.7. Multiple Views

In GKS:1985, the workstation transformation maps the NDC picture directly into device coordinates (DC), so that only a single view of the NDC picture is displayed on each workstation. GKS-3D and PHIGS introduced a viewing transformation applied before the workstation transformation. The motivation for the viewing transformation in these systems was obviously to perform 3D viewing operations. Multiple views could be defined on each workstation and a view index attribute associated with each primitive is used to select the particular view transformation to be applied.

In fact the idea of viewing is just as relevant in 2D as in 3D, and so GKS-9x contains a view transformation between the NDC picture and the workstation transformation. The GKS-9x viewing mechanism is a generalization of the GKS-3D and PHIGS mechanisms. In those systems a single primitive can only be subjected to a single viewing operation. In GKS-9x, multiple view transformations can be defined, and a primitive can be displayed in any of the defined views. Selection criteria associated with the views determine which primitives appear in which views of the NDC picture. One of the extensions made by some PHIGS implementations has been to add similar facilities.

The NDC picture is transformed into a logical picture on the workstation, partitioned into views by the view transformations. The logical picture is defined in Logical Device Coordinates (LDC). Output primitives in the logical picture have logical attributes and NDC attributes bound to them. The attribute binding mechanism is exactly the same as GKS:1985, though the description has been changed somewhat to provide a more comprehensible description in the GKS-9x framework. The logical picture is then transformed into the reified picture which is realized on the workstation's display space.

### 2.8. Interfaces

GKS-9x has been developed in step with the Computer Graphics Reference Model (CGRM).[9] The GKS-9x framework maps readily into the framework provided by the CGRM. Some formal description work has been carried out to verify this claim.[6]

GKS-9x has also introduced the concept of a backdrop. Primitives may be routed to the backdrop, in which case they do not form part of the NDC, logical or reified pictures. This can be useful for applications which generate large volumes of graphical data which do not require any manipulation.

### 2.9. New Primitives

GKS-9x has introduced many other improvements, for example additional types of output primitives have been included to generate non-uniform B-spline curves, conic sections, conic sectors and conic segments. However, the implementation work which is the subject of this paper did not address these areas of GKS-9x and so they are not described any further here.

### 2.10. Operations Modelled

The GKS-9x operations modelled in the specification are:

OPEN GKS
CREATE OUTPUT PRIMITIVE
SET PRIMITIVE ATTRIBUTE
ADD SET OF NAMES TO NAMESET
REMOVE SET OF NAMES FROM NAMESET
SET WINDOW AND VIEWPORT
SET NORMALIZATION TRANSFORMATION NUMBER
DELETE PRIMITIVES
REMOVE SET OF NAMES FROM NDC PICTURE
ADD SET OF NAMES TO NDC PICTURE
SET NDC PICTURE PRIMITIVE ATTRIBUTE
REORDER NDC PICTURE
BEGIN PICTURE PART
BEGIN PICTURE PART AGAIN
END PICTURE PART
APPEND PICTURE PART
RENAME PICTURE PART
DELETE PICTURE PART
COPY PICTURE PART FROM PICTURE PART STORE
COPY NDC PICTURE TO PICTURE PART STORE
OPEN WORKSTATION
CLOSE WORKSTATION
SET REPRESENTATION
SET VIEW SELECTION CRITERION
SET VIEW
SET NDC VISUAL EFFECTS
SET WORKSTATION WINDOW AND VIEWPORT
SET WORKSTATION SELECTION CRITERION

## 3. The GKS-9x Specification

### 3.1. Structure

The specification consists of a number of OBJ modules, described below.

### 3.2. INTEGER

Coordinate systems in GKS are based on real numbers. The OBJ interpreter used in this work only supported natural numbers. For the purposes of this specification, coordinates were expressed as integers and a mapping of integers onto natural numbers was provided.

**obj** *INTEGER*
**sorts** *int*
**ops** *_+_ : int int → int (ASSOC COMM )*
*_-_ : int int → int*
*_*_ : int int → int (ASSOC COMM )*
*_div_ : int int → int*
*ps_ : nat → int*
*ng_ : nat → int*
**vars** *i,j : int*
*n,m : nat*
**eqns** *(ps n + ps m = ps (n+m))*
*(ng n + ps m = ps (m-n) IF (m>n))*
*(ng n + ps m = ng (n-m) IF (n>m))*
*(ng n + ng m = ng (n+m))*
*(ng n + ps m = ps (m-m) IF (n==m))*

*(ps n - ps m = ps (n-m) IF (n>m))*
*(ps n - ps m = ng (m-n) IF (m>n))*
*(ps n - ps m = ps (n-m) IF (n==m))*
*(ng n - ng m = ng (n+m))*
*(ps n - ng m = ps (n+m))*
*(ng n - ps m = ng (n+m))*

*(ng n * ps m = ng (m*n))*
*(ng n * ng m = ps (m*n))*
*(ps n * ps m = ps (n*m))*

*(ps n div ps m = ps (n div m))*
*(ps n div ng m = ng (n div m))*
*(ng n div ps m = ng (n div m))*
*(ng n div ng m = ps (n div m))*

**jbo**

### 3.3. NAMES

This object defines the basic sorts for names, picture part names, workstation identifiers and normalization transformation numbers. For illustration, limited numbers of constants of each sort are defined.

**obj** *NAMES*
**sorts** *Name PicPartName WsId NormTran*
**ops** *WPT1*: → *Name*
*WPT2*: → *Name*
*WPT3*: → *Name*
*NAM,BIG,BEN*: → *Name*
*IME,RED,BALL*: → *Name*
*TRIANGLE,SMALL*: → *Name*
*POINT,A,B,LINE*: → *Name*
*BLUE,GREEN,HAT*: → *Name*
*N4*: → *Name*
*N5*: → *Name*
*N6*: → *Name*
*N7*: → *Name*
*NONAMEPN*: → *PicPartName*
*P1*: → *PicPartName*
*P2*: → *PicPartName*
*P3*: → *PicPartName*
*P4*: → *PicPartName*
*OPENPP*: → *PicPartName*
*W1*: → *WsId*
*W2*: → *WsId*
*W3*: → *WsId*
*W4*: → *WsId*
*pptoname*: *PicPartName* → *Name*
*wsidtoname*: *WsId* → *Name*
*NTN0*: → *NormTran*
*NTN1*: → *NormTran*
*NTN2*: → *NormTran*
**jbo**

### 3.4. NAMESETS

This object defines the basic operations on namesets (sets of *Names*), including operations to add a name to a nameset (*addname*), remove a name from a nameset (*removessn*), remove a set of names from a nameset (*subtractset*) and test for membership of a nameset (*member*).

**obj** *NAMESETS /NAMES*
**sorts** *NameSet*
**ops** *emptyNS*: → *NameSet*
*_U_*: *NameSet NameSet* → *NameSet* (*ASSOC COMM ID:emptyNS*)
*addname*: *Name NameSet* → *NameSet*
*mkSet*: *Name* → *NameSet*
*renamessn*: *Name Name NameSet* → *NameSet*
*subtractset*: *NameSet NameSet* → *NameSet*
*removessn*: *Name NameSet* → *NameSet*
*member*: *Name NameSet* → *BOOL*
**vars** *ns,ns1,ns2:NameSet*
*on,nn,n,n1,m:Name*
**eqns** *((ns) U (ns) =ns)*

*(addname(n,ns) = (mkSet(n)) U (ns))*
*(renamessn(on,nn,emptyNS) = emptyNS)*
*(renamessn(on,nn,(ns1) U (ns2)) = (renamessn(on,nn,ns1)) U (renamessn(on,nn,ns2)))*
*(renamessn(on,nn,mkSet(n)) = mkSet(n) IF not (on == n))*
*(renamessn(on,nn,mkSet(on)) = mkSet(nn))*

*(subtractset(ns,(ns1) U (ns2)) = (subtractset(ns,ns1)) U (subtractset(ns,ns2)))*
*(subtractset((ns1) U (ns2),ns) = subtractset(ns2,subtractset(ns1,ns)))*
*(subtractset(ns,emptyNS) = emptyNS)*
*(subtractset(ns,ns) = emptyNS)*
*(subtractset(emptyNS,ns) = ns)*
*(subtractset(mkSet(n),mkSet(n1)) = mkSet(n1) IF not (n == n1))*

*(removessn(n,(ns1) U (ns2)) = (removessn(n,ns1)) U (removessn(n,ns2)))*
*(removessn(n,emptyNS) = emptyNS)*
*(removessn(n,mkSet(n)) = emptyNS)*
*(removessn(n,mkSet(n1)) = mkSet(n1) IF not (n == n1))*

*(member(n,emptyNS) = F)*
*(member(n,mkSet(m)) = (n==m))*
*(member(m, (ns1)U(ns2)) = (member(m,ns1)) or (member(m,ns2)))*
**jbo**

### 3.5. ATTRIBUTES

For illustration, a limited number of constants of each sort of attribute values are defined. The operation *selectlogattr* selects the value of a logical attribute from individually specified values or from a bundle, depending on the value of the corresponding attribute source flag.

**obj** *MATRIX23/ INTEGER*
**sorts** *Matrix23*
**ops** *mktransfcor*: *int int int int int int* → *Matrix23*
*mult*: *Matrix23 Matrix23* → *Matrix23*
**vars** *a,b,c,d,e,f,g,h,z,l,m,n:int*
**eqns** *(mult(mktransfcor(a,b,c,d,e,f), mktransfcor(g,h,z,l,m,n))*
*= mktransfcor((a*g)+(d*h),(b*g)+(e*h),(c*g)+(f*h)+z,(a*l)+(d*m),(b*l)+(e*m),(c*l)+(f*m)+n))*
**jbo**

**obj** *ATTRIBUTES /NAMESETS MATRIX23*
**sorts** *Identification Source Logical NDCattr asfs ASF PlInd*
*Linetype Linewidth Highl Det PlBun TransMode*
**ops** *identifa: NameSet → Identification*
*asfsa: ASF ASF → asfs*
*sourcea: asfs PlInd → Source*
*logicala: Linetype Linewidth → Logical*
*ndca: Matrix23 Matrix23 → NDCattr*
*REPLACE, PRE, POST: → TransMode*
*modifyndca: Matrix23 TransMode Matrix23 TransMode NDCattr → NDCattr*
*mkBundle: Linetype Linewidth → PlBun*
*selectlogattr: asfs Logical PlBun → Logical*
*LOCAL, GLOBAL: → Matrix23*
*INDIVIDUAL, BUNDLED: → ASF*
*PLI0, PLI1, PLI2: → PlInd*
*SOLID, DASHED, DOTTED: → Linetype*
*THIN, THICK, MEDIUM: → Linewidth*
*NORMAL, HIGHLIGHTED: → Highl*
*DETECTABLE, UNDETECTABLE: → Det*
**vars** *lt,lt1: Linetype*
*lw,lw1: Linewidth*
*global,global1, local,local1: Matrix23*
*trmode, trmode1: TransMode*
**eqns** *(modifyndca(global,REPLACE,local,REPLACE,ndca(local1,global1)) = ndca(local,global))*
*(modifyndca(global,REPLACE,local,PRE,ndca(local1,global1))*
*= ndca(mult(local1,local), global))*
*(modifyndca(global,REPLACE,local,POST,ndca(local1,global1))*
*= ndca(mult(local,local1), global))*
*(modifyndca(global,PRE,local,REPLACE,ndca(local1,global1))*
*= ndca(local,mult(global1, global)))*
*(modifyndca(global,POST,local,REPLACE,ndca(local1,global1))*
*= ndca(local,mult(global,global1)))*
*(modifyndca(global,PRE,local,PRE,ndca(local1,global1))*
*= ndca(mult(local1,local), mult(global1, global)))*
*(modifyndca(global,POST,local,POST,ndca(local1,global1))*
*= ndca(mult(local,local1), mult(global,global1)))*

*(selectlogattr(asfsa(BUNDLED,BUNDLED),logicala(lt,lw), mkBundle(lt1,lw1))*
*= logicala(lt1,lw1))*
*(selectlogattr(asfsa(BUNDLED,INDIVIDUAL),logicala(lt,lw), mkBundle(lt1,lw1))*
*= logicala(lt1,lw))*
*(selectlogattr(asfsa(INDIVIDUAL,BUNDLED),logicala(lt,lw), mkBundle(lt1,lw1))*
*= logicala(lt,lw1))*
*(selectlogattr(asfsa(INDIVIDUAL,INDIVIDUAL),logicala(lt,lw), mkBundle(lt1,lw1))*
*= logicala(lt,lw))*
**jbo**

### 3.6. SELECTION

This object defines selection criteria and an operation *satisfy* which delivers true (*T*) if a nameset satisfies a given selection criterion and false (*F*) if it does not.

**obj** *SELECTION / NAMESETS ATTRIBUTES*
**sorts** *SelectCrit SelectDisp SelectHighl SelectDet*
**ops** *SELECTALL,REJECTALL: → SelectCrit*

*DISPLAY*: → *SelectDisp*
*HIGHLIGHTING*: → *SelectHighl*
*DETECTABILITY*: → *SelectDet*
*contains*: *NameSet* → *SelectCrit*
*isin*: *NameSet* → *SelectCrit*
*sequals* : *NameSet* → *SelectCrit*
*sand*: *SelectCrit SelectCrit* → *SelectCrit*
*sor*: *SelectCrit SelectCrit* → *SelectCrit*
*snot*: *SelectCrit* → *SelectCrit*

*satisfy*: *NameSet SelectCrit* → *BOOL*

*highlt*: *NameSet SelectCrit* → *Highl*
*dect*: *NameSet SelectCrit* → *Det*

**vars** *ns,ns1,ns2:NameSet*
*n:Name*
*s,s1,s2:SelectCrit*
*h:Highl*
*d:Det*
*Selcrit,sv,sh,sd:SelectCrit*
*wsid:WsId*

**eqns** (*satisfy*(*ns,sand*(*s1,s2*)) = (*satisfy*(*ns,s1*)) *and* (*satisfy*(*ns,s2*)))
(*satisfy*(*ns, sor*(*s1,s2*)) = (*satisfy*(*ns,s1*)) *or* (*satisfy*(*ns,s2*)))
(*satisfy*(*ns, snot*(*s*)) = *not*(*satisfy*(*ns,s*)))
(*satisfy*(*ns,contains*(*mkSet*(*n*))) = *member*(*n,ns*))
(*satisfy*(*ns,contains*(*emptyNS*)) = *T*)
(*satisfy*(*ns,contains*((*ns1*)*U*(*ns2*))) = (*satisfy*(*ns,contains*(*ns1*))) *and* (*satisfy*(*ns,contains*(*ns2*))))
(*satisfy*(*emptyNS,isin*(*ns*)) = *T*)
(*satisfy*(*mkSet*(*n*),*isin*(*ns*)) = *member*(*n,ns*))
(*satisfy*((*ns1*) *U* (*ns2*), *isin*(*ns*)) = *satisfy*(*ns1,isin*(*ns*)) *and satisfy*(*ns2, isin*(*ns*)))
(*satisfy*(*ns,sequals*(*ns1*)) = (*ns*==*ns1*))
(*satisfy*(*ns,SELECTALL*) = *T*)
(*satisfy*(*ns,REJECTALL*) = *F*)

(*highlt*(*ns,sh*) = *HIGHLIGHTED IF* (*satisfy*(*ns,sh*)==*T*))
(*highlt*(*ns,sh*) = *NORMAL IF not*(*satisfy*(*ns,sh*)==*T*))

(*dect*(*ns,sd*) = *DETECTABLE IF* (*satisfy*(*ns,sd*)==*T*))
(*dect*(*ns,sd*) = *UNDETECTABLE IF not*(*satisfy*(*ns,sd*)==*T*))

**jbo**

### 3.7. TABLE

This object defines a general table sort which is used later to define bundle and other tables.

**obj** *TABLE*
**sorts** *tb index telem*
**ops** *emptyTb*: → *tb*
*_U_*: *tb tb* → *tb* (*ASSOC COMM ID*:*emptyTb*)
*_%_*: *index telem* → *tb*
*__*: *tb index* → *tb*
*_+_*: *tb tb* → *tb*
*_[_]*: *tb index* → *telem*
**vars** *t1,t2*: *tb*
*i,j*: *index*

*el*: *telem*
**eqns** ((*t1*)U(*t1*) = *t1*)
((*i* % *el*)\(*i*) = *emptyTb*)
((*emptyTb*)\(*i*) = *emptyTb*)
((*j* % *el*)\(*i*) = (*j* % *el*) *IF not* (*i*==*j*))
((((*t1*) U (*t2*)))\(*i*) = ((*t1*)\(*i*)) U ((*t2*)\(*i*)))
((*t1*) + (*i* % *el*) = ((*t1*)\(*i*)) U ((*i* % *el*)))
(((*t1*) U (*i* % *el*))[ *i* ] = *el*)
(((*t1*) U (*i* % *el*))[ *j* ] = *t1* [ *j* ] *IF not* (*i*==*j*))
**jbo**

### 3.8. NDCPOINT, NDCPOINTS, LISTOFNDCPOINTS

These objects define points in WC and NDC coordinates, list of such points and an operation to transform lists of points from WC to NDC.

**obj** *NDCPOINT /MATRIX23 INTEGER*
**sorts** *WCPoint NDCPoint*
**ops** *mkWCPoint*: *int int* → *WCPoint*
*mkNDCPoint*: *int int* → *NDCPoint*
*transf*: *WCPoint Matrix23* → *NDCPoint*
*transfNDC*: *NDCPoint Matrix23* → *NDCPoint*
*X,Y*: → *int*
**vars** *x,y,a,b,c,d,e,f*: *int*
**eqns** (*transf*(*mkWCPoint*(*x,y*),*mktransfcor*(*a,b,c,d,e,f*))
= *mkNDCPoint*((*x***a*)+(*y***b*)+*c*, (*x***d*)+(*y***e*)+*f*))
(*transfNDC*(*mkNDCPoint*(*x,y*),*mktransfcor*(*a,b,c,d,e,f*))
= *mkNDCPoint*((*x***a*)+(*y***b*)+*c*, (*x***d*)+(*y***e*)+*f*))
**jbo**

**obj** *NDCPOINTS /NDCPOINT*
**sorts** *WCPoints NDCPoints*
**ops** *emptyWCPoints*: → *WCPoints*
*addWCpoint*: *WCPoint WCPoints* → *WCPoints*
*emptyNDCPoints*: → *NDCPoints*
*addNDCpoint*: *NDCPoint NDCPoints* → *NDCPoints*
*transfPoints*: *WCPoints Matrix23* → *NDCPoints*
*transfNDCPoints*: *NDCPoints Matrix23* → *NDCPoints*
**vars** *pn*: *WCPoint*
*ndcpoint*: *NDCPoint*
*ndcpoints*: *NDCPoints*
*pnts*: *WCPoints*
*mat*: *Matrix23*
**eqns** (*transfPoints*(*emptyWCPoints, mat*) = *emptyNDCPoints*)
(*transfPoints*(*addWCpoint*(*pn,pnts*), *mat*) = *addNDCpoint*(*transf*(*pn,mat*), *transfPoints*(*pnts,mat*)))
(*transfNDCPoints*(*emptyNDCPoints, mat*) = *emptyNDCPoints*)
(*transfNDCPoints*(*addNDCpoint*(*ndcpoint, ndcpoints*), *mat*)
= *addNDCpoint*(*transfNDC*(*ndcpoint,mat*), *transfNDCPoints*(*ndcpoints,mat*)))
**jbo**

**obj** *LISTOFNDCPOINTS /NDCPOINTS*
**sorts** *ListofWCP ListofNDCP*
**ops** *emptyLofWCP*: → *ListofWCP*
*addWCpoints*: *WCPoints ListofWCP* → *ListofWCP*
*emptyLofNDCP*: → *ListofNDCP*

*addNDCpoints*: *NDCPoints ListofNDCP* → *ListofNDCP*
*transfLofPoints*: *ListofWCP Matrix23* → *ListofNDCP*
*transfLofNDCPoints*: *ListofNDCP Matrix23* → *ListofNDCP*
*LWCP1,LWCP2,LWCP3,LWCP4,LWCP5*: → *ListofWCP*
**vars** *pnts*: *WCPoints*
*lopnts*: *ListofWCP*
*ndcpoints*: *NDCPoints*
*lndcpoints*: *ListofNDCP*
*mat*: *Matrix23*
**eqns** (*transfLofPoints*(*emptyLofWCP*, *mat*) = *emptyLofNDCP*)
(*transfLofPoints*(*addWCpoints*(*pnts*,*lopnts*), *mat*)
= *addNDCpoints*(*transfPoints*(*pnts*,*mat*),*transfLofPoints*(*lopnts*,*mat*)))
(*transfLofNDCPoints*(*emptyLofNDCP*, *mat*) = *emptyLofNDCP*)
(*transfLofNDCPoints*(*addNDCpoints*(*ndcpoints*, *lndcpoints*), *mat*)
= *addNDCpoints*(*transfNDCPoints*(*ndcpoints*, *mat*),*transfLofNDCPoints*(*lndcpoints*, *mat*)))
**jbo**

### 3.9. LISTOFNT

This object defines a list of normalization transformations. This is used later as a component of the GKS-9x state list.

**image**
(*TABLE => LISTOFNT*) / *NAMES MATRIX23*
**sorts** (*tb => ListofNT*)
(*telem => Matrix23*)
(*index => NormTran*)
**ops** (*emptyTb*: → *tb => emptyLNT*)
**endim**

### 3.10. SETOFPOLYLINE

This object defines the SET OF POLYLINE function. This is the only primitive considered in this specification. It is represented by the list of points in NDC which define the geometry of the primitive and the identification, source and logical attributes associated with the primitive.

**obj** *SETOFPOLYLINE /ATTRIBUTES LISTOFNDCPOINTS*
**sorts** *SetofPolyline*
**ops** *mkSetofPolyline*: *ListofNDCP Identification NDCattr Source Logical* → *SetofPolyline*
**jbo**

### 3.11. PICTUREPART and PICTUREPARTSTORE

These objects define picture parts (a list of primitives) and picture part store (a set of named picture parts). Operations on picture part store are defined. These correspond closely to the GKS-9x functions which manipulate the picture part store.

**obj** *PICTUREPART / SETOFPOLYLINE*
**sorts** *PicturePart*
**ops** *emptyPP*: → *PicturePart*
*addPP*: *SetofPolyline PicturePart* → *PicturePart*
*appendPP*: *PicturePart PicturePart* → *PicturePart*
*addnames*: *NameSet PicturePart* → *PicturePart*
*addattr*: *NameSet Matrix23 TransMode Matrix23 TransMode PicturePart* → *PicturePart*
**vars** *pp,pp1*: *PicturePart*
*p*: *SetofPolyline*
*ns,ns1*: *NameSet*

*lndcp*: *ListofNDCP*
*sa*: *Source*
*la*: *Logical*
*ndc*: *NDCattr*
*local*, *global*: *Matrix23*
*trmode*, *trmode1*: *TransMode*

**eqns** (*appendPP*(*emptyPP*,*pp*) = *pp* )
(*appendPP*(*addPP*(*p*,*pp*),*pp1*) = *addPP*(*p*,*appendPP*(*pp*,*pp1*)))
(*addnames*(*ns*,*emptyPP*) = *emptyPP*)
(*addnames*(*ns*,*addPP*(*mkSetofPolyline*(*lndcp*,*identifa*(*ns1*),*ndc*,*sa*,*la*),*pp*))
= *addPP*(*mkSetofPolyline*(*lndcp*,*identifa*((*ns*)*U*(*ns1*)), *ndc*, *sa*, *la*),
*addnames*(*ns*,*pp*)))
(*addattr*(*ns*,*global*,*trmode*,*local*,*trmode1*,*emptyPP*) = *emptyPP*)
(*addattr*(*ns*,*global*,*trmode*,*local*,*trmode1*,*addPP*(*mkSetofPolyline*(
*lndcp*,*identifa*(*ns1*),*ndc*,*sa*,*la*),*pp*))
= *addPP*(*mkSetofPolyline*(*lndcp*,*identifa*((*ns*)*U*(*ns1*)),
*modifyndca*(*global*,*trmode*,*local*,*trmode1*,*ndc*), *sa*,*la*), *addnames*(*ns*,*pp*)))

**jbo**

**obj** *PICTUREPARTSTORE / PICTUREPART*
**sorts** *PPS*
**ops** *emptyPPS*: → *PPS*
*mkPPS*: *PicPartName PicturePart* → *PPS*
*_U_*: *PPS PPS* → *PPS* (*ASSOC COMM ID*:*emptyPPS*)
*beginpicturepart*: *PicPartName PPS* → *PPS*
*deletepicturepart*: *PicPartName PPS* → *PPS*
*renamepicturepart*: *PicPartName PicPartName PPS* → *PPS*
*appendpicturepart*: *PicPartName PicPartName Matrix23 TransMode Matrix23 TransMode*
*NameSet PPS* → *PPS*
*getpicturepart*: *PicPartName PPS* → *PicturePart*
*addtoPP*: *PicPartName SetofPolyline PPS* → *PPS*
*PPisinPPS*: *PicPartName PPS* →*BOOL*
**vars** *pps*,*pps1*,*pps2*: *PPS*
*pp*,*pp1*,*pp2*: *PicturePart*
*pn*,*pn1*,*pn2*: *PicPartName*
*p*: *SetofPolyline*
*local*, *global*: *Matrix23*
*trmode*,*trmode1*: *TransMode*
*ns*:*NameSet*
**eqns** ((*pps*) *U* (*pps*) = *pps*)
(*mkPPS*(*pn*,*emptyPP*) = *emptyPPS*)
(*beginpicturepart*(*pn*,*pps*) = (*mkPPS*(*pn*,*emptyPP*)) *U* (*pps*))

(*deletepicturepart*(*pn*,*emptyPPS*) = *emptyPPS*)
(*deletepicturepart*(*pn*,(*pps1*) *U* (*pps2*)) = (*deletepicturepart*(*pn*,*pps1*)) *U* (*deletepicturepart*(*pn*,*pps2*)))
(*deletepicturepart*(*pn*,*mkPPS*(*pn*,*pp*)) = *emptyPPS*)
(*deletepicturepart*(*pn*,*mkPPS*(*pn1*,*pp*)) = *mkPPS*(*pn1*,*pp*) *IF not*(*pn*==*pn1*))

(*renamepicturepart*(*pn*,*pn1*,*emptyPPS*) = *emptyPPS*)
(*renamepicturepart*(*pn*,*pn1*,*mkPPS*(*pn*,*pp*)) = *mkPPS*(*pn1*,*pp*))
(*renamepicturepart*(*pn*,*pn1*,(*pps1*) *U* (*pps2*))
= (*renamepicturepart*(*pn*,*pn1*,*pps1*)) *U* (*renamepicturepart*(*pn*,*pn1*,*pps2*)))
(*renamepicturepart*(*pn*,*pn1*,(*mkPPS*(*pn*,*pp*))) = *mkPPS*(*pn1*,*pp*))
(*renamepicturepart*(*pn*,*pn1*,(*mkPPS*(*pn2*,*pp*))) = *mkPPS*(*pn2*,*pp*) *IF not*(*pn*==*pn2*))

*(appendpicturepart(pn,pn1,global,trmode,local,trmode1,ns,pps)*
*= (mkPPS(pn1,appendPP(addattr(ns,global,trmode,local,trmode1,getpicturepart(pn,pps)),*
*getpicturepart(pn1,pps)))) U*
*(deletepicturepart(pn1,pps)))*

*(getpicturepart(pn,emptyPPS) = emptyPP)*
*(getpicturepart(pn,(mkPPS(pn,pp))U(pps)) = pp )*
*(getpicturepart(pn,(mkPPS(pn1,pp)) U (pps)) = getpicturepart(pn,pps) IF not (pn==pn1))*

*(PPisinPPS(pn,emptyPPS) = F)*
*(PPisinPPS(pn, (pps1) U (pps2)) = PPisinPPS(pn,pps1) or PPisinPPS(pn,pps2))*
*(PPisinPPS(pn1, mkPPS(pn,pp)) = (pn1==pn))*

*(addtoPP(pn,p,emptyPPS) = emptyPPS)*
*(addtoPP(pn,p,mkPPS(pn,pp)) = mkPPS(pn,addPP(p,pp)))*
*(addtoPP(pn,p,mkPPS(pn1,pp)) = mkPPS(pn1,pp) IF not(pn==pn1))*
*(addtoPP(pn,p,(pps1) U (pps2)) = (addtoPP(pn,p,pps1)) U (addtoPP(pn,p,pps2)))*

**jbo**

### 3.12. NDCPICTURE

This object models the NDC picture and the operations on the NDC picture which modify the values of identification and logical attributes associated with primitives in the NDC picture.

**obj** *NDCPICTURE / PICTUREPART SELECTION*
**sorts** *NDCPicture Position*
**ops** *emptyNDCP*: → *NDCPicture*
*FRONT, BACK*: → *Position*
*addNDCP*: *SetofPolyline NDCPicture* → *NDCPicture*
*appendNDCP*: *NDCPicture NDCPicture* → *NDCPicture*
*setattrNDCP*: *SelectCrit PlInd NDCPicture* → *NDCPicture*
*setattrNDCP*: *SelectCrit asfs NDCPicture* → *NDCPicture*
*setattrNDCP*: *SelectCrit Linetype NDCPicture* → *NDCPicture*
*setattrNDCP*: *SelectCrit Linewidth NDCPicture* → *NDCPicture*
*removenameNDCP*: *NameSet SelectCrit NDCPicture* → *NDCPicture*
*addnameNDCP*: *NameSet SelectCrit NDCPicture* → *NDCPicture*
*deleteNDCP*: *SelectCrit NDCPicture* → *NDCPicture*
*renameNDCP*: *Name Name SelectCrit NDCPicture* → *NDCPicture*
*selectpicture*: *NDCPicture SelectCrit* → *NDCPicture*
*lastp*: *NDCPicture* → *NameSet*
*getfirst*: *NDCPicture SelectCrit* → *NameSet*
*getlast*: *NDCPicture SelectCrit* → *NameSet*
*insert*: *NDCPicture NDCPicture SelectCrit Position* → *NDCPicture*
**vars** *ndcp, ndcp1:NDCPicture*
*ns,ns1:NameSet*
*n,on,nn:Name*
*Selcrit:SelectCrit*
*pp:PicturePart*
*lndcp*: *ListofNDCP*
*pli,pli1*: *PlInd*
*asf,asf1*: *asfs*
*lt,lt1*: *Linetype*
*lw,lw1*: *Linewidth*
*sa*: *Source*
*la*: *Logical*

*ndc*: *NDCattr*
*position*: *Position*

**eqns** (*appendNDCP*(*emptyNDCP*, *ndcp*) = *ndcp*)
(*appendNDCP*(*addNDCP*(*mkSetofPolyline*(*lndcp*, *identifa*(*ns*), *ndc*,
*sourcea*(*asf*,*pli*),*la*),*ndcp*), *ndcp1*)
= *addNDCP*(*mkSetofPolyline*(*lndcp*, *identifa*(*ns*), *ndc*,
*sourcea*(*asf*,*pli*),*la*), *appendNDCP*(*ndcp*,*ndcp1*)))

(*setattrNDCP*(*Selcrit*,*pli1*,*emptyNDCP*) = *emptyNDCP*)
(*setattrNDCP*(*Selcrit*,*pli1*,*addNDCP*(*mkSetofPolyline*(*lndcp*, *identifa*(*ns*), *ndc*,
*sourcea*(*asf*,*pli*),*la*),*ndcp*))
= *addNDCP*(*mkSetofPolyline*(*lndcp*,*identifa*(*ns*), *ndc*,
*sourcea*(*asf*,*pli1*),*la*),*setattrNDCP*(*Selcrit*,*pli1*,*ndcp*))
*IF* (*satisfy*(*ns*,*Selcrit*)==*T*))
(*setattrNDCP*(*Selcrit*,*pli1*,*addNDCP*(*mkSetofPolyline*(*lndcp*, *identifa*(*ns*), *ndc*,
*sourcea*(*asf*,*pli*),*la*), *ndcp*))
= *addNDCP*(*mkSetofPolyline*(*lndcp*,*identifa*(*ns*),*ndc*,
*sourcea*(*asf*,*pli*),*la*),*setattrNDCP*(*Selcrit*,*pli1*,*ndcp*))
*IF* (*satisfy*(*ns*,*Selcrit*)==*F*))
(*setattrNDCP*(*Selcrit*,*asf1*,*emptyNDCP*) = *emptyNDCP*)
(*setattrNDCP*(*Selcrit*,*asf1*,*addNDCP*(*mkSetofPolyline*(*lndcp*,*identifa*(*ns*), *ndc*,
*sourcea*(*asf*,*pli*),*la*),*ndcp*))
= *addNDCP*(*mkSetofPolyline*(*lndcp*,*identifa*(*ns*), *ndc*,
*sourcea*(*asf1*,*pli*),*la*), *setattrNDCP*(*Selcrit*,*asf1*,*ndcp*))
*IF* (*satisfy*(*ns*,*Selcrit*)==*T*))
(*setattrNDCP*(*Selcrit*,*asf1*,*addNDCP*(*mkSetofPolyline*(*lndcp*,*identifa*(*ns*), *ndc*,
*sourcea*(*asf*,*pli*),*la*),*ndcp*))
= *addNDCP*(*mkSetofPolyline*(*lndcp*,*identifa*(*ns*), *ndc*,
*sourcea*(*asf*,*pli*),*la*), *setattrNDCP*(*Selcrit*,*asf1*,*ndcp*))
*IF* (*satisfy*(*ns*,*Selcrit*)==*F*))
(*setattrNDCP*(*Selcrit*,*lt1*,*emptyNDCP*) = *emptyNDCP*)
(*setattrNDCP*(*Selcrit*,*lt1*,*addNDCP*(*mkSetofPolyline*(*lndcp*,*identifa*(*ns*), *ndc*, *sa*,
*logicala*(*lt*,*lw*)),*ndcp*))
= *addNDCP*(*mkSetofPolyline*(*lndcp*,*identifa*(*ns*), *ndc*, *sa*,
*logicala*(*lt1*,*lw*)),*setattrNDCP*(*Selcrit*,*lt1*,*ndcp*))
*IF* (*satisfy*(*ns*,*Selcrit*)==*T*))
(*setattrNDCP*(*Selcrit*,*lt1*,*addNDCP*(*mkSetofPolyline*(*lndcp*,*identifa*(*ns*), *ndc*, *sa*,
*logicala*(*lt*,*lw*)), *ndcp*))
= *addNDCP*(*mkSetofPolyline*(*lndcp*,*identifa*(*ns*), *ndc*, *sa*,
*logicala*(*lt*,*lw*)), *setattrNDCP*(*Selcrit*,*lt1*,*ndcp*))
*IF* (*satisfy*(*ns*,*Selcrit*)==*F*))
(*setattrNDCP*(*Selcrit*,*lw1*,*emptyNDCP*) = *emptyNDCP*)
(*setattrNDCP*(*Selcrit*,*lw1*,*addNDCP*(*mkSetofPolyline*(*lndcp*,*identifa*(*ns*), *ndc*, *sa*,
*logicala*(*lt*,*lw*)),*ndcp*))
= *addNDCP*(*mkSetofPolyline*(*lndcp*,*identifa*(*ns*), *ndc*, *sa*,
*logicala*(*lt*,*lw1*)),*setattrNDCP*(*Selcrit*,*lw1*,*ndcp*))
*IF* (*satisfy*(*ns*,*Selcrit*)==*T*))
(*setattrNDCP*(*Selcrit*,*lw1*,*addNDCP*(*mkSetofPolyline*(*lndcp*, *identifa*(*ns*), *ndc*, *sa*,
*logicala*(*lt*,*lw*)),*ndcp*))
= *addNDCP*(*mkSetofPolyline*(*lndcp*,*identifa*(*ns*), *ndc*, *sa*,
*logicala*(*lt*,*lw*)), *setattrNDCP*(*Selcrit*,*lw1*,*ndcp*))
*IF* (*satisfy*(*ns*,*Selcrit*)==*F*))

(*addnameNDCP*(*ns1*,*Selcrit*,*emptyNDCP*) = *emptyNDCP*)

*(addnameNDCP(ns1,Selcrit,addNDCP(mkSetofPolyline(lndcp, identifa(ns), ndc, sa,la),ndcp))*
*= addNDCP(mkSetofPolyline(lndcp,identifa((ns)U(ns1)), ndc, sa,la),*
*addnameNDCP(ns1,Selcrit,ndcp))*
*IF (satisfy(ns,Selcrit)==T))*
*(addnameNDCP(ns1,Selcrit,addNDCP(mkSetofPolyline(lndcp, identifa(ns), ndc, sa,la),ndcp))*
*= addNDCP(mkSetofPolyline(lndcp,identifa(ns), ndc, sa,la),*
*addnameNDCP(ns1,Selcrit,ndcp))*
*IF not (satisfy(ns,Selcrit)==T))*

*(removenameNDCP(ns1,Selcrit,emptyNDCP) = emptyNDCP)*
*(removenameNDCP(ns1,Selcrit,addNDCP(mkSetofPolyline(lndcp, identifa(ns), ndc, sa,la),ndcp))*
*= addNDCP(mkSetofPolyline(lndcp, identifa(subtractset(ns1,ns)), ndc, sa, la),*
*removenameNDCP(ns1,Selcrit,ndcp))*
*IF (satisfy(ns,Selcrit)==T))*
*(removenameNDCP(ns1,Selcrit,addNDCP(mkSetofPolyline(lndcp, identifa(ns), ndc, sa, la), ndcp))*
*= addNDCP(mkSetofPolyline(lndcp,identifa(ns), ndc, sa,la),*
*removenameNDCP(ns1,Selcrit,ndcp))*
*IF not (satisfy(ns,Selcrit)==T))*

*(deleteNDCP(Selcrit,emptyNDCP) = emptyNDCP)*
*(deleteNDCP(Selcrit,addNDCP(mkSetofPolyline(lndcp, identifa(ns), ndc, sa,la), ndcp))*
*= deleteNDCP(Selcrit,ndcp) IF (satisfy(ns,Selcrit)==T))*
*(deleteNDCP(Selcrit,addNDCP(mkSetofPolyline(lndcp, identifa(ns), ndc, sa,la), ndcp))*
*= addNDCP(mkSetofPolyline(lndcp,identifa(ns), ndc, sa, la), deleteNDCP(Selcrit,ndcp))*
*IF (satisfy(ns,Selcrit)==F))*

*(renameNDCP(on,nn,Selcrit,emptyNDCP) = emptyNDCP)*
*(renameNDCP(on,nn,Selcrit,addNDCP(mkSetofPolyline(lndcp, identifa(ns), ndc, sa,la), ndcp))*
*= addNDCP(mkSetofPolyline(lndcp, identifa(renamessn(on, nn, ns)), ndc, sa, la),*
*renameNDCP(on,nn,Selcrit,ndcp)) IF satisfy(ns,Selcrit))*
*(renameNDCP(on,nn,Selcrit,addNDCP(mkSetofPolyline(lndcp, identifa(ns), ndc, sa, la), ndcp))*
*= addNDCP(mkSetofPolyline(lndcp, identifa(ns), ndc, sa, la),*
*renameNDCP(on,nn,Selcrit,ndcp)) IF not satisfy(ns,Selcrit))*

*(selectpicture(emptyNDCP,Selcrit) = emptyNDCP)*
*(selectpicture(addNDCP(mkSetofPolyline(lndcp,identifa(ns), ndc, sa, la), ndcp), Selcrit)*
*= addNDCP(mkSetofPolyline(lndcp,identifa(ns), ndc, sa,la), selectpicture(ndcp,Selcrit))*
*IF (satisfy(ns,Selcrit)==T))*
*(selectpicture(addNDCP(mkSetofPolyline(lndcp,identifa(ns), ndc, sa,la), ndcp), Selcrit)*
*= selectpicture(ndcp, Selcrit) IF not (satisfy(ns,Selcrit)==T))*

*(getfirst(addNDCP(mkSetofPolyline(lndcp,identifa(ns), ndc, sa, la), ndcp), Selcrit)*
*= ns IF (satisfy(ns,Selcrit)==T))*
*(getfirst(addNDCP(mkSetofPolyline(lndcp,identifa(ns), ndc, sa, la), ndcp), Selcrit)*
*= getfirst(ndcp, Selcrit) IF not (satisfy(ns,Selcrit)==T))*

*(getlast(addNDCP(mkSetofPolyline(lndcp,identifa(ns), ndc, sa, la), ndcp), Selcrit)*
*= ns IF (satisfy(ns,Selcrit)==T) and*
*(lastp(selectpicture(addNDCP(mkSetofPolyline(lndcp, identifa(ns), ndc, sa, la), ndcp), Selcrit))==ns))*
*(getlast(addNDCP(mkSetofPolyline(lndcp,identifa(ns), ndc, sa, la), ndcp), Selcrit)*
*= getlast(ndcp, Selcrit)*
*IF not (satisfy(ns,Selcrit)==T) or*
*not (lastp(selectpicture(addNDCP(mkSetofPolyline(lndcp, identifa(ns),*
*ndc, sa, la), ndcp), Selcrit))==ns))*

*(lastp(addNDCP(mkSetofPolyline(lndcp,identifa(ns),ndc,sa,la), ndcp)) = ns IF (ndcp==emptyNDCP))*
*(lastp(addNDCP(mkSetofPolyline(lndcp,identifa(ns),ndc,sa,la), ndcp)) = lastp(ndcp)*
*IF not (ndcp==emptyNDCP))*

*(insert(emptyNDCP, ndcp, Selcrit, position) = ndcp)*
*(insert(ndcp1, addNDCP(mkSetofPolyline(lndcp, identifa(ns), ndc, sourcea(asf,pli),la),ndcp),*
*Selcrit, FRONT)*
*= appendNDCP(ndcp1, addNDCP(mkSetofPolyline(lndcp, identifa(ns),*
*ndc, sourcea(asf,pli),la), ndcp))*
*IF (getfirst(addNDCP(mkSetofPolyline(lndcp, identifa(ns), ndc,*
*sourcea(asf,pli),la),ndcp),Selcrit)==ns))*
*(insert(ndcp1, addNDCP(mkSetofPolyline(lndcp, identifa(ns), ndc, sourcea(asf,pli),la),ndcp),*
*Selcrit, FRONT)*
*= addNDCP(mkSetofPolyline(lndcp,identifa(ns),*
*ndc,sourcea(asf,pli),la),insert(ndcp1,ndcp,Selcrit,FRONT))*
*IF not (getfirst(addNDCP(mkSetofPolyline(lndcp,identifa(ns),ndc,*
*sourcea(asf,pli),la),ndcp),Selcrit)==ns))*
*(insert(ndcp1, addNDCP(mkSetofPolyline(lndcp, identifa(ns), ndc, sourcea(asf,pli),la),ndcp),*
*Selcrit, BACK)*
*= addNDCP(mkSetofPolyline(lndcp,identifa(ns),*
*ndc,sourcea(asf,pli),la), appendNDCP(ndcp1, ndcp))*
*IF (getlast(addNDCP(mkSetofPolyline(lndcp, identifa(ns), ndc,*
*sourcea(asf,pli),la),ndcp),Selcrit)==ns))*
*(insert(ndcp1, addNDCP(mkSetofPolyline(lndcp, identifa(ns), ndc, sourcea(asf,pli),la),ndcp),*
*Selcrit, BACK)*
*= addNDCP(mkSetofPolyline(lndcp,identifa(ns),*
*ndc,sourcea(asf,pli),la), insert(ndcp1, ndcp, Selcrit, BACK))*
*IF not (getlast(addNDCP(mkSetofPolyline(lndcp, identifa(ns), ndc,*
*sourcea(asf,pli),la),ndcp),Selcrit)==ns))*

**jbo**

### 3.13. PPNDCPCONVERTER

This object describes the operations which copy a picture part into the NDC picture and copy a portion of the NDC picture into a picture part.

**obj** *PPNDCPCONVERTER / NDCPICTURE*
**sorts**
**ops** *copyPPNDCP: PicturePart SelectCrit Matrix23 TransMode*
*Matrix23 TransMode NameSet NDCPicture → NDCPicture*
*copyNDCPPP: NDCPicture → PicturePart*
**vars** *sply: SetofPolyline*
*ndcp:NDCPicture*
*ns,ns1:NameSet*
*n,on,nn:Name*
*Selcrit:SelectCrit*
*pp:PicturePart*
*lndcp: ListofNDCP*
*pli,pli1: PlInd*
*asf,asf1: asfs*
*lt,lt1: Linetype*
*lw,lw1: Linewidth*
*sa: Source*
*la: Logical*
*ndc: NDCattr*

*global,local: Matrix23*
*trmode,trmode1: TransMode*
**eqns** *(copyPPNDCP(emptyPP,Selcrit,global,trmode,local,trmode1,ns,ndcp)=ndcp)*
*(copyPPNDCP(addPP(mkSetofPolyline(lndcp, identifa(ns1), ndc, sa,la), pp),*
*Selcrit, global, trmode,local,trmode1, ns, ndcp)*
*= addNDCP(mkSetofPolyline(lndcp, identifa((ns1)U(ns)),*
*modifyndca(global, trmode,local,trmode1, ndc), sa,la),*
*copyPPNDCP(pp,Selcrit,global,trmode,local,trmode1,ns,ndcp))*
*IF (satisfy(ns1,Selcrit)==T))*
*(copyPPNDCP(addPP(mkSetofPolyline(lndcp, identifa(ns1), ndc, sa, la), pp),*
*Selcrit, global, trmode, local, trmode1, ns, ndcp)*
*= copyPPNDCP(pp,Selcrit,global,trmode,local,trmode1,ns,ndcp)*
*IF (satisfy(ns1,Selcrit)==F))*
*(copyNDCPPP(emptyNDCP) = emptyPP)*
*(copyNDCPPP(addNDCP(sply,ndcp)) = addPP(sply,copyNDCPPP(ndcp)))*
**jbo**

### 3.14. VIEW and LISTOFVW

*VIEW* describes views. A view consists of a selection criterion and a transformation matrix. *LISTOFVW* describes a list of view transformations and is used in describing the workstation state list.

**obj** *VIEW /SELECTION MATRIX23*
**sorts** *View ViewInd*
**ops** *mkView: SelectCrit Matrix23 Matrix23 → View*
*sets: SelectCrit View → View*
*seta: Matrix23 Matrix23 View → View*
*VWN0: → ViewInd*
*VWN1: → ViewInd*
*VWN2: → ViewInd*
*VWN3: → ViewInd*
*SELCRIT: → SelectCrit*
*SELCRIT1: → SelectCrit*
*VWT: → Matrix23*
*VWT1: → Matrix23*
**vars** *viewsel, viewsel1: SelectCrit*
*omatrix, mmatrix, omatrix1, mmatrix1: Matrix23*
**eqns** *(sets(viewsel1, mkView(viewsel, omatrix, mmatrix))*
*= mkView(viewsel1, omatrix, mmatrix))*
*(seta(omatrix1, mmatrix1, mkView(viewsel, omatrix, mmatrix))*
*= mkView(viewsel, omatrix1, mmatrix1))*
**jbo**

**image**
*(TABLE => LISTOFVW) / VIEW*
**sorts** *(tb => ListofVW)*
*(telem => View)*
*(index => ViewInd)*
**ops** *(emptyTb: → tb => emptyLVW)*
**endim**

### 3.15. PLBUNDLETABLE

This object describes polyline bundle tables.

**image**
(*TABLE => PLBUNDLETABLE*) */ATTRIBUTES*
**sorts** (*tb => PlBunTab*)
(*telem => PlBun*)
(*index => PlInd*)
**ops** (*emptyTb*: → *tb => emptyBt*)
**endim**

### 3.16. WSL

This object describes the Workstation State List.

**obj** *WSL / SELECTION PLBUNDLETABLE LISTOFVW*
**sorts** *wsl VisEffSt*
**ops** *MATRIX23*: → *Matrix23*
*ALLOW, SUPPRESS*: → *VisEffSt*
*mkWSL*: *PlBunTab ListofVW Matrix23 VisEffSt SelectCrit SelectCrit SelectCrit* →*wsl*
**jbo**

### 3.17. LDCPOINT, LDCPOINTS, LISTOFLDCPOINTS

These objects describe points and lists of points in LDC space. The transformation from lists of points in NDC to lists of points in LDC is also defined.

**obj** *LDCPOINT /NDCPOINT*
**sorts** *LDCPoint*
**ops** *mkLDCPoint*: *int int* → *LDCPoint*
*transfMapping*: *NDCPoint Matrix23* → *LDCPoint*
*transfVW*: *NDCPoint Matrix23 Matrix23* → *LDCPoint*
**vars** *x,y,a,b,c,d,e,f*: *int*
*a1,b1,c1,d1,e1 f1*: *int*
**eqns** (*transfMapping*(*mkNDCPoint*(*x,y*),*mktransfcor*(*a,b,c,d,e,f*))
= *mkLDCPoint*((*x*a*)+(*y*b*)+*c*, (*x*d*)+(*y*e*)+*f*))
(*transfVW*(*mkNDCPoint*(*x,y*), *mktransfcor*(*a,b,c,d,e,f*), *mktransfcor*(*a1,b1,c1,d1,e1 f1*))
= *transfMapping*(*transfNDC*(*mkNDCPoint*(*x,y*), *mktransfcor*(*a,b,c,d,e,f*)),
*mktransfcor*(*a1,b1,c1,d1,e1 f1*)))
**jbo**

**obj** *LDCPOINTS /NDCPOINTS LDCPOINT*
**sorts** *LDCPoints*
**ops** *emptyLDCPoints*: → *LDCPoints*
*addLDCpoint*: *LDCPoint LDCPoints* → *LDCPoints*
*transfVWPoints*: *NDCPoints Matrix23 Matrix23* → *LDCPoints*
**vars** *pn*: *NDCPoint*
*pnts*: *NDCPoints*
*mat, mat1*: *Matrix23*
**eqns** (*transfVWPoints*(*emptyNDCPoints, mat, mat1*) = *emptyLDCPoints*)
(*transfVWPoints*(*addNDCpoint*(*pn,pnts*), *mat, mat1*)
= *addLDCpoint*(*transfVW*(*pn,mat, mat1*), *transfVWPoints*(*pnts,mat, mat1*)))
**jbo**

**obj** *LISTOFLDCPOINTS /LDCPOINTS LISTOFNDCPOINTS*
**sorts** *ListofLDCP*

**ops** *emptyLofLDCP*: → *ListofLDCP*
*addLDCpoints*: *LDCPoints ListofLDCP* → *ListofLDCP*
*transfVWLofPoints*: *ListofNDCP Matrix23 Matrix23* → *ListofLDCP*
**vars** *pnts*: *NDCPoints*
*lopnts*: *ListofNDCP*
*mat, mat1*: *Matrix23*
**eqns** (*transfVWLofPoints*(*emptyLofNDCP, mat, mat1*) = *emptyLofLDCP*)
(*transfVWLofPoints*(*addNDCpoints*(*pnts,lopnts*), *mat, mat1*)
= *addLDCpoints*(*transfVWPoints*(*pnts,mat, mat1*), *transfVWLofPoints*(*lopnts,mat, mat1*)))
**jbo**

### 3.18. SETOFLOGPOLYLINE, LOGICALPICTURE

These objects describe the set of polyline primitive in the logical picture and the logical picture itself, together with the operations to transform the NDC picture to the logical picture.

**obj** *SETOFLOGPOLYLINE /ATTRIBUTES LISTOFLDCPOINTS*
**sorts** *SofLOGPolyline*
**ops** *mkSofLOGPolyline*: *ListofLDCP Identification Logical Highl Det* → *SofLOGPolyline*
**jbo**

**obj** *LOGICALPICTURE /SETOFLOGPOLYLINE PPNDCPCONVERTER WSL*
**sorts** *LOGPicture*
**ops** *emptyLOGP*: → *LOGPicture*
*addLOGP*: *SofLOGPolyline LOGPicture* → *LOGPicture*
*appendLOGP*: *LOGPicture LOGPicture* → *LOGPicture*
*applyvt*: *NDCPicture View PlBunTab SelectCrit SelectCrit* → *LOGPicture*
*applyviews*: *NDCPicture ListofVW PlBunTab SelectCrit SelectCrit* → *LOGPicture*
**vars** *i*: *ViewInd*
*Selcrit,sh,sd*: *SelectCrit*
*omatrix, mmatrix*: *Matrix23*
*vw*: *View*
*lvw1,lvw2*: *ListofVW*
*npcp,npcp1*: *LOGPicture*
*ndcp*:*NDCPicture*
*ns*:*NameSet*
*lndcp*: *ListofNDCP*
*pp*: *SofLOGPolyline*
*sa*: *Source*
*la*: *Logical*
*local, global*: *Matrix23*
*pli*:*PlInd*
*pbt*: *PlBunTab*
*a,a1*: *ASF*
*asf*: *asfs*
*lt,blt*: *Linetype*
*lw,blw*: *Linewidth*
**eqns** (*appendLOGP*(*emptyLOGP, npcp*) = *npcp*)
(*appendLOGP*(*addLOGP*(*pp,npcp*),*npcp1*) = *addLOGP*(*pp,appendLOGP*(*npcp,npcp1*)))

(*applyvt*(*emptyNDCP, vw ,pbt,sh,sd*) = *emptyLOGP*)
(*applyvt*(*addNDCP*(*mkSetofPolyline*(*lndcp,identifa*(*ns*), *ndca*(*local, global*),
*sourcea*(*asf,pli*),*la*),*ndcp*), *mkView*(*Selcrit,omatrix,mmatrix*),*pbt,sh,sd*)
= *addLOGP*(*mkSofLOGPolyline*(*transfVWLofPoints*(
*transfLofNDCPoints*(*transfLofNDCPoints*(*lndcp, local*), *global*), *omatrix,mmatrix*),

*identifa(ns),*
*selectlogattr(asf,la,pbt [ pli ]),highlt(ns,sh),dect(ns,sd)),*
*applyvt(ndcp,mkView(Selcrit,omatrix,mmatrix),pbt,sh,sd))*
*IF satisfy(ns,Selcrit))*
*(applyvt(addNDCP(mkSetofPolyline(lndcp,identifa(ns), ndca(local, global),*
*sourcea(asf,pli),la), ndcp), mkView(Selcrit,omatrix,mmatrix),pbt,sh,sd)*
*= applyvt(ndcp,mkView(Selcrit,omatrix,mmatrix),pbt,sh,sd)*
*IF not (satisfy(ns,Selcrit)))*
*(applyviews(emptyNDCP, lvw1, pbt, sh, sd) = emptyLOGP)*
*(applyviews(ndcp, (lvw1)U(lvw2), pbt, sh, sd)*
*= appendLOGP(applyviews(ndcp, lvw1, pbt, sh, sd), applyviews(ndcp, lvw2, pbt, sh, sd))*
*IF not (ndcp==emptyNDCP))*
*(applyviews(ndcp, (i % vw), pbt, sh, sd) = applyvt(ndcp,vw,pbt,sh,sd))*

**jbo**

### 3.19. DCPOINT, DCPOINTS, LISTOFDCPOINTS, SETOFRFPOLYLINE

These objects describe points, lists of points and the set of polyline primitive in the reified picture.

**obj** *DCPOINT / LDCPOINT MATRIX23*
**sorts** *DCPoint*
**ops** *mkDCPoint: int int → DCPoint*
*transfWS: LDCPoint Matrix23 → DCPoint*
**vars** *x,y,a,b,c,d,e,f: int*
**eqns** *(transfWS(mkLDCPoint(x,y),mktransfcor(a,b,c,d,e,f))*
*= mkDCPoint((x*a)+(y*b)+c, (x*d)+(y*e)+f))*
**jbo**

**obj** *DCPOINTS /LDCPOINTS DCPOINT*
**sorts** *DCPoints*
**ops** *emptyDCPoints: → DCPoints*
*addDCpoint: DCPoint DCPoints → DCPoints*
*transfWSPoints: LDCPoints Matrix23 → DCPoints*
**vars** *pn: LDCPoint*
*pnts: LDCPoints*
*mat: Matrix23*
**eqns** *(transfWSPoints(emptyLDCPoints, mat) = emptyDCPoints)*
*(transfWSPoints(addLDCpoint(pn,pnts), mat)*
*= addDCpoint(transfWS(pn,mat), transfWSPoints(pnts,mat)))*
**jbo**

**obj** *LISTOFDCPOINTS /DCPOINTS LISTOFLDCPOINTS*
**sorts** *ListofDCP*
**ops** *emptyLofDCP: → ListofDCP*
*addDCpoints: DCPoints ListofDCP → ListofDCP*
*transfWSLofPoints: ListofLDCP Matrix23 → ListofDCP*
**vars** *pnts: LDCPoints*
*lopnts: ListofLDCP*
*mat: Matrix23*
**eqns** *(transfWSLofPoints(emptyLofLDCP, mat) = emptyLofDCP)*
*(transfWSLofPoints(addLDCpoints(pnts,lopnts), mat)*
*= addDCpoints(transfWSPoints(pnts,mat),transfWSLofPoints(lopnts,mat)))*
**jbo**

**obj** *SETOFRFPOLYLINE /ATTRIBUTES LISTOFDCPOINTS*
**sorts** *SofRFPolyline*
**ops** *mkSofRFPolyline: ListofDCP Identification Logical Highl Det → SofRFPolyline*
**jbo**

### 3.20. WINDOWSANDVIEWPORTS

This object describes the workstation window and viewport and a simplified specification of the workstation transformation (the isotropic nature of this transformation is not described here).

**obj** *WINDOWSANDVIEWPORTS /NDCPOINT DCPOINT*
**sorts** *WCWin NDCVp LDCWin DCVp*
**ops** *mkWCWin: WCPoint WCPoint → WCWin*
*mkNDCVp: NDCPoint NDCPoint → NDCVp*
*mkLDCWin: NDCPoint NDCPoint → LDCWin*
*mkDCVp: DCPoint DCPoint → DCVp*
*gettransfNT: int int int int int int int int → Matrix23*
*gettransfWS: int int int int int int int int → Matrix23*
*WSW: → LDCWin*
*WSV: → DCVp*
**vars** *xw1,yw1,xw2,yw2,xv1,yv1,xv2,yv2: int*
**eqns** *(gettransfNT(xw1,yw1,xw2,yw2,xv1,yv1,xv2,yv2)*
*= mktransfcor(((xv2-xv1) div (xw2-xw1)),ps 0,xv1-xw1,ps 0,*
*((yv2-yv1) div (yw2-yw1)),yv1-yw1))*
*(gettransfWS(xw1,yw1,xw2,yw2,xv1,yv1,xv2,yv2)*
*= mktransfcor(((xv2-xv1) div (xw2-xw1)),ps 0,xv1-xw1,ps 0,*
*((yv2-yv1) div (yw2-yw1)),yv1-yw1))*
**jbo**

### 3.21. GSL

This object describes the GKS State List.

**obj** *GSL / NAMESETS LISTOFNT ATTRIBUTES WINDOWSANDVIEWPORTS*
**sorts** *gsl*
**ops** *mkGSL: NameSet NameSet Matrix23 Matrix23 PlInd asfs*
*Linetype Linewidth NormTran ListofNT PicPartName → gsl*
*addnt: NormTran WCWin NDCVp gsl → gsl*
*getPicPartName: gsl → PicPartName*
**vars** *xw1,yw1,xw2,yw2,xv1,yv1,xv2,yv2: int*
*opw,ns: NameSet*
*pli: PlInd*
*asf: asfs*
*local, global: Matrix23*
*ltype: Linetype*
*lw: Linewidth*
*ntn,ntn1: NormTran*
*ntl: ListofNT*
*pn: PicPartName*
**eqns** *(addnt (ntn1,mkWCWin(mkWCPoint(xw1,yw1),mkWCPoint(xw2,yw2)),*
*mkNDCVp(mkNDCPoint(xv1,yv1),mkNDCPoint(xv2,yv2)),*
*mkGSL(opw,ns,local,global,pli,asf,ltype,lw,ntn,ntl,pn))*
*= mkGSL(opw,ns,local,global,pli,asf,ltype,lw,ntn,*
*(ntl + (ntn1 % (gettransfNT(xw1,yw1,xw2,yw2,xv1,yv1,xv2,yv2)))),pn) IF not (ntn1==NTN0))*
*(getPicPartName(mkGSL(opw,ns,local,global,pli,asf,ltype,lw,ntn,ntl,pn)) = pn)*
**jbo**

### 3.22. REIFIEDPICTURE

This object describes the reified picture and the transformation from the logical picture to the reified picture.

**obj** *REIFIEDPICTURE /SETOFRFPOLYLINE LOGICALPICTURE*
**sorts** *REIFPicture*
**ops** *emptyRFP*: → *REIFPicture*
*addRFP*: *SofRFPolyline REIFPicture* → *REIFPicture*
*logptoreifp*: *LOGPicture Matrix23 REIFPicture* → *REIFPicture*
**vars** *wst*: *Matrix23*
*soflogp*: *SofLOGPolyline*
*lnpcp*: *ListofLDCP*
*logp*: *LOGPicture*
*ia*: *Identification*
*la*: *Logical*
*h*: *Highl*
*d*: *Det*
**eqns** (*logptoreifp*(*emptyLOGP*,*wst*) = *emptyRFP*)
(*logptoreifp*(*addLOGP*(*mkSofLOGPolyline*(*lnpcp*,*ia*,*la*,*h*,*d*),*logp*),*wst*)
= *addRFP*(*mkSofRFPolyline*(*transfWSLofPoints*(*lnpcp*,*wst*),*ia*,*la*,*h*,*d*),
*logptoreifp*(*logp*,*wst*)))
**jbo**

### 3.23. WORKSTATIONS

This object describes workstations and the operations on them.

**obj** *WORKSTATIONS / REIFIEDPICTURE WINDOWSANDVIEWPORTS*
**sorts** *Workstations*
*OPS mkWSs*: *WsId REIFPicture wsl* →*Workstations*
*emptyWorkstations*: →*Workstations*
*_U_*: *Workstations Workstations* →*Workstations* (*ASSOC COMM ID*:*emptyWorkstations*)
*OpenWS*: *WsId Workstations* →*Workstations*
*updatereifp*: *NDCPicture VisEffSt SelectCrit SelectCrit SelectCrit*
*PlBunTab ListofVW Matrix23 REIFPicture* → *REIFPicture*

*DisplayNDCPWorkstations*: *NDCPicture Workstations* →*Workstations*
*DisplayNDCPWs*: *WsId NDCPicture Workstations* →*Workstations*

*deleteWs*: *WsId Workstations* →*Workstations*
*setsdis*: *WsId NDCPicture SelectCrit Workstations* → *Workstations*
*setsh*: *WsId NDCPicture SelectCrit Workstations* → *Workstations*
*setsd*: *WsId NDCPicture SelectCrit Workstations* → *Workstations*
*setVeffects*: *WsId VisEffSt NDCPicture Workstations* → *Workstations*
*setrep*: *WsId PlInd PlBun NDCPicture Workstations* → *Workstations*
*setviewrep*: *WsId ViewInd Matrix23 Matrix23 NDCPicture Workstations* → *Workstations*
*setviewsel*: *WsId ViewInd SelectCrit NDCPicture Workstations* → *Workstations*
*setwwandvw*: *WsId LDCWin DCVp NDCPicture Workstations* → *Workstations*
**vars** *wkstns*,*wkstns1*,*wkstns2*:*Workstations*
*wsid*,*wsid1*:*WsId*
*ns*,*ns1*,*acw*,*awi* ,*vs*,*hs*,*ds*:*NameSet*
*ndcp*:*NDCPicture*
*reifp*:*REIFPicture*
*wksl*: *wsl*
*Selcrit*,*s*,*sdis*,*sdis1*,*sh*,*sh1*,*sd*,*sd1*,*viewsel*,*viewsel1*:*SelectCrit*
*pp*:*PicturePart*

```
    vef,vef1:VisEffSt
    pbt: PlBunTab
    bl: PlBun
    pli: PlInd
    vwl: ListofVW
    vwn: ViewInd
    view: View
    wst, omatrix, mmatrix, omatrix1, mmatrix1: Matrix23
    wsww: LDCWin
    wsvw: DCVp
    xw1,yw1,xw2,yw2,xv1,yv1,xv2,yv2: int
    h:Highl
    d:Det
eqns ((wkstns)U(wkstns) = wkstns )

    (OpenWS(wsid,wkstns)
        =(mkWSs(wsid,emptyRFP,
          mkWSL(
            (PLI0 % mkBundle(DASHED,THICK)) U
            (PLI1 % mkBundle(DOTTED,MEDIUM)),
            (VWN0 % mkView(SELECTALL,mktransfcor(ps 1,ps 0,ps 0,ps 0,ps 1,ps 0),
              mktransfcor(ps 1,ps 0,ps 0,ps 0,ps 1,ps 0))) U
            (VWN1 % mkView(REJECTALL,
              mktransfcor(ps 1,ps 0,ps 0,ps 0,ps 1,ps 0),
              mktransfcor(ps 1,ps 0,ps 0,ps 0,ps 1,ps 0))),
              mktransfcor(ps 1,ps 0,ps 0,ps 0,ps 1,ps 0),ALLOW,
              SELECTALL,REJECTALL,REJECTALL))) U (wkstns))

    (updatereifp(ndcp, SUPPRESS, sdis, sh, sd, pbt, vwl, wst, reifp)= reifp)
    (updatereifp(ndcp, ALLOW, sdis, sh, sd, pbt, vwl, wst, reifp)
    = logptoreifp(applyviews(selectpicture(ndcp, sdis),
    vwl, pbt, sh, sd), wst))

    (DisplayNDCPWorkstations(ndcp,emptyWorkstations) = emptyWorkstations)

    (DisplayNDCPWorkstations(ndcp,(wkstns1) U (wkstns2))
    =(DisplayNDCPWorkstations(ndcp,wkstns1)) U
    (DisplayNDCPWorkstations(ndcp,wkstns2)))
    (DisplayNDCPWorkstations(ndcp, mkWSs(wsid, reifp,
    mkWSL(pbt, vwl, wst, vef, sdis, sh, sd)))
        = mkWSs(wsid, updatereifp(ndcp,vefs,dis,sh,sd,pbt,vwl,wst,reifp),mkWSL(pbt,vwl,wst,vef,sdis,sh,sd)))
    (DisplayNDCPWs(wsid,ndcp,emptyWorkstations) = emptyWorkstations)
    (DisplayNDCPWs(wsid,ndcp,(wkstns1) U (wkstns2))
        = (DisplayNDCPWs(wsid,ndcp,wkstns1)) U
    (DisplayNDCPWs(wsid,ndcp,wkstns2)))
    (DisplayNDCPWs(wsid,ndcp,
        mkWSs(wsid,reifp,mkWSL(pbt,vwl,wst,vef,sdis,sh,sd)))
        = mkWSs(wsid,updatereifp(ndcp,vef,sdis,sh,sd,pbt,vwl,wst,reifp),mkWSL(pbt,vwl,wst,vef,sdis,sh,sd)))
    (DisplayNDCPWs(wsid,ndcp,mkWSs(wsid1,reifp,wksl))
        = mkWSs(wsid1, reifp, wksl) IF not (wsid == wsid1))

    (deleteWs(wsid, emptyWorkstations) = emptyWorkstations)
    (deleteWs(wsid, (wkstns1) U (wkstns2))
        = (deleteWs(wsid, wkstns1)) U (deleteWs(wsid, wkstns2)))
```

*(deleteWs(wsid, mkWSs(wsid, reifp, wksl)) = emptyWorkstations)*
*(deleteWs(wsid, mkWSs(wsid1, reifp, wksl))*
*= mkWSs(wsid1, reifp, wksl) IF not (wsid == wsid1))*

*(setsdis(wsid,ndcp,sdis, emptyWorkstations) = emptyWorkstations)*
*(setsdis(wsid,ndcp, sdis, (wkstns1) U (wkstns2))*
*= (setsdis(wsid,ndcp,sdis,wkstns1)) U (setsdis(wsid,ndcp,sdis,wkstns2)))*
*(setsdis(wsid, ndcp, sdis1,mkWSs(wsid, reifp, mkWSL(pbt,vwl,wst,vef,sdis,sh,sd )))*
*= mkWSs(wsid, updatereifp(ndcp, vef, sdis1, sh, sd, pbt, vwl, wst, reifp), mkWSL(pbt,vwl,wst,vef,sdis1,sh,sd)))*
*(setsdis(wsid1,ndcp,sdis1,mkWSs(wsid,reifp,*
*mkWSL(pbt,vwl,wst,vef,sdis,sh,sd)))*
*= mkWSs(wsid,reifp,mkWSL(pbt,vwl,wst,vef,sdis,sh,sd))*
*IF not (wsid1==wsid))*

*(setsh(wsid,ndcp,sh, emptyWorkstations) = emptyWorkstations)*
*(setsh(wsid,ndcp,sh, (wkstns1) U (wkstns2))*
*= (setsh(wsid,ndcp,sh, wkstns1)) U (setsh(wsid,ndcp,sh, wkstns2)))*
*(setsh(wsid,ndcp,sh1,mkWSs(wsid,reifp,mkWSL(pbt,vwl,wst,vef,sdis,sh,sd )))*
*= mkWSs(wsid, updatereifp(ndcp, vef, sdis, sh1, sd, pbt, vwl,wst, reifp), mkWSL(pbt,vwl,wst,vef,sdis,sh1,sd)))*
*(setsh(wsid1,ndcp,sh1,mkWSs(wsid,reifp,*
*mkWSL(pbt,vwl,wst,vef,sdis,sh,sd )))*
*= mkWSs(wsid,reifp,mkWSL(pbt,vwl,wst,vef,sdis,sh,sd))*
*IF not (wsid1==wsid))*

*(setsd(wsid,ndcp,sd, emptyWorkstations) = emptyWorkstations)*

*(setsd(wsid,ndcp,sd, (wkstns1) U (wkstns2))*
*= (setsd(wsid,ndcp,sd, wkstns1)) U (setsd(wsid,ndcp,sd, wkstns2)))*
*(setsd(wsid,ndcp,sd1,mkWSs(wsid,reifp,mkWSL(pbt,vwl,wst,vef,sdis,sh,sd )))*
*= mkWSs(wsid, updatereifp(ndcp, vef, sdis, sh, sd1, pbt, vwl, wst, reifp), mkWSL(pbt,vwl,wst,vef,sdis,sh,sd1)))*
*(setsd(wsid1,ndcp,sd1,mkWSs(wsid,reifp,mkWSL(pbt,vwl,wst,vef,sdis,sh,sd )))*
*= mkWSs(wsid,reifp,mkWSL(pbt,vwl,wst,vef,sdis,sh,sd ))*
*IF not (wsid1==wsid))*

*(setVeffects(wsid,vef1,ndcp,emptyWorkstations) = emptyWorkstations)*
*(setVeffects(wsid,vef1,ndcp,(wkstns1) U (wkstns2))*
*= (setVeffects(wsid,vef1,ndcp,wkstns1)) U (setVeffects(wsid,vef1,ndcp,wkstns2)))*
*(setVeffects(wsid, vef1, ndcp,mkWSs(wsid,reifp,mkWSL(pbt,vwl,wst,vef,sdis,sh,sd )))*
*= mkWSs(wsid, updatereifp(ndcp, vef1, sdis, sh, sd, pbt, vwl, wst, reifp), mkWSL(pbt,vwl,wst,vef1,sdis,sh,sd)))*

*(setVeffects(wsid1,vef1,ndcp, mkWSs(wsid,reifp,mkWSL(pbt,vwl,wst,vef,sdis,sh,sd)))*
*= mkWSs(wsid,reifp,mkWSL(pbt,vwl,wst,vef,sdis,sh,sd))*
*IF not (wsid1==wsid))*

*(setrep(wsid,pli,bl,ndcp,emptyWorkstations) = emptyWorkstations)*
*(setrep(wsid,pli,bl,ndcp,(wkstns1) U (wkstns2))*
*= (setrep(wsid,pli,bl,ndcp,wkstns1)) U (setrep(wsid,pli,bl,ndcp,wkstns2)))*
*(setrep(wsid,pli,bl,ndcp,mkWSs(wsid,reifp,mkWSL(pbt,vwl,wst,vef,sdis,sh,sd)))*
*= mkWSs(wsid, updatereifp(ndcp, vef, sdis, sh, sd, pbt + (pli % bl),*
*vwl, wst, reifp),*
*mkWSL(pbt + (pli % bl), vwl, wst, vef, sdis, sh, sd)))*
*(setrep(wsid1, pli, bl, ndcp, mkWSs(wsid, reifp, mkWSL(pbt, vwl, wst, vef, sdis, sh, sd)))*
*= mkWSs(wsid, reifp, mkWSL(pbt, vwl, wst, vef, sdis, sh, sd))*
*IF not (wsid1==wsid))*

*(setviewrep(wsid, vwn, omatrix, mmatrix, ndcp, emptyWorkstations)*
*=emptyWorkstations)*
*(setviewrep(wsid,vwn,omatrix,mmatrix,ndcp,(wkstns1) U (wkstns2))*
*= (setviewrep(wsid,vwn,omatrix,mmatrix,ndcp,wkstns1)) U*
*(setviewrep(wsid,vwn,omatrix,mmatrix,ndcp,wkstns2)))*
*(setviewrep(wsid,vwn,omatrix1,mmatrix1,ndcp,mkWSs(wsid,reifp,*
*mkWSL(pbt, vwl, wst, vef, sdis, sh, sd)))*
*= mkWSs(wsid,*
*updatereifp(ndcp, vef, sdis, sh, sd, pbt, vwl + (vwn % seta(omatrix1, mmatrix1, vwl [ vwn ])), wst, reifp),*
*mkWSL(pbt, vwl + (vwn % seta(omatrix1, mmatrix1, vwl [ vwn ])),wst,vef,sdis,sh,sd)))*

*(setviewrep(wsid1, vwn, omatrix1, mmatrix1, ndcp,*
*mkWSs(wsid, reifp, mkWSL(pbt, vwl, wst, vef, sdis, sh, sd)))*
*= mkWSs(wsid, reifp, mkWSL(pbt, vwl, wst, vef, sdis, sh, sd))*
*IF not (wsid1==wsid))*
*(setviewsel(wsid, vwn, viewsel, ndcp, emptyWorkstations) = emptyWorkstations)*
*(setviewsel(wsid, vwn, viewsel, ndcp, (wkstns1) U (wkstns2))*
*= (setviewsel(wsid, vwn, viewsel, ndcp, wkstns1)) U (setviewsel(wsid, vwn, viewsel, ndcp, wkstns2)))*
*(setviewsel(wsid, vwn, viewsel1, ndcp, mkWSs(wsid, reifp,*
*mkWSL(pbt, vwl, wst, vef, sdis, sh, sd)))*
*= mkWSs(wsid, updatereifp(ndcp, vef, sdis, sh, sd, pbt,vwl + (vwn % sets(viewsel1, vwl [ vwn ])), wst, reifp),*
*mkWSL(pbt, vwl + (vwn % sets(viewsel1, vwl [ vwn ])), wst, vef, sdis, sh, sd)))*
*(setviewsel(wsid1, vwn, viewsel1, ndcp, mkWSs(wsid, reifp,*
*mkWSL(pbt, vwl, wst, vef, sdis, sh, sd)))*
*= mkWSs(wsid, reifp, mkWSL(pbt, vwl, wst, vef, sdis, sh, sd))*
*IF not (wsid1==wsid))*

*(setwwandvw(wsid,wsww,wsvw,ndcp,emptyWorkstations) = emptyWorkstations)*
*(setwwandvw(wsid, wsww, wsvw, ndcp, (wkstns1) U (wkstns2))*
*= (setwwandvw(wsid, wsww, wsvw, ndcp, wkstns1)) U (setwwandvw(wsid, wsww, wsvw, ndcp, wkstns2)))*
*(setwwandvw(wsid, mkLDCWin(mkNDCPoint(xw1, yw1),*
*mkNDCPoint(xw2, yw2)), mkDCVp(mkDCPoint(xv1, yv1),*
*mkDCPoint(xv2, yv2)), ndcp, mkWSs(wsid, reifp,*
*mkWSL(pbt, vwl, wst, vef, sdis, sh, sd)))*
*= mkWSs(wsid, updatereifp(ndcp, vef, sdis, sh, sd, pbt, vwl,*
*gettransfWS(xw1,yw1,xw2,yw2,xv1,yv1,xv2,yv2), reifp),*
*mkWSL(pbt,vwl,gettransfWS(xw1,yw1,xw2,yw2,xv1,yv1,xv2,yv2),*
*vef,sdis,sh,sd)))*

*(setwwandvw(wsid1, wsww, wsvw, ndcp, mkWSs(wsid,reifp,mkWSL(pbt, vwl,wst,vef,sdis,sh,sd)))*
*= mkWSs(wsid,reifp,mkWSL(pbt,vwl,wst,vef,sdis,sh,sd))*
*IF not (wsid1==wsid))*

**jbo**

### 3.24. GKS

This is the top object in the object hierarchy which describes the GKS-9x functions.

**obj** *GKS / GSL PICTUREPARTSTORE WORKSTATIONS*
**sorts** *gks*
**ops** *mkGKS: gsl PPS NDCPicture Workstations → gks*
*OpenGKS: → gks*
*CreateOutputPrim: ListofWCP gks → gks*
*SetPrimitiveAttribute: PlInd gks → gks*
*SetPrimitiveAttribute: asfs gks → gks*

*SetPrimitiveAttribute*: *Linetype gks* → *gks*
*SetPrimitiveAttribute*: *Linewidth gks* → *gks*
*AddSetofNamestoNameSet*: *NameSet gks* → *gks*
*RemoveSetofNamesfromNameSet*: *NameSet gks* → *gks*

*SetWindowandViewport*: *NormTran WCWin NDCVp gks* → *gks*
*SetNormTranNumber*: *NormTran gks* → *gks*

*DeletePrimitives*: *SelectCrit gks* → *gks*
*RemoveSetofNamesfromNDCP*: *NameSet SelectCrit gks* → *gks*
*AddSetofNamestoNDCP*: *NameSet SelectCrit gks* → *gks*
*ReorderNDCPicture*: *SelectCrit SelectCrit Position gks* → *gks*
*SetNDCPictureAttribute*: *SelectCrit PlInd gks* → *gks*
*SetNDCPictureAttribute*: *SelectCrit asfs gks* → *gks*
*SetNDCPictureAttribute*: *SelectCrit Linetype gks* → *gks*
*SetNDCPictureAttribute*: *SelectCrit Linewidth gks* → *gks*

*BeginPicturePart*: *PicPartName gks* → *gks*
*EndPicturePart*: *gks* → *gks*
*BeginPPartAgain*: *PicPartName gks* → *gks*
*AppendPicturePart*: *PicPartName PicPartName Matrix23 TransMode*
*Matrix23 TransMode NameSet gks* → *gks*
*RenamePicturePart*: *PicPartName PicPartName gks* → *gks*
*DeletePicturePart*: *PicPartName gks* → *gks*
*CopyPicturePartfromPPS*: *PicPartName SelectCrit Matrix23 TransMode*
*Matrix23 TransMode NameSet gks* → *gks*
*CopyNDCPicturetoPPS*: *SelectCrit PicPartName NameSet gks* → *gks*

*OpenWorkstation*: *WsId gks* → *gks*
*CloseWorkstation*: *WsId gks* → *gks*
*SetRepresentation*: *WsId PlInd PlBun gks* → *gks*
*SetView*: *WsId ViewInd Matrix23 Matrix23 gks* → *gks*
*SetViewSelCriterion*: *WsId ViewInd SelectCrit gks* → *gks*
*SetNDCVisualEffects*: *WsId VisEffSt gks* → *gks*
*SetWSWindowandViewport*: *WsId LDCWin DCVp gks* → *gks*
*SetWsSelCriterion*: *WsId SelectDisp SelectCrit gks* → *gks*
*SetWsSelCriterion*: *WsId SelectHighl SelectCrit gks* → *gks*
*SetWsSelCriterion*: *WsId SelectDet SelectCrit gks* → *gks*

*TrNT1*: → *Matrix23*
*TrNT0*: → *Matrix23*

**vars** *gks1* :*gks*
*n*: *Name*
*ns,ns1,awi,opw,acw,usn,usn1,usn2,vs,hs,ds*: *NameSet*
*p*: *SetofPolyline*
*pps,pps1,pps2*: *PPS*
*pn,pn1,pn2*: *PicPartName*
*pp*: *PicturePart*
*ndcp*: *NDCPicture*
*reifp*: *REIFPicture*
*gsl1*: *gsl*
*wkstns*: *Workstations*
*vwn*: *ViewInd*
*view*: *View*

*wksl*: *wsl*
*Selcrit, Selcrit1,sdis,sh,sd,viewsel*: *SelectCrit*
*omatrix, mmatrix*: *Matrix23*
*vef,vef1*: *VisEffSt*
*pli,pli1*: *PlInd*
*bl*: *PlBun*
*asf*: *asfs*
*ndc*: *NDCattr*
*local, global,local1, global1*: *Matrix23*
*trmode,trmode1*: *TransMode*
*ltype,ltype1*: *Linetype*
*lw,lw1*: *Linewidth*
*ntn,ntn1*: *NormTran*
*ntl*: *ListofNT*
*las,las1,was,was1*: *ASF*
*ww*: *WCWin*
*vw*: *NDCVp*
*wsww*: *LDCWin*
*wsvw*: *DCVp*
*xw1,yw1,xw2,yw2,xv1,yv1,xv2,yv2*: *int*
*lofwcp*: *ListofWCP*
*lofndcp*: *ListofNDCP*
*h*: *Highl*
*d*: *Det*
*wsid,wsid1*: *WsId*
*position*: *Position*

**eqns** (*OpenGKS*
= *mkGKS*(*mkGSL*(*emptyNS, emptyNS,*
*mktransfcor*(*ps 1,ps 0,ps 0,ps 0,ps 1,ps 0*),
*mktransfcor*(*ps 1,ps 0,ps 0,ps 0,ps 1,ps 0*),
*PLI0, asfsa*(*BUNDLED, INDIVIDUAL*), *SOLID, THIN,NTN0,*
((*NTN0 % mktransfcor*(*ps 1,ps 0,ps 0,ps 0,ps 1,ps 0*)) *U*
(*NTN1 % mktransfcor*(*ps 2,ps 0,ps 0,ps 0,ps 2,ps 0*))),
*NONAMEPN*),*emptyPPS,emptyNDCP,emptyWorkstations*))
(*CreateOutputPrim*(*lofwcp,mkGKS*(*mkGSL*(*opw,ns,local, global,pli,asf,*
*ltype, lw,ntn,ntl,NONAMEPN*),*pps,ndcp,wkstns*))
= *mkGKS*(*mkGSL*(*opw,ns,local, global,pli,asf,ltype,lw,ntn,ntl,NONAMEPN*),*pps,*
*addNDCP*(*mkSetofPolyline*(*transfLofPoints*(*lofwcp,ntl [ ntn ]*),*identifa*(*ns*), *ndca*(*local, global*),
*sourcea*(*asf,pli*), *logicala*(*ltype,lw*)), *ndcp*),
*DisplayNDCPWorkstations*(*addNDCP*(*mkSetofPolyline*(*transfLofPoints*(*lofwcp,ntl [ ntn ]*),
*identifa*(*ns*), *ndca*(*local, global*),
*sourcea*(*asf,pli*), *logicala*(*ltype,lw*)), *ndcp*), *wkstns*)))
(*CreateOutputPrim*(*lofwcp,mkGKS*(*mkGSL*(*opw,ns,local, global,pli,*
*asf,ltype, lw,ntn,ntl,pn*), *pps, ndcp, wkstns*))
= *mkGKS*(*mkGSL*(*opw,ns,local, global,pli,asf,ltype,lw,ntn,ntl,pn*),
*addtoPP*(*pn,mkSetofPolyline*(*transfLofPoints*(*lofwcp,ntl [ ntn ]*),
*identifa*(*ns*), *ndca*(*local, global*), *sourcea*(*asf,pli*),*logicala*(*ltype,lw*)),*pps*),
*ndcp,wkstns*) *IF not* (*pn == NONAMEPN*))
(*SetPrimitiveAttribute*(*pli1, mkGKS*(*mkGSL*(*opw,ns,local, global,pli,*
*asf,ltype,lw,ntn,ntl,pn*),*pps,ndcp,wkstns*))
= *mkGKS*(*mkGSL*(*opw,ns,local,global,pli1,asf,ltype,lw,ntn,ntl,pn*),*pps,ndcp,wkstns*))
(*SetPrimitiveAttribute*(*asfsa*(*las1,was1*), *mkGKS*(*mkGSL*(*opw,ns,local,*
*global,pli, asfsa*(*las,was*),*ltype,lw,ntn,ntl,pn*),*pps,ndcp,wkstns*))
= *mkGKS*(*mkGSL*(*opw,ns,local, global,pli,asfsa*(*las1,was1*),*ltype,lw,ntn,ntl,pn*), *pps,ndcp,wkstns*))

*(SetPrimitiveAttribute(ltype1,mkGKS(mkGSL(opw,ns,local,global,pli,*
*asf,ltype,lw,ntn,ntl,pn),pps,ndcp,wkstns))*
*= mkGKS(mkGSL(opw,ns,local, global,pli,asf,ltype1,lw,ntn,ntl,pn),pps,ndcp,wkstns))*
*(SetPrimitiveAttribute(lw1, mkGKS(mkGSL(opw,ns,local, global,pli,*
*asf,ltype,lw,ntn,ntl,pn),pps,ndcp,wkstns))*
*= mkGKS(mkGSL(opw,ns,local, global,pli,asf,ltype,lw1,ntn,ntl,pn),pps,ndcp,wkstns))*
*(AddSetofNamestoNameSet(ns1,mkGKS(mkGSL(opw,ns,local, global,pli,asf,ltype,lw,ntn,ntl,pn), pps,ndcp,wkstns))*
*= mkGKS(mkGSL(opw,(ns) U (ns1),local, global,pli,asf,ltype,lw,ntn,ntl,pn),pps,ndcp,wkstns))*
*(RemoveSetofNamesfromNameSet(ns1,mkGKS(mkGSL(opw,ns,local, global,pli,asf,ltype,lw,ntn,ntl,pn),pps,ndcp,wks*
*= mkGKS(mkGSL(opw,subtractset(ns1,ns),local,global,pli,asf,ltype,lw,ntn,ntl,pn),pps,ndcp,wkstns))*
*(SetWindowandViewport(ntn1,*
*mkWCWin(mkWCPoint(xw1,yw1),mkWCPoint(xw2,yw2)),*
*mkNDCVp(mkNDCPoint(xv1,yv1),mkNDCPoint(xv2,yv2)),*
*mkGKS(gsl1,pps,ndcp,wkstns))*
*= mkGKS(addnt(ntn1,mkWCWin(mkWCPoint(xw1,yw1),mkWCPoint(xw2,yw2)),*
*mkNDCVp(mkNDCPoint(xv1,yv1),mkNDCPoint(xv2,yv2)),gsl1),*
*pps,ndcp,wkstns) IF not ntn1==NTN0)*
*(SetNormTranNumber(ntn1,mkGKS(mkGSL(opw,ns,local,global,pli,asf,ltype,lw,*
*ntn,ntl,pn), pps,ndcp,wkstns))*
*= mkGKS(mkGSL(opw,ns,local,global,pli,asf,ltype,lw,ntn1,ntl,pn),pps,ndcp,wkstns))*
*(DeletePrimitives(Selcrit, mkGKS(gsl1,pps,ndcp,wkstns))*
*= mkGKS(gsl1, pps, deleteNDCP(Selcrit, ndcp),*
*DisplayNDCPWorkstations(deleteNDCP(Selcrit,ndcp),wkstns)))*
*(RemoveSetofNamesfromNDCP(ns,Selcrit,mkGKS(gsl1, pps, ndcp, wkstns))*
*= mkGKS(gsl1,pps,removenameNDCP(ns,Selcrit,ndcp),*
*DisplayNDCPWorkstations(removenameNDCP(ns,Selcrit,ndcp),wkstns)))*
*(AddSetofNamestoNDCP(ns,Selcrit,mkGKS(gsl1,pps,ndcp,wkstns))*
*= mkGKS(gsl1,pps,addnameNDCP(ns,Selcrit,ndcp),*
*DisplayNDCPWorkstations(addnameNDCP(ns,Selcrit,ndcp),wkstns)))*

*(SetNDCPictureAttribute(Selcrit, pli1,*
*mkGKS(mkGSL(opw,ns,local, global,pli,asf,ltype,lw,ntn,ntl,pn),pps,ndcp,wkstns))*
*= mkGKS(mkGSL(opw,ns,local, global,pli,asf,ltype,lw,ntn,ntl,pn),pps,*
*setattrNDCP(Selcrit,pli1,ndcp),*
*DisplayNDCPWorkstations(setattrNDCP(Selcrit,pli1,ndcp),wkstns)))*
*(SetNDCPictureAttribute(Selcrit, asfsa(las1,was1),*
*mkGKS(mkGSL(opw,ns,local, global, pli, asf,ltype, lw, ntn, ntl, pn), pps, ndcp, wkstns))*
*= mkGKS(mkGSL(opw,ns,local, global,pli,asf,ltype,lw,ntn,ntl,pn), pps,*
*setattrNDCP(Selcrit, asfsa(las1,was1), ndcp),*
*DisplayNDCPWorkstations(setattrNDCP(Selcrit,asfsa(las1,was1),ndcp),wkstns)))*
*(SetNDCPictureAttribute(Selcrit, ltype1,*
*mkGKS(mkGSL(opw,ns,local, global,pli,asf,ltype,lw,ntn,ntl,pn),pps,ndcp,wkstns))*
*= mkGKS(mkGSL(opw,ns,local, global,pli,asf,ltype,lw,ntn,ntl,pn),pps,*
*setattrNDCP(Selcrit,ltype1,ndcp),*
*DisplayNDCPWorkstations(setattrNDCP(Selcrit,ltype1,ndcp),wkstns)))*
*(SetNDCPictureAttribute(Selcrit, lw1,*
*mkGKS(mkGSL(opw,ns,local, global,pli,asf,ltype,lw,ntn,ntl,pn),pps,ndcp,wkstns))*
*= mkGKS(mkGSL(opw,ns,local, global,pli,asf,ltype,lw,ntn,ntl,pn),pps,*
*setattrNDCP(Selcrit,lw1,ndcp),*
*DisplayNDCPWorkstations(setattrNDCP(Selcrit,lw1,ndcp),wkstns)))*

*(ReorderNDCPicture(Selcrit, Selcrit1, position,mkGKS(gsl1,pps,ndcp,wkstns))*
*= mkGKS(gsl1,pps, insert(selectpicture(ndcp, Selcrit),*
*selectpicture(ndcp, snot(Selcrit)), Selcrit1, position), wkstns)*

```
        IF not (selectpicture(ndcp, sand(snot(Selcrit), Selcrit1))
          == emptyNDCP))
(ReorderNDCPicture(Selcrit, Selcrit1, position,mkGKS(gsl1,pps,ndcp,wkstns))
    = mkGKS(gsl1,pps,ndcp,wkstns)
      IF (selectpicture(ndcp, sand(snot(Selcrit), Selcrit1))
        == emptyNDCP))

(BeginPicturePart(pn1,mkGKS(mkGSL(opw,ns,local, global,pli,asf,ltype,lw,ntn,ntl,pn),
      pps,ndcp,wkstns))
    = mkGKS(mkGSL(opw,ns,local, global,pli,asf,ltype,lw,ntn,ntl,pn1),
        beginpicturepart(pn1,pps),ndcp,wkstns)
        IF (pn==NONAMEPN) and (PPisinPPS(pn1,pps)==F))
(BeginPPartAgain(pn1,mkGKS(mkGSL(opw,ns,local, global,pli,asf,ltype,lw,ntn,ntl, NONAMEPN),
      pps, ndcp, wkstns))
    = mkGKS(mkGSL(opw,ns,local, global,pli,asf,ltype,lw,ntn,ntl,pn1),
        pps,ndcp,wkstns)
        IF PPisinPPS(pn1,pps))
(EndPicturePart(mkGKS(mkGSL(opw,ns,local, global,pli,asf,ltype,lw,ntn,ntl,pn),
      pps, ndcp, wkstns))
    = mkGKS(mkGSL(opw,ns,local, global,pli,asf,ltype,lw,ntn,ntl,NONAMEPN),
        pps,ndcp,wkstns))
(AppendPicturePart(pn,pn1,global,trmode,local,trmode1,ns,mkGKS(gsl1,pps,ndcp,wkstns))
    = mkGKS(gsl1,appendpicturepart(pn,pn1,global,trmode,local,trmode1,ns,pps),ndcp,wkstns)
        IF PPisinPPS(pn,pps) and PPisinPPS(pn1,pps))
(RenamePicturePart(pn,pn1,
    mkGKS(mkGSL(opw,ns,local, global,pli,asf,ltype,lw,ntn,ntl,pn2),pps,ndcp,wkstns))
    = mkGKS(mkGSL(opw,ns,local, global,pli,asf,ltype,lw,ntn,ntl,pn2),
        renamepicturepart(pn,pn1,pps),ndcp,wkstns)
        IF not (pn==NONAMEPN) and not (pn==pn2))
(RenamePicturePart(pn,pn1,
    mkGKS(mkGSL(opw,ns,local, global,pli,asf,ltype,lw, ntn,ntl,pn),pps,ndcp,wkstns))
    = mkGKS(mkGSL(opw,ns,local, global,pli,asf,ltype,lw,ntn,ntl,pn1),
        renamepicturepart(pn,pn1,pps),ndcp,wkstns)
        IF not (pn==NONAMEPN))
(RenamePicturePart(pn,pn1,
    mkGKS(mkGSL(opw,ns,local, global,pli,asf,ltype,lw, ntn,ntl,NONAMEPN),pps,ndcp,wkstns))
    = mkGKS(mkGSL(opw,ns,local, global,pli,asf,ltype,lw,ntn,ntl,NONAMEPN),
        renamepicturepart(pn,pn1,pps),ndcp,wkstns))
(DeletePicturePart(pn,mkGKS(gsl1,pps,ndcp,wkstns))
    = mkGKS(gsl1,deletepicturepart(pn,pps),ndcp,wkstns))
(CopyPicturePartfromPPS(pn,Selcrit,global1,trmode,local1,trmode1,ns1,
    mkGKS(gsl1, pps, ndcp, wkstns))
    = mkGKS(gsl1, pps, copyPPNDCP(getpicturepart(pn,pps),Selcrit, global1,trmode,local1,trmode1,ns1,ndcp),
        DisplayNDCPWorkstations(copyPPNDCP(getpicturepart(pn,pps),
          Selcrit,global1,trmode,local1,trmode1,ns1,ndcp), wkstns))
        IF not (getPicPartName(gsl1) == pn))
(CopyNDCPicturetoPPS(Selcrit,pn,ns1,
    mkGKS(mkGSL(opw,ns,local,global,pli,asf,ltype,lw,ntn,ntl,pn1),pps,ndcp,wkstns))
    = mkGKS(mkGSL(opw,ns,local, global,pli,asf,ltype,lw,ntn,ntl,pn1),
        (mkPPS(pn,(copyNDCPPP(removenameNDCP(ns1,SELECTALL,
          selectpicture(ndcp,Selcrit))))))U(pps),ndcp,wkstns)
        IF not (pn1==pn))

(OpenWorkstation(wsid,
```

*mkGKS(mkGSL(opw,ns,local,global,pli,asf,ltype,lw,ntn,ntl,pnl),pps,ndcp,wkstns))*
*= mkGKS(mkGSL(addname(wsidtoname(wsid),opw),ns,local,global,pli,asf,ltype,lw,ntn,ntl,pnl),*
*pps,ndcp,*
*DisplayNDCPWs(wsid,ndcp,OpenWS(wsid,wkstns))))*
*(CloseWorkstation(wsid,*
*mkGKS(mkGSL(opw,ns,local,global,pli,asf,ltype,lw,ntn,ntl,pnl),pps,ndcp,wkstns))*
*= mkGKS(mkGSL(removessn(wsidtoname(wsid),opw),ns,local,global,pli,asf,ltype,lw,ntn,ntl,pnl),*
*pps,ndcp,*
*deleteWs(wsid,wkstns)))*
*(SetRepresentation(wsid,pli,bl,mkGKS(gsl1,pps,ndcp,wkstns))*
*= mkGKS(gsl1,pps,ndcp,setrep(wsid,pli,bl,ndcp,wkstns)))*
*(SetView(wsid,vwn,omatrix,mmatrix,mkGKS(gsl1,pps,ndcp,wkstns))*
*= mkGKS(gsl1,pps,ndcp,setviewrep(wsid,vwn,omatrix,mmatrix,ndcp,wkstns))*
*IF not (vwn==VWN0))*
*(SetViewSelCriterion(wsid,vwn,viewsel,mkGKS(gsl1,pps,ndcp,wkstns))*
*= mkGKS(gsl1,pps,ndcp,setviewsel(wsid,vwn,viewsel,ndcp,wkstns)))*
*(SetNDCVisualEffects(wsid,vefl,mkGKS(gsl1,pps,ndcp,wkstns))*
*= mkGKS(gsl1,pps,ndcp,setVeffects(wsid,vefl,ndcp,wkstns)))*
*(SetWSWindowandViewport(wsid,wsww,wsvw,mkGKS(gsl1,pps,ndcp,wkstns))*
*= mkGKS(gsl1,pps,ndcp,setwwandvw(wsid,wsww,wsvw,ndcp,wkstns)))*
*(SetWsSelCriterion(wsid,DISPLAY,sdis,mkGKS(gsl1,pps,ndcp,wkstns))*
*= mkGKS(gsl1,pps,ndcp,setsdis(wsid,ndcp,sdis,wkstns)))*
*(SetWsSelCriterion(wsid,HIGHLIGHTING,sh,mkGKS(gsl1,pps,ndcp,wkstns))*
*= mkGKS(gsl1,pps,ndcp,setsh(wsid,ndcp,sh,wkstns)))*
*(SetWsSelCriterion(wsid,DETECTABILITY,sd,mkGKS(gsl1,pps,ndcp,wkstns))*
*= mkGKS(gsl1,pps,ndcp,setsd(wsid,ndcp,sd,wkstns)))*

**jbo**

## 4. Conclusion

The specification presented here has been developed in parallel with the development of GKS-9x. During the course of the development, a number of omissions and ambiguities were found in the GKS-9x drafts, which were fed into the ISO/IEC review process as part of the UK comments on the drafts.

The specification has been executed using the ObjEx OBJ interpreter and test cases were used to explore the behaviour of the specification.

## References


1. K.W. Brodlie, D.A. Duce, and F.R.A. Hopgood, "The New Graphical Kernel System," *Computer-Aided Design*, 1991.
2. L.B. Damnjanovic, D.A. Duce, and S.K. Robinson, "GKS-9x: Some Implementation Considerations," *Computer Graphics Forum*, vol. 12, no. 3, 1993. In press
3. L.B. Damnjanovic, D.A. Duce, and S.K. Robinson, "GKS-9x: Implementation of Selection," contained in ERCIM Research Reports ERCIM-93-R017, ERCIM, Domaine de Voluceau, Rocquencourt, B.P. 105, F-78153 Le Chesnay Cedex, France, 1993.
4. D.A. Duce, F.R.A. Hopgood, and K.W. Brodlie, "Revision of the Graphical Kernel System," *Proceedings of the Icographics '91 Conference*, 1991.
5. D.A. Duce and L.B. Damnjanovic, "Formal Specification in the Revision of GKS: An Illustrative Example," *Computer Graphics Forum*, vol. 11, no. 1, pp. 17-30, 1992.
6. D.A. Duce and F. Paterno, "A Formal Specification of a Graphics System in the Framework of the Computer Graphics Reference Model," *Computer Graphics Forum*, vol. 12, no. 1, pp. 3-20, 1993.
7. —, "Information processing systems - Computer graphics - Graphical Kernel System (GKS) functional description," ISO 7942, ISO Central Secretariat, August 1985.

8. —, "Information processing systems - Computer graphics - Programmer's Hierarchical Interactive Graphics System functional description," ISO/IEC 9592: 1, 1989.

9. —, "Information processing systems - Computer graphics - Computer Graphics Reference Model," ISO/IEC IS 11072, ISO Central Secretariat, In press.